UNIVERSITY OF NOVI SAD
FACULTY OF TECHNICAL SCIENCES
NOVI SAD

Vladimir Dimitrieski

Model-Driven Technical Space Integration Based on a Mapping Approach

- Ph.D. Thesis -

Supervisors
Ivan Luković, PhD, full professor
Sonja Ristić, PhD, associate professor

Novi Sad, December 2017

Vladimir Dimitrieski: *Model-Driven Technical Space Integration Based on a Mapping Approach,* © December 2017

SERBIAN TITLE:
Pristup integraciji tehničkih prostora zasnovan na preslikavanjima i inženjerstvu vođenom modelima

SUPERVISORS:
Ivan Luković, PhD, full professor
Sonja Ristić, PhD, associate professor

LOCATION:
Novi Sad

DATE:
December 2017

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:	
Идентификациони број, **ИБР**:	
Тип документације, **ТД**:	Монографска документација
Тип записа, **ТЗ**:	Текстуални штампани материјал
Врста рада, **ВР**:	Докторска дисертација
Аутор, **АУ**:	Владимир Димитриески
Ментори, **МН**:	др Иван Луковић, редовни професор и др Соња Ристић, ванредни професор
Наслов рада, **НР**:	Приступ интеграцији техничких простора заснован на пресликавањима и инжењерству вођеном моделима
Језик публикације, **ЈП**:	енглески
Језик извода, **ЈИ**:	српски
Земља публиковања, **ЗП**:	Србија
Уже географско подручје, **УГП**:	Војводина
Година, **ГО**:	2017
Издавач, **ИЗ**:	Факултет техничких наука
Место и адреса, **МА**:	Трг Доситеја Обрадовића 6, 21000 Нови Сад
Физички опис рада, **ФО**: (поглавља/страна/цитата/табела/слика/графика/прилога)	7/194/204/4/39/0/0
Научна област, **НО**:	Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:	Интеграција система, развој софтвера вођен моделима, трансформације модела, наменски језици, технички простори
УДК	
Чува се, **ЧУ**:	Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, 21000 Нови Сад
Важна напомена, **ВН**:	
Извод, **ИЗ**:	За потребе повећања степена аутоматизације развоја адаптера за интеграцију у индустријском окружењу, осмишљен је моделом вођен приступ развоју адаптера. У оквиру овог приступа развијен је наменски језик за спецификацију пресликавања између техничких простора који су предмет интеграције. Приступ обухвата и алгоритме за поравнање и поновно искориштење претходно креираних пресликавања са циљем аутоматизације процеса спецификације. На основу креираних пресликавања, могуће је аутоматски генерисати извршиви код адаптера. У испитивањима приступа, показано је да је могуће успешно применити моделом вођен приступ у интеграцији техничких простора као и да је могуће успешно повећати степен аутоматизације поновним искоришћењем претходно креираних пресликавања.
Датум прихватања теме, **ДП**:	22.12.2016.
Датум одбране, **ДО**:	

Чланови комисије, **КО**:	Председник:	др Гордана Милосављевић, ванредни професор	
	Члан:	др Марјан Мерник, редовни професор	
	Члан:	др Славица Кордић, доцент	Потпис ментора
	Члан, ментор:	др Соња Ристић, ванредни професор	
	Члан, ментор:	др Иван Луковић, редовни професор	

	UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES** 21000 NOVI SAD, Trg Dositeja Obradovića 6
	KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monograph documentation
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Ph.D. thesis
Author, **AU**:	Vladimir Dimitrieski
Mentors, **MN**:	Ivan Luković, Ph.D., Full Professor and Sonja Ristić, Ph.D., Associate Professor
Title, **TI**:	Model-Driven Technical Space Integration Based on a Mapping Approach
Language of text, **LT**:	English
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2017
Publisher, **PB**:	Faculty of Technical Sciences
Publication place, **PP**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/194/204/4/39/0/0
Scientific field, **SF**:	Electrical engineering and computing
Scientific discipline, **SD**:	Applied computer science and informatics
Subject/Key words, **S**/**KW**:	System Integration, Model-Driven Software Development, Model Transformations, Domain-Specific Languages, Technical Spaces
UC	
Holding data, **HD**:	Library of Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad
Note, **N**:	
Abstract, **AB**:	In order to automate development of integration adapters in industrial settings, a model-driven approach to adapter specification is devised. In this approach, a domain-specific modeling language is created to allow specification of mappings between integrated technical spaces. Also proposed is the mapping automation engine that comprises reuse and alignment algorithms. Based on mapping specifications, executable adapters are automatically generated and executed. Results of approach evaluations indicate that it is possible to use a model-driven approach to successfully integrate technical spaces and increase the automation by reusing domain-specific mappings from previously created adapters.
Accepted by the Scientific Board on, **ASB**:	22.12.2016.
Defended on, **DE**:	

Defended Board, **DB**:	President:	Gordana Milosavljević, Ph.D., Associate Professor	
	Member:	Marjan Mernik, Ph.D., Full Professor	
	Member:	Slavica Kordić, Ph.D., Docent	Menthor's sign
	Member, Mentor:	Sonja Ristić, Ph.D., Associate Professor	
	Member, Mentor:	Ivan Luković, Ph.D., Full Professor	

Obrazac **Q2.HA.06-05**- Izdanje 1

DEDICATION

This thesis is dedicated to
My wonderful parents who have raised me to be the person I am today
My wife who taught me patience and provided me motivation
My friends for supporting me on this journey

ABSTRACT

In the resent years, a new paradigm shift called Industry 4.0 is happening. A plethora of new "smart" devices and software systems is constantly created thus requiring integration in the existing machine landscapes. As industrial systems need to operate without interruptions and production process changes when introducing new elements, seamless integration is considered as a main enabler of a modern manufacturing process. In contemporary manufacturing systems in which we have a lack of a widely accepted and used communication standard, integration is addressed by the manual development of integration adapters. This is a time-consuming, error-prone, costly, and a tedious task overall. In order to automate development of integration adapters in industrial settings, we have devised a model-driven approach to adapter specification. This approach comprises three main concepts: (i) a domain-specific modeling language, (ii) high-level reuse of existing adapters, and (iii) code generation. A domain-specific modeling language is created to allow specification of mappings between integrated technical spaces. The language provides integration domain experts a set of domain-specific concepts that allow the creation high-level mappings between technical space elements being integrated. Such mappings are considered as atomic units of integration and as such may be easily reused in new adapter specifications. To increase the specification process automation even further, we propose a mapping automation engine that comprises reuse and alignment algorithms. Former algorithms use previously created mappings in order to identify correspondences in the current integration scenario, while latter algorithms identify correspondences by considering technical space elements just from the current use case. Once a specification is completed, either manually or automatically, executable adapters are automatically generated for an execution platform via code generators. The approach has been applied and analyzed in both industrial and non-industrial software integration scenarios. Results of approach analysis indicate that it is possible to use a model-driven approach to successfully integrate technical spaces and increase the automation by reusing domain-specific mappings from previously created adapters.

keywords: system integration, model-driven software development, model transformations, domain-specific languages, technical spaces

REZIME

Zajednički cilj inovacija koje su inicirale tri industrijske revolucije bilo je povišenje stepena automatizacije proizvodnog procesa, a time i efikasnosti proizvodnje i kvaliteta proizvoda. Pogled na automatizaciju izmenio se u poslednjoj deceniji usled pojave vizija kao što su Industrija 4.0 (nem. *Industrie 4.0* – I4.0) i Internet stvari (eng. *Internet of Things* – IoT). Predmet ovih vizija jesu tzv. sajber-fizički (SF) sistemi (eng. *Cyber-Physical Systems*). SF sistemi su automatizovani sistemi koji razmenjuju veliku količinu podataka u realnom vremenu kako bi organizovali i izvršavali akcije u poslovnom ili proizvodnom procesu. Sastoje se od fizičkog dela, uređaja, i sajber tj. virtualnog dela koji predstavlja virtualnu reprezentaciju uređaja. Pošto ovakvi sistemi imaju široku upotrebu u industriji proizvodnje dobara, integracija SF sistema jedan od ključnih elemenata u automatizaciji, ali trenutni pristupi integraciji ne zadovoljavaju visoke standarde industrije u pogledu fleksibilnosti i prilagodljivosti rešenja.

Proizvođači informacionih sistema i SF sistemima koji se koriste u industriji suočavaju se sa problemom integracije svojih rešenja sa postojećim SF sistemima. U fabrikama su često prisutni heterogeni sistemi, tj. sistemi od različitih proizvođača koji podržavaju različite standarde ili prilagođavaju postojeće standarde svojim potrebama, domenu ili konkretnom uređaju. Usled ovakve heterogenosti sistema, integracija SF i informacionih sistema postaje jedan od najznačajnijih problema koje treba rešiti. Integracija se najčešće obavlja ručno i skoro po pravilu je kompleksna, teška za praćenje i zahteva veliku količinu vremena i novca. Ovo nije iznenađujuće s obzirom na to da je i u tradicionalnoj industriji (Industriji 3.0), u kojoj integracija nije imala toliki značaj kako je to danas slučaj, oko 40% kompanijskog budžeta bilo trošeno na integraciju [26].

U okviru koncepta Industrije 4.0, SF sistemi su na fizičkom nivou povezani komunikacionim medijumima koji su priključeni na jedan od komunikacionih interfejsa sistema. Svi interfejsi koji učestvuju u komunikaciji, preko komunikacionih medijuma primaju i šalju podatke uređene prema određenom skupu pravila koji se naziva format ili šema podataka. Pod pojmom šema podataka (eng. *data schema*) podrazumevamo pravila uređenja podataka u širem smislu, bilo da se radi o šemi relacije koja opisuje strukturu relacije u koju se upisuju podaci u relacionoj bazi podataka, ili *eXtensible Markup Language* (XML) šemi koja opisuje strukturu XML dokumenata. Svaka od šema podataka pripada širem pojmu koji nosi naziv tehnički prostor. Tehnički prostor (eng. *technical space*) može se posmatrati kao „radni kontekst koji obuhvata skup pridruženih koncepata, znanja, resursa, potrebnih veština i alata" [40, 116], a u okviru kojeg je moguće razumeti i obrađivati kako šeme podataka tako i same podatke. Kako bi SF sistem mogao da razmenjuje a pri tome i da razume podatke koje mu drugi SF sistem šalje, potrebno je ta dva sistema integrisati. U ovoj disertaciji, integraciju SF sistema posmatramo kao kreiranje pravila transformacije između elemenata šema podataka iz odgovarajućih tehničkih prostora u okviru kojih SF sistemi razmenjuju podatke. Softverske komponente u kojima su implementirana pravila integracije SF sistema nazivamo adapterima za integraciju ili samo adapterima. Trenutno, adapteri su najčešće ručno razvijani za svaku kombinaciju tehničkih prostora. U Industriji 4.0 postoji veliki broj SF i informacionih sistema koji egzistiraju u različitim tehničkim prostorima i zbog toga je razvoj adaptera naporan i dug proces koji iziskuje upotrebu značajnih ljudskih i finansijskih resursa.

Problem u kojem postoji neusaglašenost brojnih tehničkih prostora, nazivamo međuprostorna heterogenost ili, prema [197], heterogenost modela podataka. Pored međuprostorne heterogenosti postoji i problem prostorne heterogenosti, tj. strukturalne heterogenosti kako je

nazvana u [197]. Ovaj problem se ogleda u činjenici da čak i kada su dva tehnička prostora integrisana adapterom, šema podataka prema kojoj uređaj u okviru SF sistema šalje podatke, može biti izmenjena u zavisnosti od faktora kao što su konfiguracija uređaja, verzija uređaja ili verzija procesa proizvodnje u okviru SF sistema u kojem se uređaj koristi. Prostorna heterogenost uvodi dodatnu složenost u proces ručne implementacije adaptera, jer oni moraju biti prilagodljivi i dovoljno robusni kako bi se lako automatski prilagodili na izmene unutar povezanih tehničkih prostora. U suprotnom, neophodne su nove, ručne izmene u kôdu adaptera u cilju njihovog prilagođenja novonastaloj situaciji.

Postojanje prethodno navedenih problema heterogenosti u velikoj meri usporava razvoj adaptera i indirektno može da naruši performanse celog proizvodnog procesa. Postojeći pristupi dominantno se danas oslanjaju na ručnu implementaciju adaptera u programskim jezicima opšte namene koji su neprimerenog nivoa apstrakcije za dati problem. Stoga, cilj ovog istraživanja je da se omogući efikasniji razvoj adaptera za integraciju heterogenih tehničkih prostora, razvojem radnog okvira za integraciju koji će rešiti ili bar ublažiti probleme izazvane heterogenošću tehničkih prostora.

Predmet ovog istraživanja je kreiranje radnog okvira (eng. *framework*) za integraciju tehničkih prostora korišćenjem principa razvoja softvera vođenog modelima (RSVM). U razvoju softvera vođenog modelima (eng. *Model-Driven Software Development* – MDSD) modeli ne predstavljaju samo pasivnu dokumentaciju softvera, već su i formalna osnova njegovog razvoja. Mentalni modeli nastaju kao rezultat procesa apstrakcije koji obuhvata identifikaciju, grupisanje i generalizaciju elemenata iz realnog sveta, kao i zanemarivanje svih njihovih osobina koje nisu suštinski bitne za trenutno posmatrani problem. Mentalni model može biti zapisan putem jezika za modelovanje. Takvi jezici poseduju notaciju koja omogućava vernu predstavu koncepata iz domena za koji se model kreira.

Razvoj i primena jezika za modelovanje prati četvoronivovsku arhitekturu [17, 39] čiji su glavni nivoi definisani u redosledu koji odgovara rastućem nivou apstrakcije: (M0) nivo entiteta u posmatranom delu realnog sveta odnosno sistema koji se posmatra, (M1) nivo modela entiteta koji je kreiran pomoću jezika za modelovanje, (M2) nivo meta-modela koji obuhvata apstrahovane osobine domena primene, a predstavlja i apstraktnu sintaksu jezika za modelovanje, (M3) nivo meta-meta-modela koji obuhvata domenski nezavisne koncepte za kreiranje meta-modela. Jezik za modelovanje koji obuhvata koncepte bliske određenom domenu primene naziva se namenski jezik za modelovanje (NJM) (eng. *Domain Specific Modeling Language* – DSML). Prednosti NJM u odnosu na jezike za modelovanje opšte namene koji sadrže generičke koncepte za modelovanje primenljive u bilo kojem domenu, višestruke su. Koristeći NJM domenski ekspert može da kreira modele pomoću koncepata koji su njemu bliski i koji su prilagođeni posmatranom domenu primene. Samim tim, korisnik jezika može u domenu problema da kreira modele i softverska rešenja brže, jednostavnije i sa manje grešaka nego što je slučaj kada se koriste jezici za modelovanje opšte namene. Operacije kao što su analiza, simulacija, optimizacija, paralelizacija i verifikacija rešenja u problemskom domenu obavljaju se nad konceptima bliskim samom domenu, što u mnogome poboljšava efikasnost sprovođenja ovih operacija [103, 114, 140].

Nakon kreiranja, nijedan model ne može opstati kao večno nepromenljiv ili izolovan. Brojne operacije mogu se vršiti nad modelima u cilju sprovođenja izmena, ali i premošćavanja razlika između različitih modela koji sa različitih tačaka gledišta modeluju iste entitete realnog sveta. Operacije nad modelima nazivaju se i transformacijama modela. Transformacije modela specificiraju se na nivou meta-modela, a izvršavaju se na nivou modela.

Istraživanje koje se predlaže u ovoj disertaciji usmereno je na probleme premošćavanja razlika i kreiranja transformacija između različitih tehničkih prostora, što zapravo predstavlja integraciju tehničkih prostora. Posebna pažnja biće posvećena onim tehničkim prostorima čiji je kontekst industrijsko okruženje u kojem SF i informacioni sistemi razmenjuju podatke definisane prema postojećoj šemi podataka. Pošto su adapteri za integraciju koje mi posmatramo u ovoj disertaciji softverske komponente, tehničke prostore možemo posmatrati kao tronivovske strukture uzevši u obzir samo softverske (digitalne) nivoe M1-M3. Na taj način, svi principi i dobre osobine RSVM mogu se iskoristiti i u rešavanju problema integracije takvih tehničkih prostora. Time se integracija tehničkih prostora svodi na kreiranje i primenu transformacija nad datim tehničkim prostorima. Pristup koji se u ovoj doktorskoj disertaciji predlaže omogućiće postupak kreiranja takvih specifikacija transformacija iz kojih se mogu generisati adapteri čiji je zadatak da automatski prevode modele iz jednog tehničkog prostora u drugi i obratno, čime se obezbeđuje njihova integracija. Očekuje se da će automatski generisani adapteri zameniti ručno pisane adaptere i dovesti do uštede vremena i novca, smanjenja kompleksnosti adaptera, kao i uloženog napora u njihov razvoj i održavanje.

U kontekstu prethodno uvedenih pojmova i problema, možemo uvesti i osnovnu hipotezu istraživanja opisanog u ovom radu.

Hipoteza 0: *moguće je razviti namenski jezik za modelovanje i pristup zasnovan na principima RSVM čijom primenom bi bilo moguće rešiti probleme heterogenosti u integraciji tehničkih prostora.*

Da bi predloženi NJM bilo moguće koristiti za rešavanje problema heterogenosti tehničkih prostora u realnim, industrijskim domenima, potrebno je da zadovoljava dva zahteva:

1. NJM mora omogućiti integraciju bilo koja dva tehnička prostora, a koncepti namenskog jezika moraju biti lako razumljivi ekspertima u tehničkim prostorima koji se integrišu; i
2. koncepti NJM trebalo bi da u što većoj meri podržavaju automatizaciju ponovnog iskorišćenja postojećih adaptera za integraciju.

Zadovoljenje prvog zahteva omogućilo bi korićenje jezika u mnogim domenima i tehničkim prostorima na jedinstven način. Da bi to bilo moguće, potrebno je uvesti jedinstvenu predstavu svih šema podataka. Odgovarajuća generička struktura za jedinstveno predstavljanje šema podataka može biti struktura tipa grafa jer je svaka šema podataka sastavljena od tipova entiteta, tipova poveznika kao i njihovih osobina što odgovara osnovnim konceptima grafa: čvorovima i granama. Osim jedinstvene predstave različitih šema podataka, takva struktura tipa grafa mora u svojim čvorovima sadržati i veze sa originalnim elementima šeme podataka na osnovu kojih su generički elementi kreirani. Ovo je neophodno zbog izvršavanja generisanih adaptera jer se izvršavaju nad podacima koji su uređeni prema originalnoj šemi podataka. Za razliku od izvršavanja adaptera, njihova specifikacija odvija na višem nivou apstrakcije, pomoću NJM i nad generičkom strukturom tipa grafa. Stoga, možemo definisati prvu izvedenu hipotezu.

Hipotezu 1: *moguće je kreirati univerzalnu strukturu tipa grafa, pomoću koje će biti predstavljene šeme podataka (meta-modeli) iz tronivovskih tehničkih prostora, a koja će sadržati i veze sa originalnim elementima šema podataka.*

Nakon uspostavljanja generičke strukture za predstavljanje tehničkih prostora moguće je pristupiti razvoju NJM za integraciju tehničkih prostora. Kako je jedan od osnovnih motiva za kreiranje NJM povećanje nivoa apstrakcije, potrebno je kreirati jezik koji omogućava što lakše i intuitivnije kreiranje preslikavanja između generički predstavljenih elemenata izvornog i odredišnog tehničkog prostora. Specifikacije preslikavanja, koje su kreirane na visokom nivou apstrakcije, mogu pretstavljati ulaz za proces generisanja izvršnog kôda adaptera za željenu izvršnu platformu. U skladu sa prethodnim opisom procesa kreiranja adaptera, uvodimo drugu izvedenu hipotezu.

Hipoteza 2: *moguće je kreirati namenski jezik koji će omogućiti kreiranje preslikavanja, na visokom nivou apstrakcije između generičkih predstava šema podataka, koja mogu biti iskorišćena za generisanje izvršivih adaptera za integraciju.*

Drugi zahtev koji NJM mora zadovoljiti, a koji se tiče automatizacije ponovnog iskorišćenja specifikacija, najviše se odnosi na automatizaciju procesa kreiranja preslikavanja u prisustvu prostorne heterogenosti. Razvoj radnog okvira za ponovno iskorišćenje u mnogome bi učinio razvoj NJM za integraciju složenijim, ali bi istovremeno doveo do povećanja stepena automatizacije celog procesa kreiranja adaptera. Ovakav radni okvir bi prvenstveno bio upotrebljiv u prisustvu prostorne heterogenosti ali bi ga takođe bilo moguće primeniti i u prisustvu međuprostorne heterogenosti. Očekuje se da bi ovakav radni okvir bio posebno primenjiv u domenu industrijske proizvodnje u kojem je prostorna heterogenost često prisutna. Male promene u šemama podataka izazvane su promenom konfiguracije SF sistema između različitih slučajeva korišćenja kao i primenom različitih metoda merenja i regulacije. Ovakve izmene su obično dovoljno male da se novi adapteri mogu automatski kreirati na osnovu prethodno kreiranih adaptera. Stoga, može biti definisana i naredna izvedena hipoteza koja se tiče ponovnog iskorišćenja preslikavanja.

Hipoteza 3: *moguće je kreirati proširiv radni okvir za ponovno iskorišćenje prethodno kreiranih specifikacija adaptera u prisustvu prostorne heterogenosti, čije izvršenje je zasnovano na konceptima namenskog jezika za integraciju tehničkih prostora.*

Na osnovu uvedenih hipoteza možemo formulisati zadatke istraživanja. Jedan od zadataka istraživanja je formulisanje pristupa za rešavanje problema heterogenosti koji postoje u integraciji tehničkih prostora korišćenjem principa RSVM i namenskih jezika za modelovanje. Osim istraživačkog aspekta, drugi zadatak ovog istraživanja obuhvata i kreiranje alata za primenu pristupa i njegovu primenu u kontekstu industrijske proizvodnje ali i šire, u kontekstu integracije softverskih rešenja. Očekuju se tri tipa doprinosa:

- *Teorijski doprinosi* u oblasti integracije tehničkih prostora vođene modelima, koji obuhvataju:
  - proučavanje i analizu postojećih pristupa i alata za integraciju,
  - primenu RSVM u domenu integracije tehničkih prostora,
  - identifikaciju osnovnih koncepata namenskog jezika za modelovanje u domenu integracije tehničkih prostora,
  - konceptualizaciju proširivog radnog okvira za ponovno iskorišćenje prethodno kreiranih adaptera u domenu industrijske proizvodnje i
  - formulacija metodološkog pristupa za primenu razvijenog namenskog jezika i radnog okvira za integraciju.
- *Doprinos razvoju* u vidu razvoja softverskog rešenja za integraciju tehničkih prostora koje će omogućiti primenu formulisanog pristupa za integraciju tehničkih prostora. Ovo rešenje uključuje namenski jezik za modelovanje i radni okvir za ponovno iskorišćenje prethodno specificiranih adaptera za integraciju.
- *Doprinos primeni* obuhvata primenu formulisanog pristupa za integraciju tehničkih prostora na nekoliko studija slučaja i ocenu pristupa na osnovu postignutih rezultata.

Glavni očekivani rezultat istraživanja jeste jednostavnija i lakša integracija tehničkih prostora sa ciljem skraćenja vremena potrebnog da se odgovori na promene poslovnog i proizvodnog procesa u poslovanju i da se reše problemi međuprostorne i prostorne heterogenosti.

Očekivani krajnji korisnici ovog rešenja su stručnjaci u domenu integracije u industrijskoj proizvodnji kao i inženjeri koji proizvode softverska i hardverska rešenja za industriju i koji žele da integrišu svoje SF sisteme u postojeće sisteme. Takođe, kako je pojam tehničkog prostora širok i ne odnosi se samo na domen industrije, očekivani korisnici su i softverski inženjeri koji žele da integrišu softverska rešenja koja ne poseduju adekvatan mehanizam za međusobnu razmenu podataka.

Detaljnom analizom literature (Poglavlje 3) i postojećih alata za integraciju tehničkih prostora (Poglavlje 4), uspostavljena je teorijska osnova za razvoj pristupa integraciji tehničkih prostora zasnovanog na RSVM.

Analizom literature, identifikovani su različiti pristupi integraciji tehničkih prostora, posebno u domenu industrijske proizvodnje. Ovi pristupi mogu se podeliti u dve kategorije: pristupi integraciji zasnovani na standardima i pristupi integraciji zasnovani na transformacijama.

Pristupi integraciji zasnovani na standardima obuhvataju kreiranje standardnih mehanizama za integraciju kao što su standardni komunikacioni protokoli i interfejsi. Neki od najpoznatijih savremenih standarda u domenu industrijske integracije su: *Open Platform Communications Unified Architecture* (OPC UA), *Automation Markup Language* (AML) i *Reference Architecture Model for Industry 4.0* (RAMI 4.0). Iako se standardizacija smatra najboljim i najefikasnijim pristupom integraciji, kompanije često prilagođavaju standarde svojim potrebama iz tehnoloških i poslovnih razloga. Takođe, postoji veliki broj standarda izdat od strane različtih tela za standardizaciju, tako da primena različitih standarda opet može voditi ka otežanoj integraciji različitih tehničkih prostora. Iz prethodno navedenih razloga pojavljuje se i potreba za kreiranjem pristupa integraciji zasnovanom na trasnformacijama, čiji je zadatak da obezbedi premošćavanje razlika u postojećim standardima.

U slučajevima kada kompanija ne podržava komunikacioni standard ona može razviti sopstveni protokol za komunikaciju između SF sistema. U tom slučaju, kompanija mora obezbediti i adaptere za integraciju koji će omogućiti komunikaciju njenih SF sistema sa SF sistemima ostalih proizvođača. Kompanije u ovu svrhu koriste pristupe integraciji zasnovane na transformacijama. Ovakvi pristupi obuhvataju veliki broj različitih mehanizama koji se koriste i izvan domena integracije i industrijske proizvodnje. Mehanizmi koji se koriste u integraciji mogu se kategorizovati u dve grupe: (i) mehanizmi za konsolidaciju i preslikavanje šema podataka i (ii) mehanizmi za konsolidaciju i poravnanje ontologija.

Mehanizmi za konsolidaciju i preslikavanje šema podataka potiču iz domena baza podataka. Ovakvi mehanizmi mogu se efikasno primeniti i u domenu integracije u industriji jer su svi podaci koji se razmenjuju formatirani prema implicitno ili eksplicitno definisanoj šemi podataka. Nažalost, najveći broj ovih pristupa primenjen je upravo u domenima integracije šema relacionih baza podataka i integracije XML šema dokumenata. Ograničena primena pristupa je posledica tradicionalnog viđenja domena relacionih baza podataka i XML dokumenata kao najpogodnijih domena za testiranje algoritama za konsolidaciju i preslikavanje šema podataka. Mehanizmi za konsolidaciju i preslikavanje šema podataka zasnovani na principima RSVM predstavljaju podskup prethodno navedenih mehanizama. Od značaja su, jer koriste iste principe kao i ovde predloženo istraživanje. U RSVM pristupima namenski jezici za modelovanje koriste se kako bi korisnik mogao da kreira preslikavanja između različitih šema podataka. Kao i u prethodnom slučaju, većina identifikovanih pristupa doživela je upotrebu u domenima relacionih baza podataka i XML dokumenata. Mehanizmi za poravnanje i konsolidaciju ontologija najviše su zastupljeni u domenu semantičkog veba. Znanje o domenu ili posmatranom sistemu predstavljeno je pomoću ontologija. Pošto različite osobe istu bazu znanja mogu predstaviti na različite načine, postoji potreba za konsolidovanjem znanja kao i poravnanjem ontologija postavljenih na različite načine. Ovakva heterogenost i postojanje višestrukih ontologija za isti slučaj korišćenja nisu pogodni za industrijsku primenu usled svoje kompleksnosti i uvođenja dvosmislenosti u domen primene.

Većina identifikovanih pristupa i alata razvijenih kao podrška ovim pristupima razvijeni su do nivoa prototipa. Samo nekoliko uočenih pristupa i alata se i dalje razvija i upotrebljeno je i izvan domena integracije šema relacionih baza podataka i integracije XML šema dokumenata. Jedan on najznačajnijih ovakvih alata jeste *PROTOtype PLAtform for Schema Matching* (PROTOPLASM), koji je opisan u radu [27]. Ovaj alat sastavljen je od tri modula koja odgovaraju koracima istoimenog pristupa: (i) modul za predstavljanje tehničkih prostora koji se integriše u vidu univerzalne strukture zasnovane na jeziku XML, (ii) modul za specifikaciju pravila za konsolidaciju šema podataka i (iii) modul za grafičku predstavu pravila konsolidacije i njihovo kombinovanje u izvršivu specifikaciju konsolidacionog algoritma. Pristup PROTOPLASM je poslužio kao direktna inspiracija za pristup koji je predstavljen u ovom istraživanju.

Nakon pregleda literature zaključeno je da je malo broj identifikovanih pristupa i alata dostigao potreban nivo zrelosti da može biti primenjen u integraciji tehničkih prostora u domenu industrijske proizvodnje. Osim pregleda naučnih radova, za ovo istraživanje od velikog značaja su i softverska rešenja koja omogućavaju kreiranje adaptera za integraciju tehničkih prostora, a koja su spremna za upotrebu u domenu industrijske proizvodnje.

Pretragom po definisanim ključnim rečima kao i pretragom radova i izloženih rešenja na sajmovima industrijske automatizacije, identifikovano je 18 softverskih alata koji su ispitani u okviru istraživanja. U svim alatima je implementiran isti primer koji obuhvata integraciju *Comma Separated Values* (CSV) i XML tehničkih prostora (Poglavlje 4.1), tj. integraciju senzora koji šalje podatke u CSV obliku i informacionog sistema koji prima XML dokumente. Primer je osmišljen kako bi pružio osobi koja koristi alat što bolji uvid u funkcionalnosti i osobine samog alata, a da pri tome bude jednostavan za razumevanje i implementaciju. Cilj analize alata nije bio merenje performansi kreiranih adaptera za integraciju niti identifikacija svih mogućih funkcionalnosti. Cilj analize je bio da se utvrde osobine alata u odnosu na 10 funkcionalnih i nefunkcionalnih osobina uvedenih u Poglavlju 4.1. Ove osobine obuhvataju: način distribucije aplikacije, ažurnost softvera, domen primene, korišćeni pristip kreiranju preslikavanja, korišćeni jezik za kreiranje preslikavanja, korišćeni jezik za specifikaciju izvršavanja preslikavanja, korišćeni pristup generisanju kôda i njegovom izvršavanju, nivo ponovne iskoristivosti koncepata specifikacije preslikavanja, mogućnost povećanja broja podržanih tehničkih prostora i mogućnost proširenja funkcinalnosti alata.

Za svaki alat i za svaku osobinu utvrđena je odgovarajuća vrednost. Na osnovu rezultata analize alata, za svaku osobinu utvrđena je vrednost koja se pojavljuje kod najvećeg broja alata i tu vrednost nazivamo najčešća vrednost osobine. Sve najčešće vrednosti osobina alata su poslužile za formulaciju osobina teorijskog, generičkog alata za integraciju tehničkih prostora. Po uzoru na taj alat implementiran je alat AnyMap koji služi za podršku pristupu opisanom u ovoj disertaciji. Naš cilj je da pružimo korisnicima alat na koji se nije teško navići i koji poseduje većinu osobina kao i trenutno korišćeni alati. Generički alat se može opisati kao alat koji:

- poseduje komercijalnu licencu,
- distribuiran je kao desktop aplikacija,
- namenjen je za rešavanje problema u domenu integracije podataka,
- omogućava neposredno kreiranje preslikavanja između dva tehnička prostora bez upotrebe posredničkog tehničkog prostora,
- pruža grafičku konkretnu sintaksu jezika za kreiranje preslikavanja,
- pruža tekstualnu konkretnu sintaksu jezika za specifikaciju izvršavanja preslikavanja,

- omogućava izvršenje preslikavanja u okviru alata kao i generisanje adaptera koji se izvršavaju nezavisno od alata,
- omogućava uvođenje podrške za nove tehničke prostore kada slučaj korišćenja to zahteva i
- omogućava proširenje jezika za specifikaciju izvršavanja preslikavanja i uvođenje novih generatora koda.

Jedan od zaključaka analize alata jeste da su alati sa ovim konkretnim osobinama pružali najveću fleksibilnost i najveći broj opcija prilikom rešavanja problema u domenu integracije podataka. Iako je naš alat zasnovan na ovim osobinama, postoje neke razlike između pristupa integraciji opisanog u ovom radu i pristupa koji je podržan analiziranim alatima.

Prva razlika se ogleda u tome što je naš pristup zasnovan na principima RSVM. Kao takav, pristup predviđa postojanje više konkretnih sintaksi jezika za kreiranje preslikavanja. Drugim rečima, RSVM promoviše kreiranje jedinstvene apstraktne sintakse jezika i postojanje višestrukih konkretnih sintaksi prilagođenih različitim upotrebama ovog jezika. Većina analiziranih alata je pružala isključivo jednu konkretnu sintaksu jezika za kreiranje preslikavanja, bez eksplicitne specifikacije apstraktne sintakse. Ovo dovodi do problema u primeni jezika u različitim slučajevima korišćenja. Na primer, grafička konkretna sintaksa je odgovarajuća za šeme podataka male i srednje kompleksnosti dok je tabularna konkretna sintaksa odgovarajuća za složene šeme podataka. Posedovanje višestrukih konkretnih sintaksi dovodi do povećanja produktivnosti inženjera kao i do mogućnosti primene pristupa u velikom broju domena. U ovom radu je implementiran alat *AnyMap* koji obuhvata samo grafičku konkretnu sintaksu jezika za specifikaciju preslikavanja. Iako je apstraktna sintaksa eksplicitno definisana i dodavanje nove sintakse zahteva isključivo kreiranje novih vizuelnih elemenata, za *AnyMap* je odabrana jedna sintaksa zbog slučajeva korišćenja koji su niske i srednje kompleksnosti. U takvim slučajevima, grafička konkretna sintaksa se pokazala kao najpogodnija. Kako bi problem prenatrpanih dijagrama koji je uočen u pojedinim alati kao što je *Altova MapForce* bio ublažen, jezik za specifikaciju izvršavanja preslikavanja ima tekstualnu konkretnu sintaksu. Ovako izbegavamo situaciju u kojoj oba jezika imaju grafičku konkretnu sintaksu i da čak i sa malim brojem preslikavanja dijagram postane nečitljiv.

Od svih analiziranih alata, samo su *Vorto* i *Open Mapping Software* zasnovani na principima RSVM. Osnovna razlika u odnosu na pristup predstavljen u ovom radu je način integracije dve ili više šema podataka u različitim tehničkim prostorima. U oba pristupa, integracija se odvija posredstvom univerzalnih modela podataka. Svaka šema podataka se modeluje na univerzalan način pomoću predefinisanih koncepata. Iako takva visokoapstrakna predstava olakšava specifikaciju preslikavanja, implementacioni detalji se ne uzimaju u obzir što dovodi do nemogućnosti potpunog generisanja izvršivog adaptera za integraciju.

Druga razlika između generičkog alata i alata *AnyMap* jeste nivo granularnosti elemenata koji se koriste u algoritmu za ponovno iskorišćenje preslikavanja. U većini analiziranih alata podržana je ponovna iskoristivost korisnički definisanih funkcija i prethodnih specifikacija tehničkih prostora. Iako najbitnija, ponovna iskoristivost specifikacija preslikavanja koja je podržana od strane alata *AnyMap*, podržana je od strane nekolicine alata. Jedino alat *Karma* podržava ponovno iskorišćenje preslikavanja na nivou pojedinačnih preslikavanja između elemenata i automatsko prilagođavanje ovih preslikavanja u novim slučajevima korišćenja kao i *AnyMap*. Svi ostali alati koji podržavaju ponovno iskorišćenje preslikavanja omogućavaju isključivo ponovno iskorišćenje celih specifikacija preslikavanja. Kako *Karma* ne poseduje mehanizam za izvršavanja adaptera za integraciju već samo mehanizam za kreiranje preslikavanja između šema podataka i definisanih ontologija, ovakav pristup nije pogodan za integraciju SF sistema, tj. za primenu u industrijskom okruženju.

Na osnovu zaključaka analize postojeće literature i alata za integraciju tehničkih prostora kao i na osnovu našeg iskustva u domenu integracije SF sistema, definisali smo novi pristup integraciji SF sistema zasnovan na principima RSVM (Poglavlje 5). Glavni elementi ovog pristupa jesu NJM za integraciju kao i algoritmi za ponovno iskorišćenje prethodno specificiranih preslikavanja. Pored samog pristupa, razvijen je i alat *AnyMap* u kojem je moguće kreirati preslikavanja korišćenjem namenskog jezika.

Osnovni cilj razvoja novog pristupa integraciji jeste da se korisnicima omogući integracija bilo koja dva tehnička prostora na uniforman način. Takav pristup bi pružio korisnicima mogućnost da nauče jedan jezik za integraciju, osposobe se za jedan alat, i na identičan način pristupe integraciji bilo koja dva tehnička prostora bez da detaljno poznaju unutrašnje mehanizme tih tehničkih prostora. Kako bi ovaj cilj bio dostignut, pristup definisan u ovoj disertaciji obuhvata korake za prevođenje originalnih šema podataka iz tehničkih prostora u generičku reprezentaciju razvijenu za potrebe ovog pristupa. U opštem slučaju, bilo koja šema podataka može biti predstavljena pomoću strukture tipa grafa. U okviru našeg pristupa, generička reprezentacija šeme podataka organizovana je u obliku stabla elemenata. U slučajevima cikličnih struktura i povratnih veza koje mogu da postoje u grafu šeme podataka, a kojih nema u strukturama tipa stabla, primenjeni su algoritmi za sravnjivanje i odsecanje veza koje narušavaju validnost strukture. Struktura tipa stabla odabrana je zbog svoje jednostavnosti, lakog prepoznavanja strukture od strane korisnika pristupa kao i zbog lake implementacije i vizuelizacije u trenutno korišćenim programskim jezicima pomoću kojih su implementirani odgovarajući alati koji omogućavaju primenu pristupa.

Naredni cilj razvoja novog pristupa jeste pružanje mogućnosti da korisnici razviju adapter za integraciju na višem nivou apstrakcije nego što je to trenutno slučaj. Ovakav pristup omogućava korisnicima da prilikom razvoja adaptera svu pažnju usmere ka rešavanju problema integracije bez potrebe da razmišljaju o specifičnostima programskog jezika u kojem razvijaju adapter i šablonima i strukturi samog kôda. U okviru našeg pristupa, ovakva specifikacija može biti kreirana uz pomoć NJM za integraciju koji dozvoljava specifikaciju preslikavanja, tj. pravila transformacije tehničkih prostora, između šema podataka. Korisnik ne mora imati prethodno iskustvo ni u jednom savremenom programskom jeziku jer kreirani namenski jezik sadrži koncepte iz domena integracije koji su korisnicima iz istog tog domena već poznati. Namenski jezik predstavljen u okviru ove disertacije poseduje i grafičku i tekstualnu konkretnu sintaksu. Specifikacije preslikavanja koje su kreirane uz pomoć ovog namenskog jezika služe kao ulaz za potpuno automatizovan proces generisanja izvršivog kôda adaptera. Pomoću jednog generatora kôda moguće je generisati adapter koji će biti izvršen na jednoj platformi. Za svaku novu platformu, potrebno je implementirati novi generator koji će na osnovu iste specifikacije preslikavanja biti u mogućnosti da generiše izvršni kôd za novu platformu.

Možda i najbitniji cilj našeg istraživanja i specifikacije pristupa je automatizacija razvoja adaptera. Iako je brzina razvoja adaptera povećanja podizanjem nivoa apstrakcije i uvođenjem jedinstvene strukture za reprezentaciju šema podataka, verovatno najveće ubrzanje može biti postignuto povišenjem stepena automatizacije procesa specifikacije preslikavanja. Korišćenjem algoritama za ponovno iskorišćenje prethodno kreiranih adaptera, tj. specifikacija preslikavanja, i algoritama poravnanja šema podataka, moguće je automatizovati proces specifikacije preslikavanja, a time posledično smanjiti smanjiti broj grešaka i vreme potrebno za ručnu specifikaciju adaptera. Ovo je posebno uočljivo u domenima primene u kojima su često specificirani adapteri između istih ili sličnih parova tehničkih prostora. U takvim slučajevima korišćenja, postoje male razlike između integrisanih šema podataka često uslovljene promenom konfiguracije SF sistema koji komuniciraju ili zamenom uređaja sličnim uređajima drugih proizvođača u okviru SF sistema. Ovakve razlike u šemama podataka u velikom broju slučajeva mogu biti automatski identifikovane i predstavljaju ulazne podatke za algoritme za ponovno iskorišćenje i poravnanje. Ovi algoritmi na izlazu daju kolekciju preslikavanja između elemenata šema podataka koji mogu biti primenjeni u konkretnom slučaju korišćenja. Na ovaj način moguće je automatski kreirati novi adapter primenom dobijenih preslikavanja. Kreirani adapter mora biti proveren od strane korisnika i, prema našem iskustvu, dodatno prilagođen kako bi bolje funkcionisao. Zbog toga u najvećem broju slučajeva, sa trenutnim algoritmima ponovnog iskorišćenja i poravnanja moguće je postići poluautomatsko kreiranje novih adaptera za integraciju. Naš pristup nudi opciju i za potpuno automatsku primenu predloženih preslikavanja ali i mogućnost intervencije od strane korisnika.

Svi koraci pristupa su podržani u razvijenom alatu *AnyMap* (Potpoglavlje 5.5). Alat je razvijen u obliku proširenja (eng. *plug-in*) okruženja *Eclipse*. Ova proširenja su grupisana prema svrsi u pet modula: osnovni modul (eng. *Core module*), modul za transformaciju šema podataka iz tehničkih prostora (eng. *Binding module*), modul za specifikaciju preslikavanja (eng. *Mapping Editor module*), modul za ponovno iskorišćenje preslikavanja (eng. *Ruse module*) i modul za generisanje kôda (eng. *Generator module*). U odnosu na kriterijume definisane prilikom analize postojećih alata, alat *AnyMap* ima sledeće osobine:

- *Način distribucije aplikacije* – desktop aplikacija, besplatna i komercijalna verzija;
- *Ažurnost softvera* – poslednje ažuriranje u 2017. godini;
- *Domen primene* – integracija podataka u domenu industrije 4.0;
- *Korišćeni pristip kreiranju preslikavanja* – neposredni pristup integraciji tehničkih prostora;
- *Korišćeni jezik za kreiranje preslikavanja* – grafički namenski jezik za kreiranje preslikavanja;
- *Korišćeni jezik za specifikaciju izvršavanja preslikavanja* – tekstualni jezik za specifikaciju izvršavanja preslikavanja zasnovan na programskom jeziku Java;
- *Korišćeni pristup generisanju kôda i njegovom izvršavanju* – generisanje adaptera koji se izvršavaju izvan alata *AnyMap*;
- *Nivo ponovne iskoristivosti koncepata specifikacije preslikavanja* – *specifikacije tehničkih prostora* – kreirana proširenja za transformaciju tehničkih prostora u generičku strukturu mogu biti nasleđena i prilagođena novim tehničkim prostorima, *funkcije* – korisničke funkcije mogu biti zapakovane i distribuirane kao Java biblioteke, *preslikavanja* – moguće je ponovno iskorišćenje i poravnanje preslikavanja na nivou pojedinačnih elemenata šema podataka;
- *Mogućnost povećanja broja podržanih tehničkih prostora* – bilo koji tehnički prostor može biti podržan kreiranjem novog proširenja za alat; i
- *Mogućnost proširenja funkcinalnosti alata* – moguće je kreirati nove Java biblioteke sa korisnički definisanim funkcijama i proširenja koja predstavljaju generatore kôda za nove platforme.

Predloženi pristup integraciji može biti iskorišćen u rešavanju brojnih problema, pošto je zasnovan na tronivovskim tehničkim prostorima koji se koriste u velikom broju problemskih domena (Poglavlje 6). Pristup može biti korišćen u domenu industrijske proizvodnje kao jedan od osnovnih gradivnih elemenata za automatizaciju fabrike. Kao predstavnik ovog slučaja korišćenja odabrana je integracija uređaja i informacionog sistema koji učestvuju u proizvodnji poluprovodničkih ploča. Pored primene u industriji, predloženi pristup moguće je primeniti i u neindustrijskim kontekstima. Neindustrijski problem koji će biti obrađen prilikom analize i ocene pristupa je razmena modela između različitih okruženja za meta-modelovanje. Konkretan primer koji je obrađen jeste integracija okruženja *Microsoft Visio* i *MetaEdit+*. Naš cilj je da kroz ova dva slučaja korišćenja prikažemo različite aspekte našeg pristupa. Prvi primer je odabran kako bismo prikazali da je moguće uvesti podršku za različite tehničke prostore kao i da algoritmi za ponovno iskorišćenje preslikavanja mogu biti primenjeni u praksi. U drugom primeru je pažnja bila usmerena ka pružanju podrške za visokoapstraktne šeme podataka koje su deo alata za meta-modelovanje. Takođe, u ovom primeru je pokazano da je moguće podržati novu platformu za izvršavanje kôda u alatu *AnyMap* bez izmena u drugim proširenjima ovog alata, osim dodavanja novog generatora.

Prvi slučaj korišćenja u kojem smo primenili naš alat, a koji je opisan u ovoj disertaciji (Potpoglavlje 6.1) odnosi se na integraciju industrijskih senzora koji šalju merne podatke i informacionog sistema koji prikazuje izmerene podatke u vidu grafikona. Senzori šalju podatke u obliku CSV dokumenata dok informacioni sistem može da primi podatke isključivo u obliku XML dokumenata koji odgovaraju predefinisanoj šemi podataka. Naš pristup je primenjen kako bi problem međuprostorne heterogenosti koji postoji u ovom slučaju korišćenja bio rešen. Specifikacija preslikavanja koja je kreirana u okviru ovog pristupa predstavlja ulaz u generator kôda a kao izlaz je dobijen adapter. Adapter je zasnovan na mikroservisnoj arhitekturi, a generisan je kôd u programskom jeziku Java. Mikroservisna arhitektura omogućava bolje skaliranje adaptera jer mikroservisi mogu biti pokrenuti proizvoljan broj puta kako bi se obezbedila veća propusnost, kao i brža transformacija podataka. Ovo je veoma bitno za primene u industrijskoj proizvodnji u kojoj se često sreću sistemi koji moraju da obrađuju i transformišu podatke u realnom vremenu.

U okviru istog slučaja korišćenja naš pristup je iskorišćen sa ciljem rešavanja i prostorne heterogenosti. U domenu industrijske proizvodnje neretko dolazi do promene konfiguracije mernih uređaja ili primene novih metoda merenja. Ovakve izmene u procesu merenja zahtevaju izmene u formatu CSV dokumenata koje senzori šalju. Kako bi bio omogućen nastavak rada sistema, inženjeri pristupaju ručnom menjanju postojećih adaptera. Broj izmena u adapterima eksponencijalno se povećava u odnosu na broj izmena u CSV dokumentima koje senzori šalju. Kako bi proces izmene adaptera bio automatizovan u što većoj meri, primenjen je algoritam za ponovno iskorišćenje prethodno specificiranih preslikavanja. Alat *AnyMap* je kao rezlutat primene algoritma korisniku na uvid dao listu kandidata preslikavanja koji bi mogli biti primenjeni u novom scenariju integracije. Svakom kandidatu je pridružena i verovatnoća da on odgovara trenutnom scenariju integracije. Nakon što je korisnik odabrao kandidate za koje smatra da su odgovarajući, na osnovu odabranih kandidata automatski su kreirana preslikavanja između šema podataka u novom scenariju integracije. Korisnik je u mogućnosti da ručno doradi kreirana preslikavanja kako bi ih što bolje prilagodio novom slučaju korišćenja. Kreirane specifikacije preslikavanja moguće je smestiti u repozitorijum specifikacija radi kasnije pretrage i ponovnog iskorišćenja. Java kôd adaptera je automatski generisan na osnovu kreirane specifikacije. Važno je napomenuti da, iako smo u ovom slučaju korišćenja primenili poluautomatski postupak kreiranja specifikacija preslikavanja, moguće je u potpunosti ga automatizovati. Umesto ručnog odabira kandidata, njihove primene i ručnog podešavanja kreiranih preslikavanja, na osnovu pridružene verovatnoće moguće je automatski odabrati kandidate sa najvećom verovatnoćom, primeniti ih i odmah preći na proces generisanja kôda. Sa rastom broja kreiranih specifikacija smeštenih u repozitorijum specifikacija preslikavanja raste i preciznost algoritma za ponovno iskorišćenja a samim je moguće i više se osloniti na verovatnoću pridruženu kandidatima.

U drugom slučaju korišćenja (Potpoglavlje 6.2) naš pristup iskorišćen je u kombinaciji sa pristupom M3B [106] kako bi razmena modela bila omogućena između okruženja za meta-modelovanje *MetaEdit+* i *Microsoft Visio*. Za razliku od prvog slučaja korišćenja u kojem su preslikavanja kreirana neposredno između tehničkih prostora CSV i XML, u ovom slučaju korišćenja preslikavanja su kreirana posredno, preko tehničkog prostora *Eclipse Modeling Framework* (EMF). Svi meta-modeli iz odgovarajućih okruženja za meta-modelovanje su pomoći M3B pristupa prvo transformisani u odgovarajuće meta-modele u okruženju EMF. Zatim su u tehničkom prostoru EMF kreirana preslikavanja između meta-modela u alatu *AnyMap*. Za potrebe ovog slučaja korišćenja, kreiran je i novi generator kôda. Kao rezultat procesa generisanja dobija se kôd u jeziku *Epsilon Transformation Language* (ETL) koji se izvršava u okviru tehničkog prostora EMF.

Uspešnom primenom pristupa u opisanim slučajevima korišćenja pokazano je da pristup može biti primenjen u različitim domenima primene. Takođe je pokazano da pristup ne zavisi od nivoa apstrakcije na kojem su specificirane šeme podataka koje su integrisane. Na osnovu uspešne primene u drugom slučaju korišćenja, pokazano je da pristup može biti korišćen u kombinaciji sa drugim pristupima kao što je M3B, drugim radnim okvirima kao i Java bibliotekama u cilju omogućavanja lakše i bolje integracije. Grafička i deklarativna priroda jezika za preslikavanja pokazala se kao odgovarajuća u kontekstu vizualizacije i preglednosti kreiranih specifikacija preslikavanja u poređenju sa tekstualnim transformacijama napisanim u programskim jezicima Java i ETL.

Na osnovu uspešne primene prsitupa i analize rešenja date u potpoglavlju 6.3, možemo tvrditi da su osnovna hipoteza (Hipoteza 0) i dve od tri izvedene hipoteze (Hipoteza 2 i Hipoteza 3) u potpunosti potvrđene. Hipoteza 1 koja glasi: *moguće je kreirati univerzalnu strukturu tipa grafa, pomoću koje će biti predstavljene šeme podataka (meta-modeli) iz tronivovskih tehničkih prostora a koja će sadržati i veze sa originalnim elementima šema podataka*, potvrđena je uz identifikovano ograničenje našeg pristupa u pogledu primene. Ograničenje se ogleda u činjenici da je naš pristup moguće primeniti isključivo na tronivovske tehničke prostore u kojima meta-modeli mogu biti odmah ili nakon transformacije pretstavljeni u obliku strukture tipa stabla bez značajnog gubitka semantike. Ovo ograničenje je posledica odabrane tehnologije u kojoj je implementiran alat *AnyMap* i nije konceptualne prirode. Ovakvo ograničenje će onemogućiti integraciju malog broja tehničkih prostora koji se retko susreću u domenu industrijske proizvodnje.

Pored analize zadovoljenja hipoteza, utvrđeno je da je pristup u potpunosti saglasan sa svim principima i ciljevima RSVM i da je jezik za specifikaciju preslikavanja zapravo namenski jezik za modelovanje. Posledično, naš pristup može imati sve prednosti i mane koje se dovode u vezu sa pojmovima RSVM i NJM. U potpoglavlju 6.3 ukazano je na to da će sve prednosti RSVM biti primenjive na naš pristup dok su glavni nedostaci pristupa nefleksibilnost modela transformacija kao i nedostatak mogućnosti proširenja generisanih adaptera ručno pisanim kodom. U okviru našeg budućeg istraživanja, posebna pažnja će biti posvećena rešavanju ovih problema.

Pored rešavanja uočenih nedostataka pristupa, dalje faze razvoja i istraživanja mogu biti raspoređene u tri kategorije: (i) istraživanja u domenu integracije sistema vođene modelima, (ii) dalji razvoj alata *AnyMap* i (iii) primena pristupa u novim domenima.

Istraživanja u domenu integracije sistema vođene modelima obuhvataju dalje unapređenje stepena automatizacije u kreiranju adaptera. Identifikacija novih načina za bolje i preciznije pronalaženje kandidata za preslikavanja je jedan od najbitnijih pravaca daljeg razvoja. Trenutno se kandidati pronalaze poređenjem elemenata šeme podataka koja se integriše i elemenata prethodno korišćenih šema podataka po imenu. Uvođenjem strukturalne i semantičke analize šema podataka, proces pronalaženja pogodnih kandidata za preslikavanja bi bio značajno unapređen. Drugi način za unapređenje stepena automatizacije jeste kreiranje hibridnog sistema za preporuke (eng. *recommendation system*) koji bi kreiranjem matrice aktuelnih i prethodno korišćenih elemenata šeme podataka omogućio pronalazak kandidata. Osim pronalaženja kandidata, ovakvi sistemi su u mogućnosti da izvrše označavanje rezultata i njihovu procenu u odnosu na faktore raznolikosti, kvaliteta i poverenja.

Stepen automatizacije procesa specifikacije preslikavanja, moguće je dodatno povećati automatizaciju unapređenjem procesa transformacije originalnih šema podataka u tehničkom prostoru u generičku predstavu šema podataka koju koristi naš pristup. U slučajevima kada

se radi u neprekidnom toku podatka kod kojeg šema podataka nije poznata unapred, bilo bi potrebno u realnom vremenu doneti zaključke o šemi podataka na osnovu samih podataka u toku. Takođe, potrebno je i u realnom vremenu vršiti prilagođavanje šeme podataka ukoliko podaci koji su dobijeni ne odgovaraju u potpunosti postojećoj šemi podataka. Takve izmene bi bilo potrebno propagirati i na same specifikacije preslikavanja. Naredno unapređenje transformacije šema podataka u generičku strukturu moguće je dobiti unapređenjem same generičke strukture. Kako je struktura tipa stabla bila uslovljena tehnologijom u kojoj je implementiran alat *AnyMap*, izmenom tehnologije ili identifikacijom odgovarajućeg načina vizuelizacije je moguće pretvoriti datu strukturu tipa stabla u opšti graf. Ovo zahteva spoj istraživanja iz domena integracije i interakcije čovek-računar.

Uspostavljanje metričkog okvira za evaluaciju pristupa integraciji takođe se može kategorizovati kao pravac daljeg istraživanja u domenu integracije sistema vođene modelima. Trenutno ne postoji nijedan ovakav okvir koji obuhvata formalnu kvalitativnu i kvantitativnu analizu nekog pristupa integraciji.

U domenu daljeg razvoja alata *AnyMap*, buduća istraživanja za cilj imaju unapređenje efikasnosti i efektivnosti u korišćenju alata kao i unapređenje alata u cilju pokrivenosti što večeg broja domena. Unapređenje podrške za nove tehničke prostore je jedan od osnovnih ciljeva. Potrebno je podržati često korišćene tehničke prostore i industrijske protokole kao što su MQTT, OPC UA i Modbus. Takođe, u generičkoj strukturi potrebno je dodati i nove metapodatke kako bi se na što bolji način opisale šeme podataka i na taj način dodatno unapredio proces kreiranja preslikavanja. Pored proširenja podrške za nove tehničke prostore, planiramo da obezbedimo podršku za nove platforme na kojima je moguće izvršiti generisane adaptere.

Pristup prikazan u ovoj disertaciji formulisan je sa idejom da bude primenjiv u raznim domenima, što je implicitno i sadržano u svim hipotezama istraživanja. U disertaciji su prezentovane dve primene u dva veoma različita domena i rešavani su problemi međuprostorne i prostorne heterogenosti. Naš cilj je da pristup primenimo i u domenima integracije elemenata sistema za nadgledanje postrojenja kao i u procesu poravnanja ontologija u domenu zdravstvene zaštite. Za razliku od industrijskih uređaja, elementi sistema za nadgledanje postrojenja su dominantno softverske komponente. Stoga, osim samih šema podataka, integracija mora da obuhvati i analizu procesa koji su podržani jednim takvim softverskim sistemom. Pored primena u domenu interneta svega, kao što smo to već i pokazali u ovoj disertaciji, pristup može biti primenjen i u drugim domenima. Realan problem koji postoji, a u kojem kreiranje preslikavanja i automatizacija tog procesa mogu biti od velikog značaja jeste problem poravnanja ontologija u domenu zdravstvene zaštite. Trenutno, postoji veliki broj ontologija u ovom domenu i veliki broj ustanova koristi različite ontologije za predstavljanje zdravstvenog kartona pacijenta. Pošto veliki broj ontologija pokriva sličan domen, moguće je jednom ručno kreirana preslikavanja automatski primeniti na novi par ontologija i tako omogućiti automatsku migraciju zdravstvenog kartona.

Ključne reči: integracija sistema, razvoj softvera vođen modelima, transformacije modela, namenski jezici, tehnički prostori

ACKNOWLEDGMENTS

The research presented in this thesis was supported by the following projects:

- "Intelligent Systems for Software Product Development and Business Support based on Models", III-44010, *National Project*, Ministry of Education and Science of Republic of Serbia.
- "Self-Adapting Interface Technology for the Integration of Machines and Information Systems", *Bilateral Project*, Ministry of Education and Science of Republic of Serbia and German Academic Exchange Service (DAAD); University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, and University of Leipzig, Institute for Applied Informatics (InfAI) e.V., Leipzig, Germany.
- "Discovering Effective Methods and Architectures for Integration of Modeling Spaces with Applications in Various Problem Domains", *Bilateral Project*, Ministry of Education and Science of Republic of Serbia and German Academic Exchange Service (DAAD); University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, and University of Leipzig, Institute for Applied Informatics (InfAI) e.V., Leipzig, Germany.
- "NIRO—Network for Intelligent and Self-Adapting Integration of Machines and Information Systems", IWINDORplus-041, URL: http://niro-network.de/, *International Project*, Ministry of Education and Science of Republic of Serbia and The Federal Ministry of Education and Research (BMBF); University of Novi Sad, Faculty of Technical Sciences, Novi Sad, Serbia, University of Leipzig, Institute for Applied Informatics (InfAI) e.V., Leipzig, Germany, Budapest University of Technology and Economics, Budapest, Hungary, University of Zagreb Faculty of Electrical Engineering and Computing, Zagreb, Croatia, University of Ljubljana, Ljubljana, Slovenia.
- "Multi-Paradigm Modelling for Cyber-Physical Systems (MPM4CPS)", COST Action IC1404, URL: http://www.cost.eu/domains_actions/ict/Actions/IC1404, *International Project*, European Cooperation in Science and Technology (COST), European Union.

I would like to thank my supervisors, prof. dr Ivan Luković and prof. dr Sonja Ristić, for their patience, advices, and knowledge they provided to me during my PhD studies. Without them, this thesis would be just a fraction of what it is today.

I would also like to thank Milan Čeliković and dr Slavica Kordić for their unreserved support and for being my main motivators and discussion partners.

A part of the research presented in this thesis was done during my collaboration with dr Heiko Kern and dr Fred Stefan from University of Leipzig. I would like to thank them for their help on implementing the tool, their invaluable ideas, and counsel which they have provided me.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

AML	Automation Markup Language	30
API	Application Programming Interface	14
ATL	ATL Transformation Language	7
AWS	Amazon Web Services	100
CIM	Computer Integrated Manufacturing	2
CPS	Cyber-Physical System	3
CSV	Comma Separated Values	7
DSL	Domain-Specific Language	8
DSML	Domain Specific Modeling Language	8
EBNF	Extended Backus-Naur Form	21
ECTL	Extraction, Cleaning, Transforming, Loading	11
EMF	Eclipse Modeling Framework	7
EPC	Event-Driven Process Chain	120
ERP	Enterprise Resource Planning	5
ESB	Enterprise Service Bus	15
ETL	Epsilon Transformation Language	7
GOPPRR	Graph-Object-Property-Port-Role-Relationship	120
GPL	General Purpose Language	7
GPML	General Purpose Modeling Languages	19
GUI	Graphical User Interface	22
HMI	Human-Machine Interface	30
HOT	High-Order Transformation	27
HTTP	The Hypertext Transfer Protocol	100
IDE	Integrated Development Environment	51
IIoT	Industrial Internet of Things	2
IoS	Internet of Services	4
IoT	Internet of Things	3
IS	Information System	30
IT	Information Technology	6
JDBC	Java Database Connectivity	63
JSON	JavaScript Object Notation	23
JVM	Java Virtual Machine	99
M2M	Model-to-Model	26
M2T	Model-to-Text	27

M3B	M3-Level-Based Bridges	105
MDA	Model-Driven Architecture	17
MDSD	Model-Driven Software Development	8
MDSE	Model-Driven Software Engineering	16
MES	Manufacturing Execution System	5
MOF	Meta Object Facility	21
MQTT	Message Queue Telemetry Transport	144
OPC	Open Platform Communications	50
OPC UA	Open Platform Communications Unified Architecture	30
OWL	Ontology Web Language	37
PIM	Platform Independent Models	34
PLC	Programmable Logic Controller	2
PL/SQL	Procedural Language/Structured Query Language	63
RAMI 4.0	Reference Architecture Model for Industry 4.0	29
RDBMS	Relational Database Management System	52
RDF	Resource Description Framework	60
REST	Representational State Transfer	100
SECS/GEM	SEMI Equipment Communications Standard/Generic Equipment Model	81
SQL	Structured Query Language	61
SUS	System Under Study	8
TS	Technical Space	7
TSSL	Technical Space Specific Language	7
UI	User Interface	34
UML	Unified Modeling Language	19
XMI	XML Metadata interchange	25
XML	Extensible Markup Language	7
XSD	XML Schema Definition	107
XSLT	Extensible Stylesheet Language Transformations	7

1 INTRODUCTION

In this thesis we propose a research that aims to provide a solution or at least alleviate integration problems that currently exist in the domain of Industry 4.0. In order to understand these problems and their repercussions, we need to present current manufacturing trends, such as Industry 4.0, and put them into a historical context. Afterward, we present main Industry 4.0 components with the emphasis on manufacture automation and how it relies on the integration of these components.

Manufacturing has been the driving factor behind the development of human race since its inception. The manufacture of things for a specific use began with the production of basic necessities and household items well before 4000 B.C. [101]. The end products were simple as well as the manufacturing process that usually utilized basic materials such as wood, stone, or metal. Over the following centuries, the manufacturing process gradually improved. Simple steps of the production process steadily began to grow in number and develop into more advanced and more complex versions of themselves.

Although the manufacturing process developed at a more or less steady pace over the course of history, several sudden and significant paradigm changes happened when the whole process was greatly influenced by new inventions. These sudden shifts or improvements of the manufacturing process are known as "industrial revolutions" (Figure 1). The trigger for the first industrial revolution was the invention of the steam engine by James Watt in 1784. A domination of manual labor was disrupted by the increasing mechanization which generated greater output of the produced goods and increased their quality by mitigating human errors and shortening time needed for products to reach its consumers. The subsequent revolutions were also caused by inventions that allowed even greater degree of automation in the man-

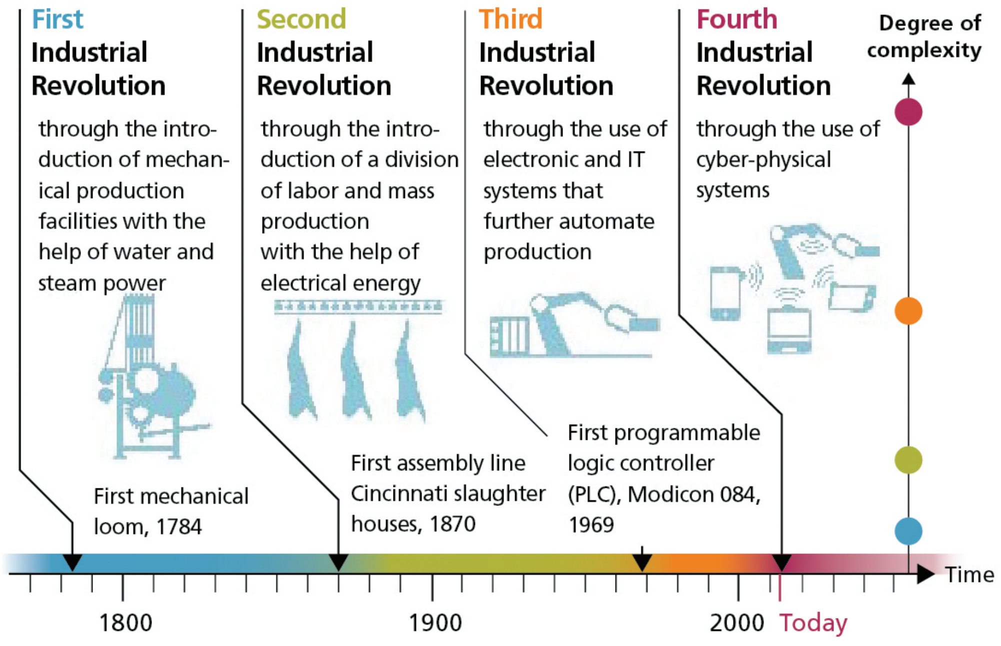

First Industrial Revolution
through the introduction of mechanical production facilities with the help of water and steam power
Second Industrial Revolution
through the introduction of a division of labor and mass production with the help of electrical energy
Third Industrial Revolution
through the use of electronic and IT systems that further automate production
Fourth Industrial Revolution
through the use of cyber-physical systems
Degree of complexity
First mechanical loom, 1784
First assembly line Cincinnati slaughter houses, 1870
First programmable logic controller (PLC), Modicon 084, 1969
Time
1800
1900
2000
Today

Figure 1: Industrial revolutions, source [137]

ufacturing process. In the 1870's, the electrical energy and the introduction of the assembly line paved the way for mass production of goods. In 1969, the first Programmable Logic Controller (PLC) was created and the digitalization began to infiltrate the manufacturing process as well as all other aspects of life. Such a widespread digitalization provided means for better and smarter machines with the aim to slowly decrease human participation in the manufacturing process. However, machines still had to be operated by humans and, as such, were not fully independent and self-adjustable to variations in the manufacturing processes.

In the resent years, a new paradigm shift is happening and it is enabled by the advances in digitalization. The shift is dubbed the 4th industrial revolution or Industry 4.0 in short. Industry 4.0 promises to improve operational effectiveness, develop entirely new "smart" services and products, as well as new business models [99]. The term Industry 4.0 (ger. *Industrie 4.0*), was coined in 2011 when Kagermann et al. promoted ideas on how to strengthen the competitiveness of German manufacturing industry [98]. The term has been later used in "High-Tech Strategy 2020" initiative of the German Government and has become eponym for all high-tech projects to be implemented by the 2020.

Outside of Germany, similar ideas and vision may be found under the names Industrial Internet and Advanced Manufacturing [61, 87]. The Industrial Internet, also called Industrial Internet of Things (IIoT), has been introduced by General Electric (GE) [75], and later put under supervision of Industrial Internet Consortium (IIC) where GE was joined by many private companies and academic institutions around the world making the IIoT a global movement. Both Industry 4.0 and IIoT encompass the same vision where machinery, people, and analytic are tightly tied together. However, unlike Industry 4.0 which focuses on manufacturing processes, the IIoT stretches beyond manufacturing and enters the sectors such as energy, transportation, healthcare, and agriculture [30]. A part of the IIoT that only focuses on the manufacturing sector was named Advanced Manufacturing in [119].

1.1 A BRIEF OVERVIEW OF INDUSTRY 4.0

The idea of smart products and smart machines in the context of manufacturing is not new. Computer Integrated Manufacturing (CIM) was a vision of 1980's, where the complex, state-of-the-art computers were introduced in factories with a goal to fully automate the production and solve cost and product quality problems that were very pronounced in the manufacturing process [82, 193]. The vision of human-less factories soon was shattered by reality in which CIM systems were extremely complex in planning as well as in construction, operation, and maintenance [203]. The technologies were not yet mature and the humans were overworked. However, the vision of fully automated and computer-centered manufacturing continued to live and evolve through the evolution of technology. The focus started to move away from big, clunky super computers that drive the production, to smart, independent computers embedded into every aspect of the manufacturing process. The basics of the omnipresent technology idea were introduced by Weiser [196] in 1991. He envisioned the world of ubiquitous computers in which computers are "*weaved into the fabric of everyday life until they are indistinguishable from it*". Weiser, and later Poslad in [160], stated that one of the main requirements for adoption of ubiquitous computers is that they are context-aware. Context-aware computers are able to provide up-to-date and relevant information about their state and environment. After context-aware computers were used in everyday life and reached satisfactory levels of functionality, stability, and security (i. e., maturity), as it often happens, they were introduced in the manufacturing process in order to improve it. Therefore, once the appropriate level of maturity was reached, ubiquitous context-aware computers slowly emerged as a main element of modern manufacturing process and currently are one of the main enablers of Industry 4.0.

Industry 4.0 is driven by the Internet, increasing number of connected devices, and future-oriented technologies for the implementation of smart machines and products. More than ever before, fast development cycles, flexibility, resource efficiency, decentralization of production, and individualization on demand are in the spotlight of the manufacturing [120, 65]. Market has changed and companies not only have to be the first to the market with their own product but they also have to provide a high degree of customization thus adapting their products to the needs of individual buyers. This often requires the production in "batch of one", where the mass produced goods are individualized and customized in order to better suit buyers' needs. This leads to the market shift from sellers to buyers, where buyers are conducting trade on their own terms. Individualization of products has become one of the main selling points for most of the companies. Additionally, as products need to be introduced to a market as soon as possible, innovation, and development periods need to be shortened.

The vision behind Industry 4.0 is creation of smart, modular, and efficient manufacturing systems in which the products control their own production. Products relay instructions and information about their current status and external conditions (i. e., production context) to the manufacturing system. Based on the received information, the system is able to adapt its behavior and perform necessary operations. For example, a product could send a message to the manufacturing system providing information about its current production phase, what are the constraints of this particular product that a manufacturing system must comply with, what is the next operation that needs to be performed, etc. This is supposed to allow mass production of products of the same type while complying with the constraints and customization requirements of a particular product.

In order to provide a definition of the term Industry 4.0, we must introduce main building blocks of the Industry 4.0 vision. According to [87, 120, 192], they are (cf. Figure 2):

- **Cyber-Physical System (CPS)** may be defined as a system where cyber and physical components are connected closely at all levels [20]. The notion of *cyber component* denotes a component used for discrete processing and communication of information, while the notion of *physical component* is used to represent a natural or man-made component that operates in continuous time in accordance with the laws of physics. According to Drath et al. [61], in addition to the physical and cyber components, each CPS requires a set of services to allow the exchange of collected data between the CPS and other actors in the process. As the concept of CPS is inherently broad by its definition and may be used to describe systems in a variety of processes, in the rest of the thesis we will use the notion of CPS to denote a CPS used in the manufacturing process. The manufacturing CPSs are sometimes called Cyber-Physical Production Systems (CPPS). In such CPSs, all physical devices in a manufacturing process are equipped with embedded computers and are connected in a common network. Embedded computers monitor the state of their physical counterparts and create and maintain their virtual representations. Factory monitoring and control systems use the virtual representation and related data streams to manage the manufacturing process.

- **Internet of Things (IoT)** is a paradigm that covers "*the pervasive presence around us of a variety of things or objects—such as Radio-Frequency IDentification (RFID) tags, sensors, actuators and mobile phones—which, through unique addressing schemas, are able to interact with each other and cooperate with their neighbors to reach common goals*" [18]. IoT can be seen as the direct enabler of CPSs as it provides a foundation for their connecting and networking. This allows CPSs to cooperate through unique addressing schemas and exchange data essential to the manufacturing process.

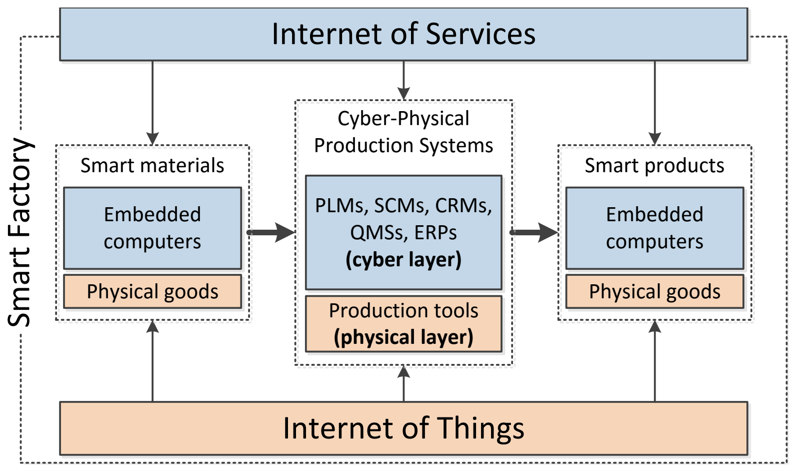


Figure 2: Fundamental concepts of Industry 4.0

- **Internet of Services (IoS)** is a vision in which companies communicate with their users and collaborators through the Internet in the form of software services. According to [36], IoS consists of the participants, an infrastructure for services, business models, and software services. Services are offered and combined into value-added groups based on their use by external actors. In the context of Industry 4.0, IoS allows communication of processed and analyzed data collected from CPSs to other interested parties, both internal and external to a manufacturing process.

- **Smart Factory** is defined as *"a factory that context-aware assists people and machines in execution of their tasks"* [127]. The *context-aware system* notion refers to a system that can consider contextual information about an object of interest. Such contextual information may be object position, size, current phase of the production, etc. A smart factory accomplishes its tasks based on the information from both physical world, e. g., machine or product position, and cyber world, e. g., electronic documents, drawings, or simulation models.

Finally, after a short overview of fundamental Industry 4.0 concepts, we present the Industry 4.0 definition given in [87]: *"Industry 4.0 is a collective term for technologies and concepts of value chain organization. Within the modular structured Smart Factories of Industry 4.0, CPSs monitor physical processes, create a virtual copy of the physical world, and make decentralized decisions. Over the IoT, CPSs communicate and cooperate with each other and humans in real time. Via the IoS, both internal and cross-organizational services are offered and utilized by participants of the value chain."*.

1.2 AUTOMATION AND INTEGRATION IN INDUSTRY 4.0

The common goal of all innovations that caused industrial revolutions was to increase the automation of the manufacturing process in order to increase the quality and speed of the production. Starting from complex CIMs in 1980's to current trends of using highly connected smart components, automation has been one of the main motivators of industrial development. However, the automation process has changed over the past few years, especially with the emergence of visions such is Industry 4.0.

The automation components are traditionally classified using an "automation pyramid". In Figure 3 we present on the left hand side Industry 3.0 pyramid and on the right Industry 4.0 automation pyramid. Bottom three levels of the Industry 3.0 automation pyramid are mostly related to the hardware infrastructure, where simple autonomous control actions

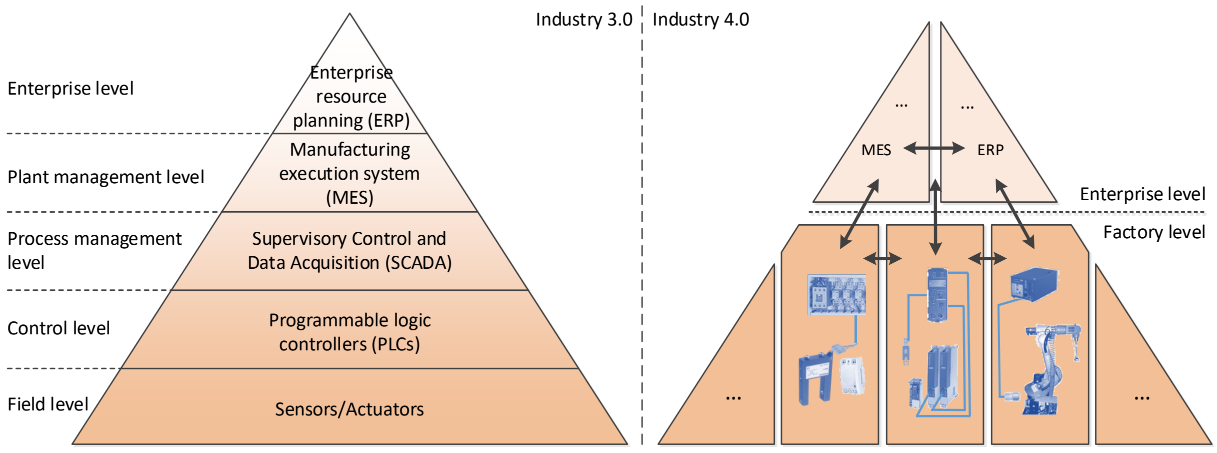


Figure 3: Automation pyramid

are performed, e. g., changing temperature or flow, together with various monitoring, performance assessment, and diagnosis functionality. At the plant management level, i. e., with the Manufacturing Execution System (MES), advanced production control algorithms are executed. Further, maintenance management, inventory control, production scheduling operations, and quality assurance may be controlled at this level. At the level of Enterprise Resource Planning (ERP) system, most of the strategic and business related planning is done. An entire supply chain of a company, among other processes comprising material procurement, manufacturing, storage, transportation, and sales, is coordinated through an ERP. Since each level requires data from the level below in order to provide services and information to end users, integration between levels is an important issue that needs to be addressed. Therefore, to ensure that a company is operational across all levels, uninterrupted information flow must be provided by the means of device and information system integration. In general, integration in the area of software and system development can be defined as: "*the process of linking separate computing systems into a whole so that these elements can work together effectively*" [125].

Often, in contemporary manufacturing systems, integration is addressed by the standardization of communication interfaces or by manual development of integration adapters [85]. Although the standardization is the best method for solving the integration issues, device or system manufacturers often adapt standards to suit their own needs or even disregard it and use the proprietary protocol due to the business or technological reasons. Therefore, factory engineers often need to create their own integration adapters that are able to communicate and transform information between the automation pyramid layers. This is a time-consuming, error-prone, costly, and a tedious task overall. Although many standards currently exist, the integration problem still remains unsolved and is one of the major problems and cost-driving factors in the industry [161].

If we take a look at the Industry 4.0 automation pyramid, at the right hand side of Figure 3, we can still identify the border between hardware and software oriented layers. At a higher abstraction level all of the layers still exist and the elements of the system can still be classified according to the affiliation to one of these layers. However, if we consider the communication aspect of these systems, the clear borders between the layers have disappeared. As everything is connected inside a smart factory, large amounts of data are exchanged in real time. Majority of materials, devices, and products are now equipped with computing devices (e. g., embedded computers and RFIDs) and networked together regardless of their computing power and purpose. The automation pyramid layer borders have disappeared as many actors in the manufacturing process have become smarter, context-aware, and can send more data to more actors than ever before. For example, a smart product may send data to a manufacturing machine to inform it how to pick it up, where to drill a hole and in which color to paint it. At the same time, the product may inform an MES in which production phase it is currently in. Further, it may inform an ERP about the geographical location of the product in order to update storage quotas and count products. As a lot of new companies enter the market with their own devices, many different protocols and data formats are used for device communication. Now, more than ever, comprehensive horizontal and vertical integration of machines and business application systems is required while implementing a smart factory. Machines at the lowest level have to be vendor-independent, flexible, and efficiently integrated with application systems from the Information Technology (IT) level and possibly with new cloud services. Therefore, a large number of adapters need to be created in order for the system to function as a whole.

1.3 MOTIVATION

With increasing automation and the degree of component coupling, factors of adaptability, quality, and efficiency of the machine integration play a central role in building and running a smart factory. Currently, the exchange of data within the automation pyramid does not meet future requirements in terms of flexibility and adaptability. As shown schematically on the right side of Figure 3, there is a vertical gap between machines at the factory level and the overlying applications and services at the enterprise level. Additionally, there exists a horizontal gap between machines from different manufacturers, customers, and domains. These machines, all located at the factory level, need to exchange data and be able to function as a whole. This is very hard to accomplish due to a number of different reasons, some of them being the usage of different protocols, hardware, and networking equipment.

Manufacturers of application systems are facing a challenge to integrate their products into the existing machine landscapes of their customers. Often, the machine and equipment landscape is heterogeneous and characterized by many different interfaces. Despite a variety of standardized industry protocols or exchange standards, machine interfaces are often adapted for a certain domain, manufacturer, or machine. Thus, integration between machines and overlying application systems causes manual adaptation effort which is complex, time-consuming, and expensive. Even in the traditional industry (Industry 3.0) around 40% of the enterprise budget was spent on the information integration tasks [26]. This number is still valid given the ever-growing number of connected devices that need to be integrated. Moreover, quality and transparency of the integration solution are hindered by manual development of the integration solutions.

Another issue that is commonly encountered in industrial use cases is the existence of many legacy machines that need to be upgraded to become "smart". Usually, when the top management of a factory decides to buy machines for the factory production floor, they expect for these machines to work for a longer period of time. Therefore, current factories could have machines that are not equipped with "smart" capabilities as they were bought before the emergence of Industry 4.0 vision. In order to make these machines smart some sort of "management shells" (cf. RAMI 4.0 in Section 3.1) around machine controllers need to be built. These management shells or smart adapters would use a single protocol to send and receive messages in order to provide a factory-wide integration. Therefore, there is still need to implement an integration adapter that will transform incoming messages received through an old protocol to messages transmitted according to the protocol used by all management shells.

All of the interfaces participating in the data exchange process send and receive data formatted according to a set of rules, i. e., data formats. In general terms, a data format and an appropriate set of tools used for its handling may be considered as a *Technical Space (TS)*. The detailed definition of the Technical Space notion is given in Section 2.2.3.

Therefore, to facilitate exchange of data between devices, adapters must be developed for each combination of technical spaces. For example, in order to import data formatted as Comma Separated Values (CSV) into an application system that can only read Extensible Markup Language (XML) documents, adapters must be developed that transform the data from the CSV TS to the XML TS. In the context of Industry 4.0, where everything is connected and a large number of elements exists in different TSs, manual development of adapters between each pair of TSs is a tedious job. This problem of TS disparity in a smart system can be named **inter-space heterogeneity** or, according to Wimmer [197], *data model heterogeneity*.

Currently, adapters are usually implemented either by using a programming language specific to the particular combination of TSs (Technical Space Specific Language (TSSL)) or by a General Purpose Language (GPL). TSSLs cannot be applied to all possible combinations of TSs and are seldom used in practice. The benefit of TSSLs is the closeness to the integration domain, i. e., they heavily rely on using the concepts of data formats being transformed. Therefore, adapter developers can easily learn and use TSSLs. Some examples of the TSSLs are Extensible Stylesheet Language Transformations (XSLT) [42], for specifying transformations in the XML TS, and ATL Transformation Language (ATL) [95] and Epsilon Transformation Language (ETL) [113], for specifying transformation in the Eclipse Modeling Framework (EMF) TS [38]. On the other hand, GPLs such as Java or C# can also be used to integrate any two TSs. These languages often come equipped with libraries that provide parsing ability for data in majority of TSs. However, as these languages are of a general purpose, it is on the developer to create mappings using the parsing libraries with the generic programming language concepts at their disposal. Because of the diversity of data formats and a lack of TSSLs, developers nowadays usually opt for GPLs. This further slows down the process of adapter creation, as inappropriate concepts are often used. In the end, none of these languages really provide a reliable and universal solution to the inter-space heterogeneity problem. Therefore, a new approach is needed.

In addition to the inter-space heterogeneity, additional problem of **intra-space heterogeneity** is often encountered. According to Wimmer [197], this problem can also be named *structural heterogeneity*. Even if the two devices are integrated with an adapter, the schema according to which a device sends data, may vary based on many factors including device configuration, device version, or the process in which it is used. Intra-space heterogeneity problem introduces even more complexity to the manual implementation of the adapters as they must be robust enough to adapt themselves to encountered changes. Alternatively, manual changes of code are needed in order to ensure the proper operation of adapter under new circumstances. This issue can be addressed with the creation of highly reusable and easily adjustable adapters. However, in the world of GPLs this is very hard to accomplish. Although the adapter code can be structured so as to allow easier extension and reuse, the constructs in GPLs are still too generic and not suitable for domain knowledge representation and its reuse.

The existence of the aforementioned heterogeneity issues greatly slows down the development of adapters and indirectly may hinder the performance of the entire manufacturing process. Existing approaches depend to much on the programing languages that are at the inappropriate level of abstraction and usually limited to predefined set of scenarios. Therefore, the aim of the research proposed in this thesis is to develop an integration language and appropriate integration approach that will alleviate both inter-space and intra-space heterogeneity problems.

1.4 DESCRIPTION OF THE RESEARCH

In this section we propose a research aimed at specifying an approach to TS integration with the main goal to mitigate heterogeneity problems that currently exist in the integration domain. Although there are many possibilities and methodologies to choose from, for the specification of the integration approach we plan to follow Model-Driven Software Development (MDSD) principles. MDSD approaches are usually centered around a language that is specific to a certain domain of application (Domain-Specific Language (DSL)). In this research we are focusing on the domain of TS integration. Several well-known benefits of MDSD and DSL-centric approaches are: (i) better expressiveness of the approach in the given domain which directly leads to a significant increase in productivity [103], (ii) the approach can be learned and used easier by users from the domain [114], and (iii) the approach would offer a possibility for analysis, verification, optimization, parallelization, and transformation in the terms of domain-specific constructs [140]. The notions of MDSD and DSLs and their main characteristics are discussed in more detail in Section 2.2. In the rest of this section we will present goals, hypotheses, and expected results of the proposed research. In this section we propose a research aimed at specifying an approach to integration of TSs with the main goal to mitigate heterogeneity problems that currently exist in the integration domain. Although there are many possibilities and methodologies to choose from, for the specification of the integration approach we plan to follow MDSD principles. MDSD approaches are usually centered around a language that is specific to a certain domain of application (DSL). In this research we are focusing on the domain of TS integration. Several well-known benefits of MDSD and DSL-centric approaches are: (i) better expressiveness of the approach in the given domain which directly leads to a significant increase in productivity [103], (ii) the approach can be learned and used easier by users from the domain [114], and (iii) the approach would offer a possibility for analysis, verification, optimization, parallelization, and transformation in the terms of domain-specific constructs [140]. The notions of MDSD and DSLs and their main characteristics are discussed in more detail in Section 2.2. In the rest of this section we will present goals, hypotheses, and expected results of the proposed research.

The main topic of the proposed research is the creation of a framework for the integration of TSs based on the main principles of the MDSD approach. The framework is centered on a language for the integration of TSs that may be categorized as a Domain Specific Modeling Language (DSML) for the integration domain. DSML can be seen as a specialization of a wider notion of Domain-Specific Language (DSL) [186] and instead of manipulating a program code like DSL, DSMLs are focused on creating and operating on models. Data originating from a source TS represent a model of the System Under Study (SUS) device that has sent it. The integration adapters are created at the level of data schemas, i. e., at the level of meta-models, and as such they can be considered as model transformations. Considering all of the aforementioned, we formulate the basic hypothesis of our research:

Hypothesis 0 *It is possible to solve heterogeneity problems in TS integration by creating appropriate DSMLs and following the principles of MDSD approach.*

The main goal of the proposed research, derived directly from this hypothesis, is to define a methodological approach and a software solution in which the MDSD principles and DSMLs will be used to overcome heterogeneity issues in order to allow integration of TSs. The derived hypotheses, which lead to the formulation of research approaches whose aim is to corroborate the Hypothesis 0 are given in the rest of this section.

In order for the proposed DSML to be useful in the real world and be able to overcome heterogeneity issues presented in Section 1.2, it must satisfy the following two requirements:

1. provide means to integrate two arbitrary TSs with the language concepts that can be easily understood by users familiar with the TSs being integrated, and
2. provide concepts that are reusable and allow for the process of reuse to be automated as much as possible.

The first requirement addresses the problem of inter-space heterogeneity. If the users of such a language understand both the data schema concepts and have a language specifically tailored for the integration domain, they would create adapters easier, faster, and with less effort. Such a language should be understandable by domain experts from any TS domain, just like it is the case with the XML and EMF experts and the integration languages specific to each of these TSs (e. g., XSLT, ATL, and ETL). The development process could be improved even further if the same language would be used for the combination of arbitrary TSs just as GPLs are used. Therefore, such a language must have the benefits of both kinds of languages in order to replace them for the TS integration.

In order to create an integration language that is used across various domains and TSs, different data schemas (i. e., meta-models) must be represented in the same way to be used by the language. There are two possible approaches to creating such a representation. First approach comprises developing one or more DSMLs for each of the TSs. Different DSMLs would enable different type of users to model the same system from different viewpoints. Using the developed DSMLs, users can specify data schemas at a higher abstraction level using concepts close to their comprehension of the domain. The benefit of such approach would be better definition of integration semantics as it is more obvious what concepts from TSs are integrated. However, such approach requires a lot of effort to implement a DSML for each TS, or to adapt existing DSMLs to allow for integration language to be used on top of them. Another drawback of such approach is that a right level of DSML abstraction is hard to achieve. If the abstraction is too high, specified transformation would not have all the necessary information in order to be executed on the data level. If the abstraction is too low, the integration language does not differ much from the data schema already present in the technical space.

The second approach is closer to the system implementation and is based on representing existing TS meta-models with a common representation that is at the same level of abstraction as the original meta-model. As each meta-model comprises entity types, relationships, and properties, it may be possible to find a common, graph-like representation to which all of meta-models from different TSs could be mapped onto. Such a generic representation of TS meta-models would allow for the same integration language to be used for any combination of TSs. Additionally, as the integration adapters must perform transformations on the original source model, such a generic representation must preserve links to the original data elements that will be used in the integration process. Therefore, the following hypothesis may be introduced:

Hypothesis 1 *It is possible to represent data schemas (i. e., meta-models) from the three-level technical spaces in a uniform way by using a graph-like representation, while preserving links to original elements.*

Once the generic representation is provided and the appropriate tools for importing TS meta-models are created, a domain-specific integration language may be developed. As it is used to specify relationships between source and target TS in a graphical way, the integration language may be classified as a relationship-based mapping language. Relationship-based mapping systems rely on the specification of high-level relationships between elements (i.e., attributes or sets of attributes) of the source and target TSs. The user starts the mapping design process by providing, usually through a graphical interface, all known attribute correspondences between elements of a source and a target TS. Once such a specification is created, it can be used as an input to other processes such is the generation of adapters and verification of correspondences [7]. Therefore, the next hypothesis of our work is:

Hypothesis 2 *It is possible to create a relationship-based mapping language that allows the creation of high-level mappings between the uniform data schema representations, from which the data integration adapters can be generated.*

The second requirement for the integration language, the reuse of language concepts, mainly addresses the problem of intra-space heterogeneity. The integration language and its concepts should be created in such a way to be easily and automatically reused in new integration projects. Reuse also helps in overcoming the inter-space heterogeneity as integration of new technical spaces could be done on the basis of constructs from previously defined adapters. Although both heterogeneity issues are tackled by implementing a reuse framework and it introduces more complexity to the development of a mapping language, it should be possible to achieve greater degree of reuse automation as the industrial context often comprises similar scenarios slightly adapted to some configuration changes. Therefore, the next hypothesis may be introduced:

Hypothesis 3 *It is possible to create an extensible reuse framework based on the created domain-specific integration language that will allow reuse of previously defined integration adapters in the presence of intra-space heterogeneity.*

After introducing these hypotheses, we may also state that the main goal of this research is to provide an MDSD approach for a structured, automated, and reusable integration of TSs. The central idea of the approach is a TS independent coupling component that, in addition to the domain-specific integration language, also allows a systematic reuse of integration knowledge from previous integration projects. The reuse or adaptation of existing integration knowledge to new projects is to be provided via framework in an automated and transparent way.

The expected results comprises the following contributions:

- *Theoretical contributions* in the field of model-driven integration of technical spaces. Such contributions will include:
  - survey on existing integration approaches and software solutions;
  - application of MDSD in the TS integration domain relying on a generic representation of meta-model structure;
  - identification of main concepts needed for the implementation of a domain-specific language for the integration of TSs;
  - conceptualization of an extendible reuse framework specifically tailored to an Industry 4.0 integration domain; and
  - specification of a methodological approach for the application of the developed integration framework.
- *Development contribution* in the form of a TS integration tool that will implement the MDSD integration approach comprising an integration language and a reuse framework.
- *Application contribution* that comprises application of the integration approach on several use cases and dissemination of analysis results and lessons learned.

The main expected result of this research is easier and simpler integration of TSs with the aim to improve the response time to production process changes and solve both inter-space and intra-space heterogeneity issues. Expected end-users are integration experts and developers from companies that provide hardware and software solutions for smart factories who need to integrate their products into an existing product landscape. Further, as the technical space notion is inherently broad, the results of our research could be used by developers who want to provide data interchange between software which data is structured in a form of a three-level TS. This will be evaluated on the practical use cases that are presented in Chapter 6.

1.5 THESIS STRUCTURE

Apart from Introduction and Conclusion, the thesis is organized in six chapters.

In Chapter 2 we introduce the main terminology and the background behind the research presented in this thesis. The main notions of integration and Model-Driven Software Development are introduced and explained in detail. In this chapter we also relate main elements of this research to the introduced notions.

In Chapter 3 we present the related work in the fields of schema-based integration and matching, model-driven integration, and ontology alignment which are closely related to the topic of this thesis.

Existing data integration and Extraction, Cleaning, Transforming, Loading (ECTL) tools are presented in Chapter 4 as they provide us with the insight into current state of the art of the contemporary integration tools. We present the process of choosing tools for the survey, categorizing them, and in the end a set of evaluated characteristics. Each of the tools is evaluated in detail and the results are presented. The overall findings of the survey are given and a generic profile of an integration tool is presented which serves as the role model for the tool we implemented for our approach. Identified best practices and our personal experience in using the tools are also disseminated in this chapter.

In Chapter 5 we introduce our approach to integration adapter specification. First, we identify the place for our approach in the general integration process. Next, we introduce the integration process supported by the approach. In the same chapter we present the supporting tool, named AnyMap, and its main modules. Special attention is given to the mapping and expression languages.

Our approach is applied in several use cases of which the two most representative ones are presented in Chapter 6. The first example encompasses application of our approach in an industrial context in which sensors are integrated with information systems. In this example we will provide a complete description of the integration process with the some details regarding the AnyMap tool usage. As the reuse of mappings is often a necessity in industrial integration scenarios, we will showcase the reuse on an example where the intra-space heterogeneity is introduced by changing a sensor configuration. Second use case concerns model interchange between MetaEdit+ and Visio meta-modeling environments. We will show that our approach can be used in domains where high-level mappings are required and not the low-level serialization-related mappings like it in the first use case. In this chapter, we also analyze our approach and discuss the results in the light of specified hypotheses. Each of the hypotheses is discussed and its confirmation or rejection is presented in detail.

2 THEORETICAL FOUNDATIONS

In this chapter we present theoretical foundations of our research on model-driven TS integration based on a mapping approach. The approach can be roughly categorized as an model-driven approach to system integration. Therefore, in the following sections we introduce the main concepts and notions from the respective domains of system integration and model-driven development. This will establish the theoretical foundation of our work as well as allow easier understanding of the following chapters.

2.1 SYSTEM INTEGRATION

Both in business and industrial domains, integration is one of the main enablers of purposeful and continuous operation. Stand-alone software systems or isolated machines cannot fulfill the requirements of the modern business and industrial processes and the interaction between them is an essential element of the entire system operations. The growing market needs and the advancement of technology require for the new technologies to be constantly applied and the processes to be adapted and changed. This leads to heterogeneous systems in which both new and old software systems and machines need to coexists and communicate. This is most obvious in the industrial environment. Factories are often equipped with expensive machines and companies that bought them want to use them as long as possible in order to justify the investment. It is not rare that these machines are operational for several decades. Introduction of new software and machines in the production process is often required in order to enrich or adapt the process to the current market needs. Therefore, new machines and software need to be integrated both with existing systems and with each other in order to fulfill the information availability and the quality of the production process. To make things even more difficult, in companies there is often a significant investment already in place for a variety of system integration technologies [97]. Therefore, system integrators are not only presented with an integration problem but are also limited in the tools they are able to use. All of this makes the integration a difficult but an essential element of the system life-cycle which is often considered as one of most important strategic priorities.

The reality today is that the integration process has both technological and organizational impacts on a company. This may lead to different companies having totally different views on integration tailored to fit their preferences and use cases. Consequently, this means that there is no single definition of the *system integration* notion that will satisfy all possible viewpoints appropriate for all use cases and companies. We found several definitions of the notion but we chose to present the following two that represent the notion of system integration from different perspectives. From a technological perspective, **system integration** is "*the melding of divergent and often incompatible technologies, applications, data, and communications into a uniform information technology architecture and functional working structure*" [149]. In addition to just technology, the integration of systems involves a complete set of business processes, managerial practices, organizational interactions, structural alignments, and knowledge management. Therefore, from a non-technological standpoint, **system integration** "*represents a progressive and iterative cycle of melding technologies, human performance, knowledge, and operational processes together*". In this thesis, we rely on the first definition as we are more focused on the technological aspect of the integration. Moreover, as system integration encompasses both hardware and software integration, we must note that in this thesis we deal only with the latter one. Even in the case of machines, e. g., sensors and actuators, we look at them as CPSs and only consider their cyber part. Therefore, in the rest of the thesis, unless otherwise stated, we will use the notions of integration and system integration to denote software-level integration.

The effects that system integration may have on organizations are multifold. Integrated systems improve the competitive advantage with a unified and efficient access to the information [97]. It is much easier to get relevant, coordinated information from a variety of sources. In effect, the total becomes more than the sum of its parts. Furthermore, the integrated system facilitates better collaboration of workers regardless of their geographical location, time zone, and location of information. Integration allows information and knowledge to be simultaneously shared by workers, business partners, and even collaborative competitors [149]. The need to integrate is also driven by new forms of business and partnerships. Groups of companies and workers not only share data and information, but also have exposure to their respective business partners' operations [149]. In a factory which system is integrated both internally and externally with the partner systems, partners may have an insight into the production status of a product of interest. Customers may also be able to see the state of their product while it is being manufactured, packaged, shipped, and delivered. From this, we may see why the integration is considered as one of the main driving factors of the latest industrial revolution.

According to [97, 149], there are four kinds of system integration:

- *Interconnectivity*. Interconnectivity involves making different parts of the system work together. This includes the facilitation of simple data exchange and establishing communication between the connected parts of the system. The existing functionality of a system remains the same as this kind of integration only provides data-level integration but not integration at the functional level. This is the most basic kind of integration and all other kinds are built on top of it.

- *Interoperability*. Interoperability refers to the functional integration of different software systems or machines that allows exploitation of capabilities of all integrated system parts. This is the kind of integration most commonly found in companies. This kind of integration comprises application-level integration. The application-level integration focuses on sharing functionality and business logic instead of pure data sharing like it is the case in data-level integration. It is usually achieved through the use of Application Programming Interfaces (APIs).

- *Semantic integration*. Semantic consistency emphasizes rationalization of data elements and their meaning. In order to achieve semantic consistency, data from different parts of the integrated system must be handled and understood in a uniform manner across the system. One way of providing accessibility to data and minimizing the potential of errors in human interpretations is through the creation of standard data definitions and formats. In this kind of integration, the presentation-level integration is performed. The presentation-level integration results in an integrated system that provides a unified presentation layer, through which the users can access the functionality of the integrated system. A semantic uniformity is necessary in order to present data from disparate parts of the system in a same way.

- *Convergent integration*. Convergent integration involves the integration of technology with business processes, knowledge, and human performance. This requires the presence of all three previously introduced integration kinds, but involves factors other than technological ones. It is the highest and most sophisticated kind of integration. The business process integration is present at this level. The business process integration enables non-compromise support for business processes in the enterprise where existing solutions take part in distinctive steps of the process.

Our approach, presented in Chapter 4, aims to provide integration at the data level as this is the most important issue in the IIoT domain in which the definition of interconnectivity and interoperability are slightly adapted. Basically, by connecting the devices at the hardware level we are providing interconnectivity within the system. In order to provide interoperability, we must provide an adapter which translates messages sent between connected devices. This way, connected devices are able to operate together and thus provide a meaningful output for the system as a whole. Therefore, our goal is to provide the interoperability by connecting different inputs and outputs of devices being integrated and thus allowing them to provide a common functionality to the end user. Although we have created a common data structure that may facilitate semantic integration, which we have shown in the paper [55], it's not the primary goal of our solution. As data-level integration is the prerequisite for all other kinds of integration, our goal is to enable data-level integration in the IIoT domain by developing an appropriate approach.

In order to apply a certain integration kind, different methods, techniques, patterns, and technologies can be used. These have been developed over the years, ranging from point-to-point integration over enterprise application integration and business process management to service oriented architectures [97]. All of the methods can be roughly classified in the following four distinct categories [80, 97, 149]:

- *Vertical integration*. Vertical integration methods allow integration of system parts based on their functionality and thus creating same-function entities of integrated system parts. As the integration is preformed rather quickly and involves only necessary system parts, these are the cheapest integration methods in the short run. However, since same-function entities are not reusable and adding a new functionality requires creation and integration of a new integration entity, these methods can lead to great expenses over time.

- *Star integration*. Star integration methods encompass connecting each system part of the system with each of the remaining parts. The cost of using this method can vary based on the number of interfaces that a system part is exporting. This kind of integration provides the greatest degree of flexibility and the reuse of functionality. However, time and costs needed to integrate a new part of the system raises exponentially as the number of integrated parts increases.

- *Horizontal integration*. Horizontal integration methods use a specialized component for the exchange of messages between the disparate parts of the system. These methods, often comprising the usage of an Enterprise Service Bus (ESB), allow for each system part to be connected to the communication component only once. A communication component, or a bus as it is called, can translate sent messages and deliver them to the appropriate receiver in the system. These are the most flexible methods of integration. With systems integrated using a horizontal integration method, it is possible to completely replace one system part with another one that provides similar functionality but exports different interfaces. Such a substitution is completely transparent for the rest of the integrated system parts. However, the cost of using these methods can be significant as the cost of specifying data transformations and appropriate business logic that drives the integration cannot be avoided.

- *Integration based on a common data format*. These integration methods are based on an application-independent (or common) data format to which all other data formats are mapped. These methods usually comprise two steps. In the first step, an adapter is used or created to transform original messages to messages that conform to the common data format. Afterward, in the second step, semantic transformations are executed between two common representations of data from different system parts in order to achieve the desired level of integration between them.

Our approach can be classified as an integration approach that uses a common data format to achieve integration between different system parts. We have created a common data structure onto which all other data formats are mapped based on main principles of the Model-Driven Software Development (MDSD). Each technical space, i. e., system part, that is being integrated, is represented in a form of a generic data model. In compliance to the MDSD principles, each model is considered as a first class entity and all operations are executed on these models. Models are directly used in the process of development of integration adapters at a higher level of abstraction than it is the case when writing adapters manually at the data level. Once the models are created, transformations are written using a graphical Domain-Specific Language (DSL).

All of the aforementioned notions related to the MDSD domain are introduced in the next section.

2.2 MODEL-DRIVEN SOFTWARE ENGINEERING

Development of software, regardless of the domain, is a complex task. Increasing number of domains in which the software has been applied increases this complexity as users' requirements and needs become more and more demanding and complicated. According to [34], developers need to fight two kinds of complexity while developing software: (i) *essential complexity*, which is inherent to the problem being solved and which cannot be mitigated, and (ii) *accidental complexity*, which is not related to the problem but encompasses components, tools, and ideas that are not important or do not need to be used in order to solve the problem. Although the first kind of complexity cannot be mitigated and therefore is a constant factor that directly influences software complexity, the second kind is usually caused by developers and the inappropriate tools or methodologies that are chosen. By using general purpose programming languages and thinking in terms of programming constructs instead of constructs specific to the problem domain, developers tend to over-think or to over-complicate solutions. By providing a language with domain-specific concepts to developers and domain experts, it is possible to reduce the accidental complexity and therefore increase the quality of software solutions. The **Model-Driven Software Engineering (MDSE)** methodology and particularly **Domain-Specific Languages (DSLs)** are considered to be a viable way of reducing the accidental complexity by introducing the domain specific elements to the software development process [188].

According to [32], MDSE is defined as a *"methodology for applying the advantages of modeling to software engineering activities."* This methodology is based on explicit specification of models which are considered as first class artifacts of all software engineering activities. Thus, any software-related artifact is considered to be a model or a component of a larger model.

In this thesis we are concerned with the development activity of the software engineering process. In the context of MDSE, this activity is centered around developing software systems in form of models and as such it is called **Model-Driven Software Development (MDSD)**. Therefore, we can consider MDSD just as a specialization of a broader notion of MDSE that covers more software engineering activities such as software re-engineering, software migration, and software evolution. MDSD comes in many forms and flavors [178], and some of the well-known MDSD approaches are: (i) *Architecture-Centric MDSD* [178]—that includes modeling families of software systems by using individual, schematic, and generic implementation components, (ii) *Domain-Specific Modeling* [103]—that includes developing software using a DSML and code generation, (iii) *Generative Programming* [46]—that includes modeling families of software systems and allowing for a highly customized and optimized instance of that family to be automatically manufactured by means of configuration knowledge, (iv) *Model-Driven Architecture (MDA)* [112]—that uses three different abstraction levels (computation independent, platform independent, and platform specific) at which a software is modeled, and (v) *Model-Integrated Computing* [102]—that uses models as central artifacts in the entire development life cycle of real-time and embedded systems.

Main goals of MDSD include the increase of the development speed through automation and single point of system definition; increase in software quality through formalization; increase in component reuse and improved manageability of complexity through abstraction; greater domain expert inclusion in the development process; and better communication between different stakeholders in the software development process. Several of these goals, mainly increase in speed, quality, reuse, and complexity management, have been the main driving factor in the past software development shifts that have happened. Increase in abstraction was the main factor that allowed moving from machine language to the higher-level and symbolic languages that we use today. MDSD is therefore an obvious evolutionary step forward as it aims to increase abstraction level further by introducing domain-specific models.

2.2.1 *Models and Meta-Modeling*

Modeling is not new and has been used in many scientific and non-scientific contexts throughout the history. Examples of models range from Bohr's model of the atom in atomic physics, through molecular models in chemistry, to models representing underground railway paths in cities. In contrast to merely using models as a visualization technique, MDSD offers a significantly more effective approach that specifies models as both abstract and formal artifacts that are focal points of the entire development process. In this case, models are no longer used only for the documentation and visualization purpose but as integral parts of software, allowing an increase in both quality and speed of software development. MDSD models take the role of program code because the major part of final implementation can be generated directly from them. This is the generative approach in the MDSD domain and we rely on it in our research. It is worth mentioning that in some cases models can be considered as final implementations and as such they can be executed. This is known as a model execution approach in MDSD.

In [111], Kühne defines the notion of a **model** as *"an abstraction of a (real or language-based) system allowing predictions or inferences to be made."* In addition to this broader definition of the model, Stahl et. al. [178] define the model as *"an abstract representation of a system's structure, function or behavior."* In both definitions, abstractness does not stand for ambiguity, but for conciseness and a reduction to the essence. These model definitions can be applied to all models that are used in MDSE. Such models or a group of models represent a single point of knowledge on top of which all operations (transformations) are executed that allow for entire system to be observed and for conclusions about the system to be made. This is a consequence of the main MDSE principle: "*Everything is a model*" which is analog to the well-known principle of the object-oriented methodology "*Everything is an object*" [37].

According to [177], a model needs to have three features: (i) *mapping feature*—a model needs to be based on an original (i. e., system), (ii) *reduction feature*—a model only reflects a relevant selection of an original's properties, and (iii) *pragmatic feature*—a model needs to be usable in place of an original with respect to some purpose. Therefore, we can say that models are mentally created by the means of mental mapping and reduction in which SUS entities are identified, grouped together, generalized, and stripped of properties that are irrelevant for a particular use case. This model creation process is called **modeling**.

Once a mental model is created, **modeling languages** are needed to provide the appropriate notation for representing these models. Often, developers of a modeling language create the notation in such a way to provide domain experts, i. e., modelers, visually and semantically familiar concepts from the domain being modeled. Development of a modeling language requires identification of domain concepts that will be mapped onto appropriate language concepts and to which a graphical or textual notation is provided (cf. Section 2.2.2).

Modeling languages are developed and used according to the MDSD four-level conjecture [17, 39] which is presented in Figure 4. The levels are defined according to the degree of abstraction they imply, starting from the lowest level. At the bottom level (M0), the SUS exists in which the observed entities reside. A model of the system is created by the means of an appropriate modeling language and it resides at the next level (M1). Model contains a virtual representation of the observed system entities with only relevant information about each particular entity.

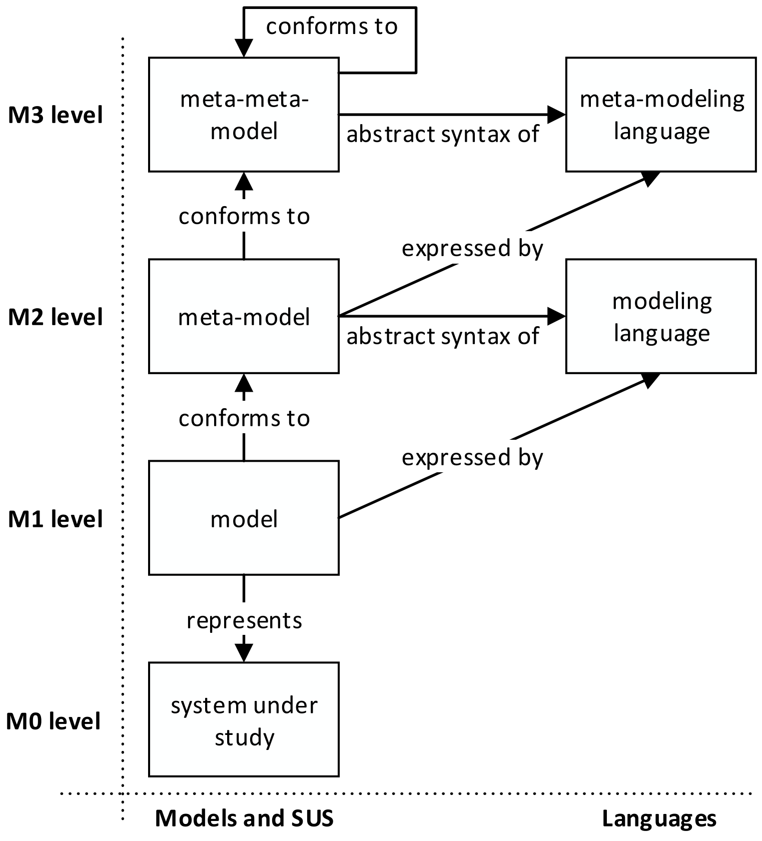


Figure 4: Model hierarchy and modeling languages

As the statement *"Everything is a model"* should be always valid, models themselves can be defined as instances of more abstract models we call **meta-models**. The creation of a modeling language requires that the real system is observed, but instead of focusing on each system entity in particular, classes (types) of entities are identified together with the necessary properties and relationships between them. Specification of entity types, properties, and relationships is called a meta-model which resides at the next level (M2) of the MDSD conjecture. To create a model, model elements are instantiated according to a type, and properties are populated with values. Therefore, it can be said that a model **conforms to** a meta-model. The meta-model represents the abstract syntax (cf. Section 2.2.2) of the modeling language as it defines all the concepts that the language needs to have in order to allow the model specification.

Meta-models also need to be specified by using a language often referred to as a **meta-modeling language**. The concepts of such a language do not depend on a particular domain and are defined by the environment in which the meta-models are specified. Meta-modeling language concepts are given in a form of a **meta-meta-model**, which resides at the top level (M3) of the MDSD conjecture. Therefore, each meta-model must conform to a particular meta-meta-model. As there is no practical benefit in introducing new levels of abstraction above M3, meta-meta-models usually conform to themselves and are specified reflexively using their own concepts.

2.2.2 *Domain-Specific Languages*

Modeling languages that rely on the domain knowledge and provide concepts close to the target domain are called Domain Specific Modeling Languages (DSMLs) and can be seen as a specialization of a wider notion of Domain-Specific Languages (DSLs) [186]. The advantage of DSMLs in comparison to General Purpose Modeling Languages (GPML), such is the Unified Modeling Language (UML) [172], is the closeness to the domain under observation and appropriateness of modeling concepts that are used for the given modeling task. By using such a language, a domain expert or a user familiar with the domain is able to specify the solution faster, with less errors, using familiar concepts than it is the case with GPMLs. In the rest of the section we are focusing on the definition and main properties of DSLs. All the conclusions and discussions can be applied to DSMLs as well.

According to [186], a DSL is "*a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain*." The key property of DSLs is that they are focused on a particular domain. This means that instead of generic and general concepts that are present in GPLs, a DSL should offer concepts specifically tailored for problem solving in a given domain and concepts that are easily understandable to domain experts. However, the main issue with the DSL definition is the inherent vagueness of the *problem domain* notion. Different problem domains vary in size, some problem domains overlap and it is not so clear where one domain starts and where the other ends. Therefore, the development of a DSL can be a very tedious and time-consuming task. However, the benefits of using the DSL can be greater than the drawbacks and problems encountered in its development.

According to [103, 186, 189], the benefits of using a DSL include but are not limited to:

- *Domain Expert Involvement*. DSLs provide means to express solutions in the idiom and at the appropriate abstraction level of a problem domain. Domain experts can easily understand and often validate, modify and even write programs using the DSL. Since a DSL captures concepts of a certain problem domain in a way that is not cluttered with implementation details, DSL programs are more semantically rich than GPL programs. This leads to easier analysis construction and more meaningful error messages. Uncluttered code can lead to easier manual validation of programs as a user is not distracted with the GPL constructs that are too generic and not important for a particular domain. Even when domain experts are not willing to write programs, they can take an active role in the development when paired with a programmer. They can think and communicate with a developer in the terms from the domain which are later directly translated to appropriate DSL elements.

- *Productivity and Quality*. Productivity can benefit from conciseness, self-documenting, and reuse which can be present to a large extent in DSL programs. The sheer amount of code a developer needs to write in a GPL can lead to accidental complexity and program specifications that are hard to read and understand. By taking advantage of conciseness and reduction of written code in DSLs directly influences productivity of a developer. In a similar way, a quality of the product is increased by removing unnecessary code, limiting the freedom of programmers to use only the best practices in a certain domain and therefore prevent accidental errors from occurring.

- *Productive Tooling*. As it is the case with DSLs that provide a better user experience for their problem domain than the GPLs, the appropriate tooling support can also be aware of the DSLs and the language constructs. The user experience and their productivity can be greatly increased by allowing DSL-aware static analysis, code completion, debuggers, and simulators. This can also lead to a faster introduction of new team members to an existing project as they need less time to learn a language and the tooling support as it is based on the domain concepts.

- *Platform Independence (Isolation)*. Sometimes, a DSL is created in order to abstract from a single technology platform. By using a DSL and a set of generators, it is possible to write a single DSL application and execute it on multiple platforms. Adding a new execution platform usually requires just adding a new code generator for that platform. This increases the maintainability and portability of code as well as the concerns expressed in the DSL are separated from the implementation details and specifics of the target execution platform.

- *Reduction of Execution Overhead*. The code, generated from DSL program specification, can be generated in a way to satisfy some strict requirements of a target platform. For example, code can be generated in such a way to guarantee the best possible performance or optimized resource usage. As the code is always generated in a same way, all generated executable code artifacts will satisfy the same requirements. Besides, accidental introduction of non-optimized code and errors is prevented.

The choice of whether to develop a DSL does not depend on a single benefit from the previous list. It depends on a specific use case or on satisfying multiple benefits at the same time. However, a developer must have in mind some drawbacks of developing a DSL before going through with it [103, 186, 189]:

- *Effort and Cost of Building a DSL*. The development of a DSL is a time-consuming and a costly task. Often, it is difficult to find the proper scope for a DSL or to identify a right level of abstraction at which the concepts of the language will be used. It is also hard to balance between domain-specificity and general-purpose programming language constructs because of the vague domain borders. Often developers tend to look at the domain too widely and include more concepts than needed. Therefore, in order to reduce cost and effort of a DSL development, a developer needs to be: (i) familiar with the domain, or at least be able to communicate with a domain expert and identify appropriate abstractions and domain concepts, (ii) proficient with a language workbench for developing DSLs and (iii) an experienced language developer. Often, deciding whether to develop a DSL or not depends on the effort and cost of building it. In computer-related technical domains the benefits and return of investment can be seen shortly after generating several applications while reusing most of the DSL program code. In other domains, such as physics, chemistry, and social security, it can be harder to justify the cost of building a DSL as the benefits are not quantitative like the generated lines of code.

- *Cost of Evolution and Maintenance of a DSL*. A language that is not actively maintained and evolved will become obsolete and will not be able to cope with the advancements in the domain in which it is used. This is especially true for DSLs, as a little change in the domain can have a major impact to the language itself. Unlike DSLs, GPLs fight these issues with their generics. Therefore, in addition to the cost of building a DSL, the maintenance and evolution cost must be considered, too.

- *Language Engineering Skills*. Building DSLs requires experience and skill. Modern language workbenches have made the development easier than it was in the past. But there is still a often steep learning curve. Additionally, the specification of a good and usable language is not made simpler by better tools. It is still a form of art to identify the right domain abstractions, to create a usable and efficient language, and to create the most appropriate concrete syntax for the DSL. Once a developer gets experienced in creating DSLs another pitfall awaits: creating too much DSLs. Instead of searching and learning an existing DSL, a developer may choose to develop a new language that often remains unfinished product as it only needs to satisfy developers needs. This may result in a large set of unfinished and often incompatible DSLs each covering similar, related, or even overlapping domains.

- *Tool and Process Lock-in*. Although many of the DSL tools and language workbenches are open source, the vendor lock-in is not a big issue like the tool lock-in. The reason behind the tool lock-in is the lack of interoperability between the tools [104] (cf. Section 6.2). Once a DSL is developed in a language workbench, it is often the most convenient to continue using the same workbench although it may not be maintained anymore or it lacks tools to support all of developers' requirements. In addition to the tool lock-in, often a so called process or investment prison can be encountered. Users of a DSL can find its benefits and productivity increase so appealing that they get used to the language so much that they get themselves locked in it. Radical change of their work process may seem unattractive to them once they've become very efficient in using the DSL.

Once a developer decides to develop a DSL, he or she must define its abstract and concrete syntaxes.

The **abstract syntax** of a DSL is *"a data structure that represents the semantically relevant data expressed by a program"* [189]. It does not contain any notation details and it is essentially a tree data structure. The actual program, which is an instance of such a structure, is often called an abstract syntax tree. There are two approaches to creating an abstract syntax [32, 140]: (i) grammar-based approach and (ii) model-based approach. One of the most common ways to define an abstract syntax of a language is by using grammar rules for the language. Most notable textual notation for defining a language grammar is Extended Backus-Naur Form (EBNF). However, for modeling languages a model-based approach is often used. In this approach, a meta-meta-model such as Meta Object Facility (MOF) is used instead of EBNF in order to specify an abstract syntax of a modeling language. The abstract syntax is specified in a form of a meta-model. In order for a user to create a model that conforms to such a meta-model, a concrete syntax needs to be provided.

The **concrete syntax** of a DSL is *"what the user interacts with in order to create programs"* [189]. A DSL has exactly one abstract syntax but may have multiple concrete syntaxes defined. This is due to the facts that different users prefer different syntaxes for different use cases or that some problems are easier solved using one concrete syntax instead of another. There are several possible concrete syntaxes to choose from according to [189]:

- *Textual*. Textual DSLs are a major subgroup of DSLs and they use textual notations based on ASCII and Unicode characters to form program commands or model concepts. In [48], the authors present the main benefits of the textual concrete syntax: (i) *existing tools can be used as a fallback option* as a plain text editor can be used to edit textual files that contain DSL expressions, (ii) *existing text-based version control systems can be used* and therefore all of the benefits these systems provide can be easily utilized, and (iii) *programmers are used to textual syntaxes* as a majority of contemporary programming languages are textual. The same authors provide the list of main drawbacks of textual DSLs: (i) *notation verbosity* can cause a model to become unreadable and overcrowded as there is just a limited support for hiding model elements, (ii) *the structure of the model is harder to comprehend* as relationships are often not so visible when surrounded by a lot of text, and (iii) *navigation is not as intuitive as is the case with graphical DSLs* as the navigation is mainly done by scrolling sequentially through the text, searching for text patterns, listing, or jumping to all usages of certain model element.

- *Symbolic*. Symbolic DSLs are basically textual DSLs enriched with a special set of characters for building formulas and easier and more concise creation of program expressions. Often, these additional concepts comprise superscript, subscript, or fraction bars. In the rest of this thesis, we will often refer to these kinds of languages as *formula-based* languages as their commands resemble mathematical formulas. Symbolic representations are often used when the domain heavily relies on symbols like it is the case in mathematical and scientific domains.

- *Graphical*. Graphical DSLs are another major subgroup of DSLs and they use graphical shapes in order to represent program or model elements and relationships between them. Graphical syntax is very good at emphasizing relationships between data. Some of the main benefits of the graphical syntax are [48]: (i) *model structure is easier to comprehend*, which is true for languages that represent relationships and are traditionally seen as diagrammatic, but in general it depends on the user experience and personal preferences, (ii) *easier model navigation* achieved by using operations as zooming and panning of contemporary visual editors that allow for every part of the model to be easily accessed, and (iii) *visual languages are easier to learn* than the textual language as it is more intuitive to take an element from the palette and through a trial and error create diagrams than start writing to an empty text file. On the other hand side, according to [48], some of the main drawbacks of the graphical syntax are: (i) *it is hard to develop and maintain* such a DSL and the effort is proportional to the amount of work needed to develop and maintain a fully-fledged graphical editor especially during the evolution of the DSL, and (ii) *serialization format is different from the presentation format*, which can be problematic if we need to handle the models through a version control system.

- *Tabular/Matrical*. Tabular and Matrical DSLs use tables and matrices in order to represent model constructs and program commands. Tables and matrices are best used when two independent dimensions of the data need to be related and presented to the end-user. This is due to the fact that tables and matrices emphasize readability over the ability to write data [189].

- *Form-based*. Sometimes, a form-based Graphical User Interface (GUI) is built to facilitate input of parameters that influence the program execution or generation process. For some users, this is a preferred way of using a system instead of using one of the previously specified syntaxes. Most commonly, data entered through a form is serialized in some of the common serialization formats like XML or JavaScript Object Notation (JSON) or directly to a textual file consisting of a textual syntax constructs of the same language. Therefore, these forms are not so often considered as a concrete syntax of a DSL with an argument that "*considering GUI forms as a kind of a concrete syntax would make any GUI application a DSL*" [189]. Our opinion is that a form-based GUI can be considered as a DSL in a small number of use cases where a DSL is in fact a language for domain-specific configuration of a process or a set of processes in a domain. Arguably, in these cases, a user might benefit from a form through which it can enter such a configuration. That is the reason why in the rest of this thesis we refer to such languages as *configuration-based* languages.

It should be noted that it is possible to provide multiple concrete syntaxes for a single DSL. Different aspects of the domain can be described using different concrete syntaxes. Furthermore, a same diagram or domain-specific program can be specified using different concrete syntaxes at the same time.

For our approach, presented in Chapter 5, we have chosen a model-based approach to create abstract syntax of our DSML. We have used the Ecore meta-meta-model [179] and the choice was influenced by our decision to use Eclipse as a development environment and a GUI container that is enriched with our tool-specific plug-ins.

In the survey of integration tools which we present in Chapter 4, we have identified a type of the concrete syntax used in each of these tools. In that chapter, we also discuss the appropriateness of each syntax in the domain of system integration and mapping or transformation specification. We are aware that such a discussion is subjective and tends to be biased by previous experience and prejudice. However, we will try to be as objective as possible in drawing conclusions. Based on these findings, for the concrete syntax of the DSL we have developed, we chose the graphical syntax. In our opinion, this syntax is the most suitable syntax for the creation of mappings in the IIoT integration domain.

2.2.3 *Modeling Spaces and Technical Spaces*

The four-level conjectures considered in Section 2.2 can be considered as modeling spaces. The notion of the **modeling space** describes "*an application domain in which formal modeling methods are applied in order to derive the solution for an identified problem*" [32]. As we are considering only the four-level abstraction architectures, the more appropriate definition of the modeling space notion is given by Đurić et al. [184] who define a modeling space as "*a modeling architecture defined by a particular meta-meta-model*". Each meta-model defines a viewpoint and its models a view on the real world. A meta-meta-model defines core concepts for defining meta-models and is defined recursively, by itself. If the meta-meta-model was defined by using concepts from another, more abstract model, it would be considered as a meta-model in another modeling space. This can be seen in Figure 5 where two modeling spaces are presented. On the left side, the XML Schema modeling space is presented, while on the right side we present the widely used MOF modeling space. Also, from the same figure, it can be seen that modeling spaces can be defined at different abstraction levels.

In [184], the authors classify modeling spaces as: (i) conceptual modeling spaces, focused on conceptual (abstract or semantic) things that are able to represent semantics, but are not focused on techniques for representation or sharing their abstractions, and (ii) concrete modeling spaces, equipped with notation, but lack the means to represent the semantic. This classification is based on the model definition in which a system can be viewed both from a real-world and a language-based perspective. This dual viewpoint is a consequence of a changeable perspective or a viewport through which we can look at the system at different abstraction levels. Different viewports on the same XML use case are presented in Figure 5.

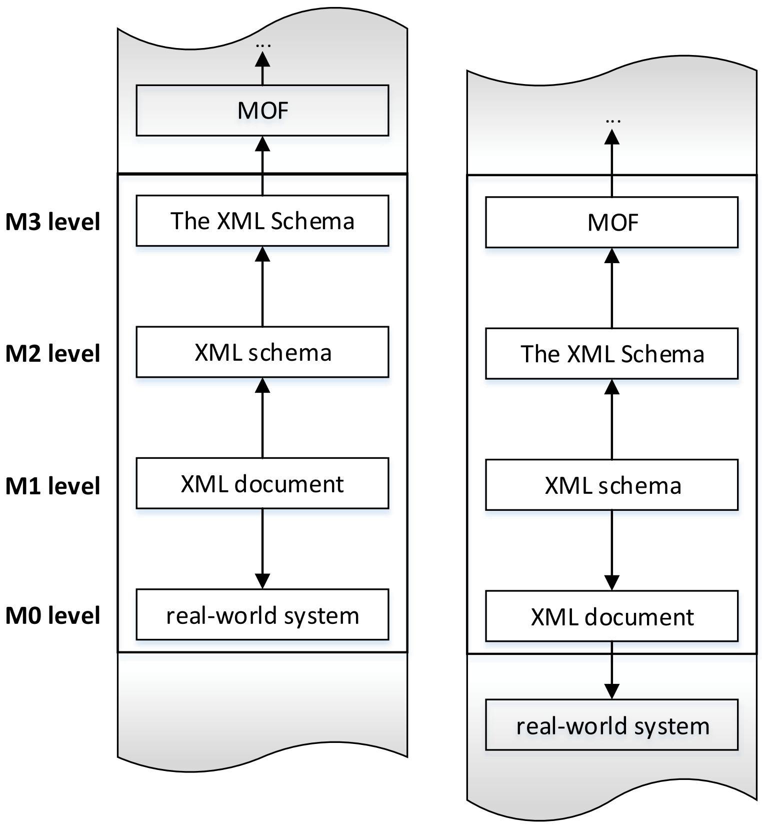


Figure 5: Movable abstraction viewport

At the left hand side of Figure 5, we present an example of a four-level viewport with the real-world system at the M0 level. As a real-world system is the one that we need to represent with a model, at this level no abstraction is applied. Properties of a real-world system can be represented with an XML document which is in turn specified using an XML schema. In the domain of XML modeling, every schema is described with the XML Schema language that is self-described and is located at the M3 level of the four-level conjecture. This is an example of a viewport usually encountered in the domains where users are interested in modeling a system but looking at the lower levels of abstraction. This can be seen as a concrete modeling space.

Our first use case (cf. Section 6.1) fits to this category. We are interested in the data exchanged between machines and information systems and transforming the data sent by machines to data that can be understood by information systems. Such data is a model of the physical world as measured by sensor machines. Therefore we are using the lower abstraction viewport and at the lowest level a *real* system is found.

In some other cases, it might be necessary to move the viewport up the abstraction ladder. In the linguistic and schema-centric domains ,e. g., XML domain, the model itself is a schema document that constraints data found at the M0 level. At the M2 level one can find a definition of the language for schema specification while at the M3 level the generic framework for language definition such as MOF is found. Our second use case (cf. Section 6.2) concerns integration of two language workbenches. At the lowest level of abstraction a model is found as an instance of the appropriate schema. Therefore, the system in this case is *language based*. This is an example of a conceptual modeling space.

The concept of a modeling space is inspired by a more generic and inherently broader concept of Technical Space (TS). According to Bezivin, Kurtev et al. [40, 116] the technical space may be defined as: "*a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities*". This definition does not define TSs precisely, but it can be defined over the notion of a modeling space, too.

A TS is "*a means for grouping modeling spaces that have something in common or simply need to interact*" [184]. Most often a TS is built around a single modeling space, whereas the role of other modeling spaces in the same TS is supportive (e. g., implementation) or implicit (e. g., documentation). For example, the Ecore modeling space is at the center of the EMF TS. However, the EMF TS also partially includes other modeling spaces: XML and EBNF in the area of XML Metadata interchange (XMI) representation, EBNF in the area of repository implementation, an implicit modeling space that includes literature. On the other hand, one modeling space which is peripheral to one TS may be the central modeling space of another TS. The bridge connecting two modeling spaces is also a means for connecting surrounding technical spaces.

Some examples of computer science TSs include CSV TS, XML TS, and EMF TS. Examples outside of the computer science field include house construction TS in civil engineering field, where the technical space comprises materials, rules, techniques, and building skills required to construct a proper place for living, and car construction TS in mechanical engineering field with similar TS elements. In this thesis we consider only computer science and software engineering TSs with additional Industry 4.0–specific TSs defined in the next paragraph.

As the MDSD approach to model integration relies on model transformations which are based on the four-level conjecture, integrated TSs have to be represented in a suitable way. TSs only deal with the virtual representation of SUS entities and as such most of them may be considered to have three levels presented at the right hand side of Figure 6. Together with the SUS (M0), TSs form the appropriate four-level structure suitable for an MDSD integration approach. Examples of the frequently used three-level TSs are presented at the right hand side of Figure 6. In the proposed research we will focus only on three-level TSs.

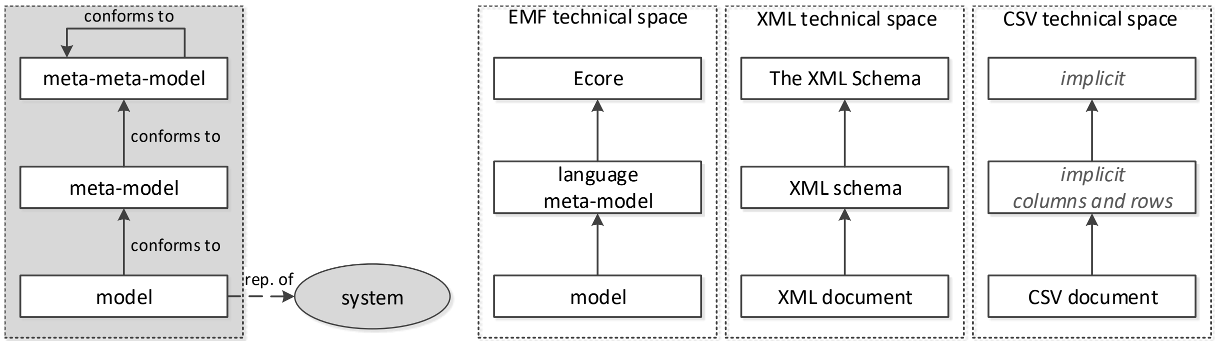


Figure 6: Three-level TS architecture with examples

2.2.4 *Model Transformations*

Once specified, no model can continue to exist unchanged or isolated, and therefore operation are often applied on them. Such operations are called *model transformations*. Analogously to Wirth's well known equation [199] designed for general purpose programming languages "*Algorithms + Data Structures = Programs*", the following equation may be applied in MDSD approaches "*Models + Model Transformations = Software*" [32].

If we consider the notion of a modeling space, a transformation can be defined as a bridge between the two modeling spaces. Such transformations are also called Model-to-Model (M2M) transformations as they are executed on top of models and also produce models as their output. The transformation can be considered as a model itself in a separate, transformation modeling space which is independent of transformed modeling spaces [184, 111, 38]. This complies with the basic MDSD principle that *"Everything is a model"*.

According to [184], there are two usage scenarios for different modeling spaces: (i) *parallel spaces*, in which two modeling spaces can model the same part of a real world system but from different perspectives, and (ii) *orthogonal spaces*, in which one modeling space models the concepts from another modeling spaces as they were real-world entities, i. e., one modeling space is represented in another modeling space. In order to exchange models between parallel spaces, model transformations need to be specified and executed. These transformations are also models, and should be developed in a modeling space that can represent both the source and the target modeling spaces [184]. The main concepts for the development of model transformations and their relations to the meta-modeling concepts are given in Figure 7.

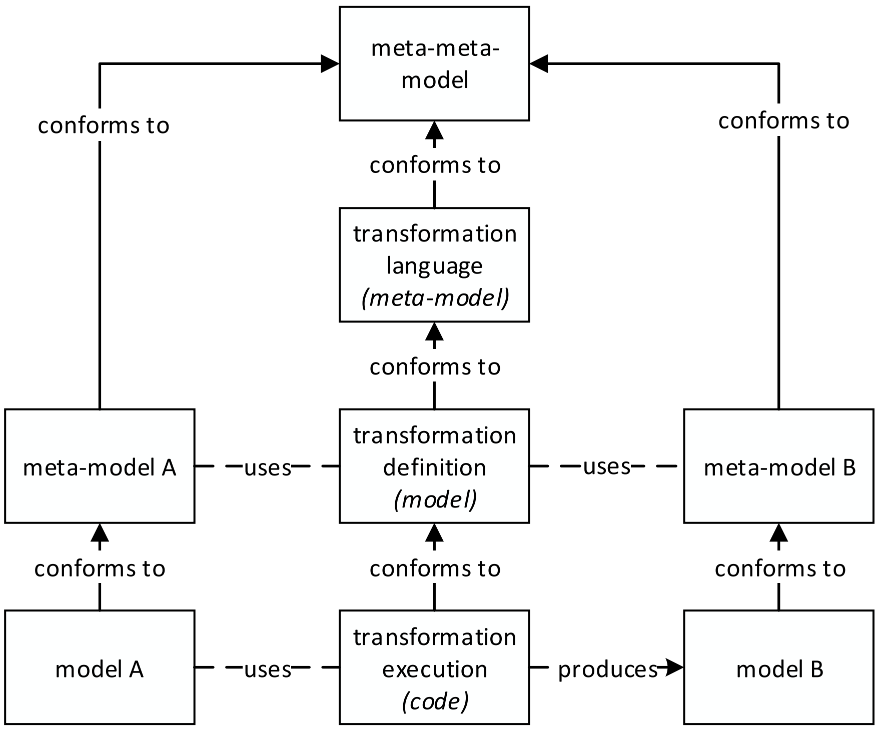


Figure 7: Model-to-Model transformations: role and definition

Model transformations are specified at the level of meta-models but are executed at the model level. This way, a transformation may be specified once, for a combination of source and target meta-models, and then executed multiple times for each source model that conforms to the source meta-models. The output of such a transformation is a model that conforms to the target meta-model. Model transformations are specified by using a transformation language which can also be characterized as a DSL for the domain of model transformations. Therefore the same language development rules could be applied for the development of transformation languages as well as for the development of DSMLs. Also, a transformation specification can be considered as a model on its own as it conforms to a meta-model which is in fact the abstract syntax of the transformation language. In turn, this meta-model conforms to a meta-meta-model that defines the modeling space in which the transformation is specified.

In the taxonomy of model transformations [139] multiple transformation classifications are proposed and some important transformation properties are explained. In the rest of this section we give an overview of the most important classifications of model transformations important for our approach laid out in the Chapter 5.

The first classification is based on a number of source and target models participating in the transformation. It is possible that a transformation has multiple source and multiple target models. Depending on source and target models' multiplicity, there are: *one-to-one*, *one-to-many*, *many-to-one*, and *many-to-many*.

Source and target models need not to reside at the same abstraction level. Based on the abstraction level difference between source and target models, model transformations can be classified as *horizontal* and *vertical*. In horizontal transformations both source and target models reside at the same level of abstraction. Typical examples of horizontal transformations are refactoring and migration from one programming language to another. On the other hand, vertical transformations are created between models specified at different levels of abstraction. Examples of these transformations include refinement and code generation.

According to the output of the transformation all model transformations can be classified as *Model-to-Model (M2M)* and *Model-to-Text (M2T)* transformations. The output of the M2M transformation is a model that conforms to the appropriate meta-model. During the specification of M2M transformations, both source and target meta-models are considered. During runtime, target meta-model is used to instantiate the produced model. The output of the M2T transformation is a model, usually text, e. g., free form text or programming code. In another words, generated text is implicitly considered to be a model of reality but without explicit meta-model. A model-to-text transformation is often implemented as a code generator [45, 106].

Source and target models need to be expressed in some modeling language. *Endogenous* transformations are specified between models conforming to the same meta-model, i. e., that are expressed in the same modeling language. Examples of endogenous transformations are optimization, refactoring, and simplification of a model. *Exogenous* transformations are specified between models expressed in different modeling languages and thus conforming to different meta-models. Examples of exogenous transformations include reverse engineering, migration, and code generation.

The direction of transformation can be criteria for the classification, too. *Bidirectional* transformations do not impose direction of transformations when specified. If a transformation is defined between models A and B, it allows for the creation of model B from model A, i. e., forward transformation, and creation of model A from model B, i. e., backward transformation [32]. In general, it is required to write less transformation rules, as the bidirectionality allows for the same set of rules to be used in both directions. *Unidirectional* transformations only provide transformations in one direction, i. e., from a source model to a target model.

An interesting aspect of the "*everything is a model*" statement is that transformations can be seen as models themselves [32]. This implies that operations can be executed on top of transformation models in order to change or adapt them. These operations are seen as transformations that transform other transformations. They are known as *High-Order Transformations (HOTs)*. Example of such a transformation is refactoring of other transformations in order to improve their structure.

Our mapping language, which is used to integrate different TS (cf. Chapter 5), can be seen as a model transformation language that transforms a model from a source TS into a corresponding model from a target technical space. The transformations specified with this language are, in general, unidirectional, many-to-many, M2M transformations that can be both exogenous and endogenous. These transformations can be considered as models and we allow reuse of model elements between different integration scenarios. Therefore, the reuse of elements and automatic creation of new transformation specifications can be considered as a HOT.

2.3 SUMMARY

In this thesis we present an approach to system integration based on the model-driven development principles. Therefore, in this chapter we have introduced the main terminology and concepts from the domains of system integration and model-driven development which constitute the theoretical foundations of our work.

Our approach aims to provide interoperability by using the integration method based on a common data format. Our approach comprises two steps that correspond to the common data format method: (i) we convert original data format to a generic data format used internally by our tool and then (ii) we create transformations, i. e., mappings, between the two generic data representations which can be executed and thus provide integration between disparate system parts.

Our integration approach presented is based on the notions and principles from the model-driven development domain. We consider each of the integrated system parts as a separate three-level Technical Space (TS). Both input and output generic data formats may be considered as models corresponding to the generic tree meta-model (cf. Chapter 5). The transformations between these two models are created by using a graphical Domain-Specific Language (DSL) that we have developed solely for this purpose. These transformations can be further classified as a unidirectional, many-to-many model-to-model transformation language that can be both exogenous and endogenous. All created transformations can be seen as models themselves. Therefore, all operations on these transformation models, like reuse, validation and refactoring, are considered as HOTs.

In the next chapter we give an overview of the current state of the art in the domain of integration and industrial automation as it is our main application domain.

3 STATE-OF-THE-ART

In this chapter we present an overview of state-of-the-art in the integration and schema consolidation domains. In general, we can distinguish between the following two mechanisms: standardization and transformation. Standardization can be defined as a development process of a standard which avoids heterogeneity a priori by defining a common structure of data being exchanged. For integration in the context of Industry 4.0, there is a variety of standards which overcome both inter-space and intra-space heterogeneity. Some of the important and novel standards in the age of Industry 4.0 are presented in Section 3.1. However, in practice, such standards and data structures are frequently adapted to a specific domain, manufacturer, or machine. Thus, a mapping or transformation approach is necessary to overcome the heterogeneity between different data structures. The aforementioned unification mechanisms are not mutually exclusive. A proprietary structure can be mapped to a standard one by using a transformation.

Traditionally, transformation-based integration approaches were classified under the names *Schema matching*, *Schema mapping*, *Ontology matching*, and *Ontology alignment*. Schema-based approaches from these categories are presented in Section 3.2. As our approach concerns the model-driven integration of technical spaces, we have surveyed existing literature on the topics of model-driven schema matching and model-driven integration of both industrial and non-industrial software systems. This specific subset of schema matching approaches is presented in Section 3.3. Ontology-based approaches are presented in Section 3.4, while in Section 3.5 we conclude this chapter.

3.1 INDUSTRY AUTOMATION REFERENCE MODELS AND STANDARDS

Most of the integration problems can be solved by introducing standards that are usually made for a particular layer of the manufacturing system. These system layers and standards are often organized in a form of reference architectures to allow easier classification and separation of concerns. In this Section, we only present several contemporary standards and reference architectures closely related to the proposed research and Industry 4.0 vision.

The Reference Architecture Model for Industry 4.0 (RAMI 4.0) [204], presented in Figure 8, is developed by several German institutions. It illustrates the connection between IT, companies, and products life cycle through a three-dimensional space in which each dimension represents a layered view on these concepts. The left horizontal axis represents the life cycle of facilities and products. Furthermore, a distinction is made between "types" and "instances". A "type" becomes an "instance" when design and prototyping have been completed and the actual product is being manufactured. The main building block according to such a view is an *i4.0 component* representing a unified description of assets (real world objects participating in the development process), products, and networking information. A management shell should be implemented for each of the i4.0 component assets (e. g., sensors, actuators, and other machines) and stored in a data store. The management shell may be seen as a virtual representation off an asset containing both status information and data produced by the i4.0 component. The reference model allows the representation of physical devices in the form of management shells during the entire life cycle. Along with the right hand horizontal axis the location of the functionality and responsibilities are given in the hierarchical organization. The reference model broadens the traditional hierarchical levels by adding the Product level at the bottom, and the Connected World that goes beyond the boundaries of the individual factory at the top. In addition, RAMI 4.0 allows the description and implementation of highly flexible concepts. This leverages the transition process of current manufacturing systems to Industry 4.0 by providing an easy step by step migration environment. Left vertical axis represents IT perspective which is comprised of various layers such as business, functional, and information. These layers corresponds to the IT way of thinking where complex projects are decomposed into smaller manageable parts.

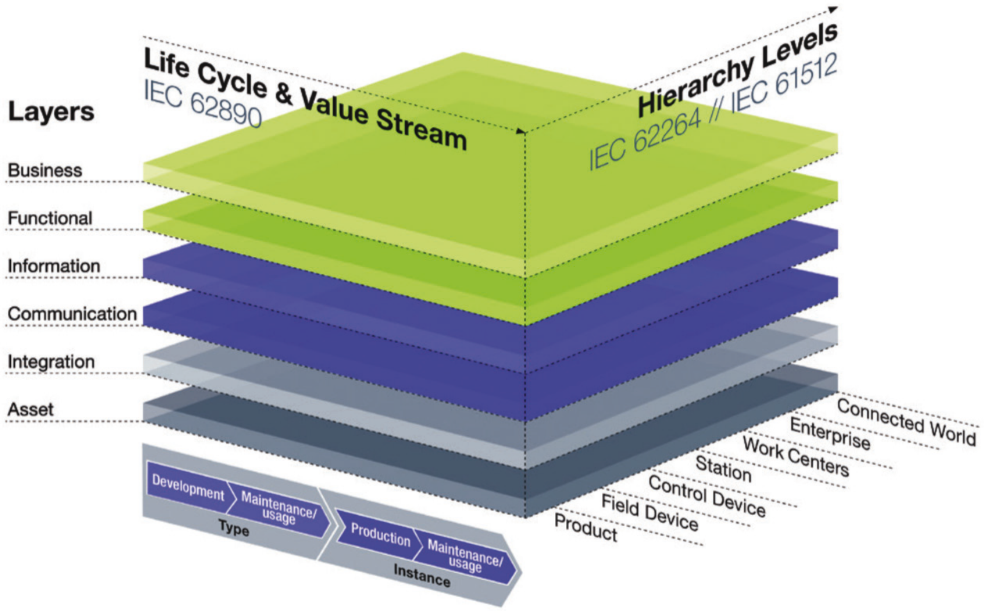


Figure 8: Reference Architecture Model for Industry 4.0 (RAMI 4.0), source [204]

Complementary to the proposed reference model, many standards are currently used at different organizational and technological layers of an enterprise. As the focus of the future research will be the integration of machines and Information Systems (ISs) in the context of Industry 4.0, we describe the following two standards that target similar issue: Open Platform Communications Unified Architecture (OPC UA) [135] and Automation Markup Language (AML) [62].

AML is a standard developed for the field of production systems engineering and commissioning. The data exchange format proposed for AML is an XML schema-based data format developed in order to support the data exchange in a heterogeneous engineering tools landscape. The goal of AML is to allow interconnection of engineering tools from different disciplines, e. g., mechanical plant engineering, electrical design, process engineering, process control engineering, Human-Machine Interface (HMI) development, PLC programming, and robot programming. AML stores engineering information which structure follows the object-oriented paradigm and allows modeling of physical and logical plant components as data objects encapsulating different aspects. Typical objects in plant automation comprise information on topology, geometry, kinematics, and logic, whereas logic comprises sequencing, behavior, and control. Therefore, an important focus is on the exchange of object-oriented data structures, geometry, kinematics, and logic. AML combines existing industry data formats that are designed for storing and exchanging different aspects of engineering information. These data formats are used on an "as-is" basis within their own specifications and are not branched for AML needs [74].

OPC UA is a platform-independent standard that defines a machine-to-machine communication protocol. It is applicable to manufacturing software in application areas such as Field Devices, Control Systems, MESs, and ERPs. These systems are intended to exchange information and to use command and control for industrial processes. OPC UA defines a common infrastructure model to facilitate this information exchange. It specifies: (i) the information model to represent structure, behavior and semantics, (ii) the message model to interact between applications, (iii) the communication model to transfer the data between end-points, and (iv) the conformance model to guarantee interoperability between systems. OPC UA supports robust, secure communication that assures the identity of actors in the process and resists attacks. Information is conveyed using OPC UA-defined and vendor-defined data types. OPC UA can be mapped onto a variety of communication protocols and data can be encoded in various ways to trade off portability and efficiency. The OPC UA specifications are layered to isolate the core design from the underlying computing technology and network transport. This allows OPC UA to be mapped to future technologies as necessary, without negating the basic design. Data can be encoded in the form of XML document or an UA Binary representation [77].

The creation of these standards aims to provide a detailed description of the appropriate component of the manufacturing process. As production processes evolve, these standards must grow accordingly to reflect introduced changes. This makes the interoperability between the different standards or even different versions of the same standards a problem to solve.

3.2 SCHEMA-BASED INTEGRATION APPROACHES

The proposed research aims at integrating three-level TSs at the meta-model level. Traditionally such approaches were named schema matching approaches [23]. Although the schema matching approaches originate from the domain of relational databases and XML systems, many of the algorithms, approaches, and principles are still valid in general purpose integration.

In the book [23], edited by Bellahsene et al., a survey on schema matching techniques and approaches may be found. This survey focuses on the usage of semantic matching to perform schema evolution and schema merging. It also gives an overview on the currently used matching approaches, visualization, versioning, and collaboration techniques. In this book, schema matching is defined as: "*the task of finding semantic correspondences between elements of two schemas*." Although closely related, schema matching should not be confused with schema mapping. According to Ten Cate et al. [181], schema mapping may be defined as "*a high-level, declarative specification of the relationship between two database schemas, typically called the source schema and the target schema*." Schema mappings are usually specified with a visual notation. Therefore, schema matching systems are not to be confused with the schema mapping systems, where the former one is concerned with (semi-) automatically providing a set of mapping elements but the latter comprises a tool that allows specification of mappings between source and target schemas where the mappings are taken as an input for executable code generators. Schema mapping systems often allow the manual specification of the mappings but may also contain schema matching modules that can (semi-) automatically assist users in finding the appropriate mapping candidates.

In the recent years, a number of additional surveys and evaluations of schema matching and schema mapping approaches were conducted. Do et al. and Rahm and Bernstein [58, 163] classify existing research work on schema matching by the type of the implemented matching approach. Shvaiko and Euzenat build upon these surveys in [175] and introduce more detailed classification. Based on their type, the following approaches to automatic schema matching are identified in these surveys: (i) *instance-level similarity approaches* that use instance data and identify patterns and other characteristics of the data in order to find matching schema elements, (ii) *single element data or semantic similarity approaches* that use isolated schema element information to find matches, (iii) *element structure similarity approaches* that find matches by comparing schema structures and structural patterns, (iv) *constraint similarity approaches* that compare constraints explicitly or implicitly defined over a set of schema elements, (v) *repository based approaches* that use the previously defined matches and apply it to a new context, (vi) *hybrid approaches* that directly combine several matching approaches to determine match candidates based on multiple criteria or information sources, and (vii) *composite approaches* that combine the results of several independently executed matchers, including hybrid matchers.

Another contribution of these surveys [58, 163] is the introduction of taxonomy of matching features with the aim to identify possible techniques for automation of the matching process. Rahm and Bernstein propose that a generic Matcher tool should have at least: (i) schema importers that convert schemas to a generic representation, (ii) generic match implementation language for the specification of matches between source and target schema elements, and (iii) a global repository for storing identified matches. Although the paper focuses on the web service and database integration domains, conclusions and identified concepts can be generalized and applied to other integration problems as well. The following approaches and tools are classified, described, and compared in detail: Learning Source Descriptions (LSD) [59], Semantic Knowledge Articulation Tool (SKAT) [147], DIKE [154], ARTEMIS [41], Cupid [134], Clio [145], Similarity flooding (SF) [138], Delta [43], Tess [122], Tree matching [195], Autoplex [25], Automatch [24], COMA [56], Embley et al. approach [71], GLUE [60], S-Match [79] and TransSCM [146]. While most of the aforementioned tools aim to solve a matching problem in a specific domain, a few approaches like Clio, Cupid, COMA, and SF, try to address the schema matching problem in a generic way that is suitable for various application domains.

In their subsequent survey, Bernstein, Madhavan, and Rahm [28] cover ten years of research and advancement in the field of schema matching that have passed from their initial survey [163]. They present the new emerging approaches to matching elements: (i) *graph matching* that compare schema structures by using graph-based algorithms, (ii) *usage-based matching* that analyzes tool logs for user matching activities, (iii) *document content similarity* that groups instance data into documents and match them based on information retrieval techniques, and (iv) *document link similarity* where concepts in two ontologies are regarded as similar if the entities referring to those concepts are similar. In addition to these information-based techniques, in [28] a plethora of new techniques for creating hybrid and composite approaches is identified with a note that a trend can be spotted of switching from pure machine learning approaches to ontology alignment and matching. Several new tools have also been covered with the survey: COMA++ [19], ASMOV [94], Falcon-AO [89], RiMON [124], AgreementMaker [44], and OII Harmony [173]. The authors argue that due to the availability of large numbers of schemas on the web, a holistic matching approach is becoming quite appealing and it is needed more than ever before. However, the existing approaches have been applied in the domains where the schemas are small, with just a few, well-understood underlying concepts. Also, it can be observed that only a few of aforementioned matching technologies and tool has made it into commercial offerings. Therefore, there is still a need for a better and practically usable approach.

Based on the findings of their surveys and evaluations of schema matching approaches, Aumueller et al. and Do et al. [19, 57] developed a schema matching tool named COMA++. The tool focuses on the integration of large and complex XML schemas. By a notion of large and complex XML schema, the authors consider schemas with more than 100 schema elements with user types and complex structures defined in it. COMA++ uses composite matchers, combining the power of simple matchers into one that is usually more efficient or more suitable for a specific application domain. The tool follows a divide and conquer approach, where the schema is modularized and modules are mapped independently. Afterward, a full mapping is created by merging the individual module mappings.

Unlike aforementioned surveys on general matching techniques, a survey focusing on XML schema matching is provided by Agreste et al. [2]. The authors significantly extend the scope of published surveys with a description of new techniques particularly tailored for the XML domain. Agreste et al. argue that in order to have a best fit matching technique in the domain of XML, the matching tools should be specialized for that domain and use all of its peculiarities. This way, the matches are found more efficiently, matches are more appropriate to the domain, and the greatest advantage is that the schema element semantics can be identified in a more precise way. They also provide a template, called *XML Matcher Template*, which proposes the main components and their roles and behaviors in any XML matcher. Agreste et al. also discuss several commercial prototypes designed to identify mappings between XML schemas. These prototypes are then classified by using the degree of correspondence to their XML Matcher template.

We have also identified several schema matching approaches not covered by the aforementioned surveys. At the Faculty of Technical Sciences, University of Novi Sad, a tool named Integrated Information Systems Studio (IIS*Studio) is developed with one of its core function being the integration and consolidation of relational database schemas and subschemas. The main purpose of IIS*Studio is information system development which comprises the conceptual database schema design, based on the *form type* concept [70, 131], and development of appropriate business applications. IIS*Studio comprises three main tools: IIS*Case [132, 133], IIS*UIModeler [21], and IIS*Ree [5]. IIS*Case is the core tool of IIS*Studio and provides the following functionality:

1. conceptual modeling of database schemas, transaction programs, and business applications of an IS [131, 157, 158, 159],
2. specification of check constraint at the level of a conceptual model [152],
3. automated design of relational database subschemas in the 3rd normal form (3NF) [128, 129],
4. automated integration of subschemas into a unified database schema in the 3NF [128, 129, 130, 166, 167, 168],
5. automated generation of SQL/DDL code for various database management systems (DBMSs) [4], and
6. automated generation of executable prototypes of business applications.

In the case of large systems being developed by the incremental approach, a system is decomposed into several subsystems that are modeled independently and usually by different designers. The process of independent design of subsystems and their database schemas may lead to collisions in expressing the real world constraints and business rules. Therefore, in IIS*Case, the process of system integration is not just a mere unifying of its subsystems. It is based on detecting and resolving all the formal constraint collisions. Luković, Ristić et al. [128, 129, 130, 166, 167, 168] proved that, at the level of relational data model, it is possible to automatically detect formal collisions of database constraints embedded into different subschemas, where each subschema represents a database schema of a sole IS subsystem. If collisions are detected, at least one subschema is formally not consistent with the current version of a database schema of a whole system. Programs made over inconsistent subschemas do not guarantee logically correct database updates. Therefore, the authors created and embedded into IIS*Case algorithms for detecting formal constraint collisions for the most often used constraint types at the level of relational data model. Besides, they embedded into IIS*Case a number of collision reports that assist designers in their resolving. By this, the database schema integration process based on the approach of a gradual integration of subschemas into a unified database schema is supported by IIS*Case in a large extent.

At the abstraction level of Platform Independent Modelss (PIMs), IIS*UIModeler provides conceptual modeling of common User Interface (UI) models, as well as business applications that include specifications of: (i) UI, (ii) structures of transaction programs aimed to execute over a database, and (iii) basic application functionality that includes the following "standard" data operations: read, insert, update, and delete. A PIM of business applications is combined with a selected common UI model and then automatically transformed into the program code. In this way, fully executable application prototypes are generated. IIS*Ree is a model-driven re-engineering tool that provides a set of extractors and model-to-model transformations that extract and transform relational database schemas to a conceptual model based on the form type concept. Once the conceptual model is adapted to new requirements, a set of new model-to-model transformations and code generators is used to generate relational database schema and deployment scripts.

Bernstein et al. [27] introduce a solution that aims to bring the schema mapping technique to an industrial environment. They present a prototype of a customizable schema matcher called PROTOtype PLAtform for Schema Matching (PROTOPLASM). PROTOPLASM comprises three layers: (i) an import layer in which the mapped artifacts are transformed into a common internal representation based on XML, (ii) operation layer which comprises concepts needed to build a schema matching strategy, and (iii) a graphical language layer in which the graphical representations of operational concepts are combined into matching strategy scripts which are then executed. Similarly, Raghavan et al. [162] propose a solution, named SchemaMapper, which uses a hyperbolic tree instead of a linear tree representation. In their opinion, the hyperbolic tree contributes to a faster human-performed search for an element that is needed for a matching process. Another difference between PROTOPLASM and SchemaMapper is that the latter uses a tabular mapping representation instead of line-based one, which is traditionally used. While the line-based representation may lead to overcrowded diagrams in the case of large schemas, tabular representation leads to more compact views. A drawback of the SchemaMapper is reflected in the fact that it is focused only on the XML technical space.

Alexe et al. [6, 7, 10] propose an approach to schema mapping in the domain of relational database schema integration. Unlike most of the previously listed solutions, that load entire source and target schemas and create high-level mappings between them, Alexe's approach named "divide-design-merge" allows splitting source and target schemas into smaller parts, creating mappings between these parts, and merging all partial mappings into a whole as the final step. This approach has been supported by three tools that authors have developed. Eirene [12] is a schema mapping design tool that takes as an input a set of data examples provided by the user. In turn, Eirene outputs a schema mapping that "fits" the set of data examples, if such a schema mapping exists. Afterward, a user can interact with the Muse [8] tool to refine and further design schema mappings through the use of data examples. Finally, in the merge phase, a global schema mapping is generated through the correlation of the individual schema mappings by using a MapMerge [13] application.

Muse is one of the earliest systems that adopted a different approach to schema-mapping design. This approach uses instance data examples to infer mappings between schemas according to which these data are formatted. In these approaches, schema matching does not rely on a high-level schema mapping language, but on algorithms that analyze instance data to find data constraints or patterns which are often very good indicators of the similarity between the appropriate schema elements. This type of an approach to schema mapping has been also proposed by [10, 11, 47, 81, 181, 202]

In [63], Duchateau and Bellahsene present Yet Another Matcher (YAM). YAM is a self-tuning and extensible matcher factory tool that generates a best-fit schema matching algorithm for a specific integration scenario. Based on the generated matching algorithm schema element matches are then identified and proposed to a user. The *self-tuning feature* of this approach provides the ability to produce a matcher with appropriate, user-defined, characteristics for a given scenario. The *extensible feature* enables users of a matching tool to add new similarity measures and thus increase the overall effectiveness of the system. The goal of YAM is to alleviate users of a manual configuration of matcher similarity measures including the thresholds setup and iterative adjustment of these measures. YAM automatically tunes these parameters by relying on the implemented machine learning techniques. Similar techniques were implemented in MatchPlanner [64], which is based on the decision tree while, and eTuner [121] that performs the same job by employing a set of synthetic matching scenarios involving the schema being mapped. For each eTuner synthetic scenario correct matches are known in advance and thus it is possible to evaluate produced mapping configurations.

In addition to approaches and tools described in research papers, several patents have been filed concerning schema matching approaches, notations, and systems. Thomas [182] patented a schema matching system based on a tabular representation of schemas and mapping formulas. The proposed system displays instance data beside the appropriate schema elements in order to give the user better contextual understanding of the schema elements. Once the schemas are loaded and represented in a tabular layout, textual formulas can be specified to represent relations between source and target elements. In her second patent, Thomas [183] introduces the notion of a platform independent schema representation, named *conceptual model* which is a high-level representation of schema understandable to a domain expert. Other concepts proposed by the patent is similar to concepts of the patent presented in [182]. In [174], Seligman patents a semi-automatic schema matching approach based on a linguistic processing of schema elements. Element relations, i. e., matches, are discovered by analyzing element names with a machine learning algorithm that uses both generic and domain thesauri together with the list of frequently used abbreviations. Match probabilities are provided to a user who manually chooses the mappings he deems a best-fit. Patents [88, 169], filed by Hobbs and Robertson et al. respectively, propose notations and layout algorithms to be used in matching tools. Both patents propose that mappings are represented as lines with a central (algorithmic) part of the mapping being shaped as a box to allow easier handling and spotting. Hobbs also proposes an algorithm that handles drawing and layout of the mappings used while users create mappings, load previous work, or scroll the schema elements in their views.

3.3 MODEL-DRIVEN INTEGRATION APPROACHES

The MDSD promotes the development of software systems at different levels of abstraction, and DSLs play a prominent role to reduce development costs. As one of the most time-consuming and error-prone parts of introducing a new technology or a new functionality to the existing IT landscape is integration, by means of an appropriate DSL software engineers can design a software system that can later be integrated and deployed to a variety of specific platforms using automatic transformations. As the transformations are specified at the level of meta-model, i. e., data schema, transformation rules may be seen as schema matching rules. Therefore, MDSD transformation approaches may be seen as a subset of schema matching and mapping approaches.

Büttner et al. [35] present a model-driven approach to the data integration between government institutions in Germany. The integration approach is centered around the standardization of messages, interfaces, and models of data that are being exchanged. A compliance with the standards is regulated by a central governing body that governs the specification of meta-models, i. e., data formats, for different sectors in the German government. As different standards exist, integration is essential task that needs to be performed in order for the data to be exchanged. Therefore, integration processes need to be used at the meta-model level to allow transformation of messages and their communication to other German or European institutions. Büttner et al. have developed a central repository, named XRepository, that stores all meta-modeling concepts, well-formedness rules, and process and semantic specifications that together form standards. The XGenerator tool is used to produce artifacts that are used in the integration process. These artifacts are usually web service specifications that need to be implemented by software vendors to integrate their solutions with the system.

Agt et al. [3], Kutsche et al. [117, 118], and Milanović at al. [144] present a meta-modeling approach to the integration of heterogeneous distributed IT systems named BIZYCLE. The BIZYCLE integration process is based on multilevel modeling abstractions. The integration scenario is first modeled at the computation independent level, where business aspects of an integration scenario are described. The model is then refined at the platform-specific level, where technical interfaces of the systems that should be integrated are described. For each of the supported platforms: SAP, relational and XML databases, web services, XML documents, J2EE components, and .NET applications, a specific platform specific model is created. The automation of the integration process is achieved through model extraction, systematic conflict analysis process, and code generation. Reuse is supported at the model-level via BIZYCLE Repository [143], as interface descriptions, transformation rules, and semantic annotations can be stored and shared between projects and users.

Wimmer [197] developed a meta-model bridging framework and a graphical DSL that provides bridging of different technical spaces based on data mining techniques. The framework comprises a mapping view and a transformation view. At the mapping view level, a user defines mappings between elements of two meta-models using the provided DSL. Thereby a mapping expresses also a relationship between model elements, i. e., instances of meta-models. In Wimmer's approach, mappings between meta-model elements are defined with mapping operators which are considered as processing entities encapsulating a certain kind of transformation logic. A set of applied mapping operators, also called a mapping model, defines the mapping from a left hand side (LHS) meta-model to a right hand side (RHS) meta-model. Thus, the mapping model declaratively describes the semantic correspondences on a high-level of abstraction. The transformation view is capable of executing the defined mapping models. During the execution, a mapping operator takes as input elements of the source model and produces as output semantically equivalent elements of the target model.

Huh et al [90] developed Marama Torua, a tool supporting high-level specification and implementation of complex mappings of data schemas. Complex mapping relationships are represented in multiple notational forms and users are provided with a semi-automated mapping assistance for large models. Multiple views are implemented in order to ease the process of mapping specifications for all levels of source and target schema complexity. The tool supports creation of mappings between any two technical spaces. However, if the import tool has not been already developed for a certain technical space, a user must develop it manually and map a data schema to a generic tool structure. Marama Torua comprises a set of Eclipse plug-ins allowing close integration with other tools such as schema browsers.

There are several DSLs and frameworks that are not directly related to the schema mapping, but fit better to the fields of schema matching and enterprise application integration. Vuković et al. [190, 191] present a language called Semantic-Aided Integration Language (SAIL). This language allows for the matching components to be described, generated, and used in their framework without having to be implemented in a general purpose programming language and are available without having to rebuild the entire application. The aim of the developed matching framework is to automate some of the steps in conflict resolution of the matching process. Interfaces and their elements can be semantically described using ontologies in order to facilitate this automation. Although the approach itself is based on the ontology alignment principles, the SAIL domain specific language is used to specify matching algorithms and follows all the principles of the MDSD methodology.

Another domain specific language, named Highway, is developed by Kovanović et al. [115]. Highway is developed as an internal DSL in the Clojure programming language. It may be used for implementing enterprise application integration solutions in a technology independent and functional manner. Highway uses functional programming techniques in order to simplify enterprise application integration development.

Sleiman et al. [176] propose a DSL called Guarana and a software tool to design and automatically deploy integration solutions in order to reduce integration costs. Guarana provides a set of domain specific constructors to design integration solutions. It provides an expressive graphical notation for these constructors, which allows a user to visually design an integration solution. Functions and mappings are all displayed on the same diagram thus giving a good overview of the general solution.

The Federated USer Exchange (FUSE) approach [194] represents a domain-aware approach to user model interoperability. It consists of a manual mapping process and an automatic translation process. Both processes contain two domain aware mechanisms: (i) a canonical user model and (ii) user model mapping transforms, which tailor the processes to specific domains. All mappings are first created with the canonical user model as a target. This model represents a consistent shared user model. The user model mapping transforms are mapping components specifically created and used for mapping between different user models via the canonical model. This approach differs from existing generic approaches because it incorporates domain knowledge in new processes and tools to support complex user model interoperability tasks in multiple overlapping domains.

Wischenbart et al. [200] employed a MDSD approach to the integration of data collected from social networks. Although many social networks on the Web allow access via dedicated APIs, the extraction of instance data for further use by applications is often a tedious task. As a result, instance data transformation to Linked Data in the form of Ontology Web Language (OWL), as well as the integration with other data sources are needed. This paper proposes a model-driven approach to overcome data model heterogeneity by automatically transforming schemas and instance data from JSON to OWL/XML. Authors specify a set of transformations that transform an input model, i. e., the model of JSON messages collected from social network APIs, to the OWL model, i. e., ontology used for representing social network data instances and their semantic.

In addition to aforementioned approaches, M2M transformation languages can also be seen as possible means to integrate different TSs. M2M transformation languages are specified at the level of a meta-model but are executed at the model level. Therefore it is required to extract or use existing meta-models from the integrated TSs. Once meta-models are obtained, transformation rules may be specified with one of the transformation languages. A definition and a classification of M2M transformation languages is given by Mens et al. [139] and the overview of the selected visual transformation languages may be found in our previous paper [52]. The advantage of these languages is that they are supported out of the box with well defined notations and semantics, they are usually declarative, and as we deal with the three layered technical spaces, meta-models already exist or can be easily extracted to a desired environment. However, the disadvantage of these languages is that they may be seen as general purpose mapping languages, and they are not well suited for the domain of integration thus the transformations may be verbose and hard to read and maintain.

3.4 ONTOLOGY-BASED INTEGRATION APPROACHES

What is known as schema mapping or schema matching in database and artificial intelligence domains, in semantic web community it is known under the name Ontology Alignment [67] or Ontology Matching [73]. The task of these approaches is to find groups of elements sharing the same semantics. Majority of the tools presented in Section 3.2 can also be applied to the ontology alignment process. Therefore, there is no clear line that separates these approaches and fit them into a single category. For example, although both approaches described in [190, 200] may be seen primarily as MDSD approaches, they rely on the use of ontology alignment techniques and principles to find best match candidates.

Unlike schema matching approaches that usually comprise techniques for guessing the meaning encoded in the data schemas, ontology matching systems try to exploit knowledge explicitly encoded in ontologies. In their survey [175], Shvaiko et al. focus on the comparison of the following ontology alignment solutions: Naive Ontology Mapping (NOM) [69], Quick Ontology Mapping (QOM) [68], OWL Lite Aligner (OLA) [72], Anchor-PROMPT [151]. Many other ontology alignment solutions are compared in a survey by Ardjani et al. [16]. The ontology alignment is performed according to a strategy or a combination of techniques for calculating similarity measures by using a set of parameters, e. g., weighting parameters and thresholds, and a set of external resources, e. g., thesauri and dictionaries. As a result, a set of semantic links between ontology entities is obtained. In addition to the tool comparison, Ardjani et al. introduce a classification of the ontology approaches based on the similarity measurement methods that are used. The identified methods are as follows: (i) *terminological methods* that are using terms, strings, and text for comparison, (ii) *structural methods*, which calculate the similarity by exploiting structural information, (iii) *extensional methods*, which infer the similarity between two entities, especially concepts or classes, by analyzing their extensions, i.e. their instances, and (iv) *semantic methods*, which include the methods based on an external thesauri and dictionaries and on deductive techniques that heavily rely on logical models, such as propositional satisfiability or description logic.

3.5 SUMMARY

In this chapter we presented state-of-the-art in the domains of data integration and schema consolidation. The approaches found in literature can be divided into two main categories: standardization and transformation-based approaches.

Standardization approaches aim at providing standard solutions, protocols, and processes for different layers of the integration process. As our focus is on the integration manufacturing industry, we have presented several most promising standards in this domain: RAMI 4.0, AML, and OPC UA. As production processes evolve and grow, these standards must be adapted to reflect introduced changes. Additionally, companies that have their own proprietary standards are often reluctant to adopt or participate in developing a standard due to some business reasons. Furthermore, even if a standard is adopted, in the industry where factories buy equipment to last for a long time, large portions of the device landscape are still communicating by non-standard protocols. All of the aforementioned issues make the interoperability still a problem to solve as it is necessary to integrate the different standards or even different versions of the same standard. The approach presented in this thesis aims to tackle this integration issue. As it allows integration of any two technical spaces, it is also possible to integrate existing standards including the ones presented in this chapter.

When a standard is not developed or a company chooses not to comply to a standard, the company may use a proprietary protocol and provide integration adapters to integrate with other companies' devices and information systems. These integration adapters are based on a transformation approach as they take an input data and transform it to a target data following a set of transformation rules. There are two main kinds of transformation approaches found in literature: (i) schema matching and mapping and (ii) ontology matching and alignment.

In regard to the schema-based matching approaches, we have analyzed a large number of papers and tools developed to that purpose. Although a plethora of schema matching tools were identified, only a small subset of them is still being developed and maintained. Most of the tools were developed just as prototypes implementing the necessary steps of the approach in order to validate the approach. Therefore, most of these tools cannot be used in a real-world scenario as they are either outdated or just not applicable. In regard to the applicability, most of the approaches deal with the schema-matching in relational database and XML domains as these domains are traditionally seen as the training ground for schema-based integration algorithms. Unfortunately, only small number of identified approaches is applied outside of these domains. Most notable approach that was applied in the industrial domain is the one mentioned in [27]. In contrast to these approaches which are mostly focused on matching source and target elements, our approach represents a wider view on the integration problem. Our approach considers all the the steps preceding the specification of the rules and also considers both manual and automatic integration mechanisms. Schema and ontology matching algorithms can be used by our tool in order to provide process automation.

While we surveyed the state-of-the-art in this chapter, in the next chapter we present a survey of integration and mapping tools. We identified a number of industry-ready solutions that were not covered by research papers, i. e., were not developed in academia.

4 A SURVEY OF MAPPING AND INTEGRATION TOOLS

In addition to the state-of-the-art of research work on the topics of schema matching, mapping, and integration presented in Chapter 3, our goal is also to identify visual mapping software solutions that are currently used in practice. As the goal of our research is to develop a visual mapping language for the integration domain, a survey on such tools will provide us with valuable information on current integration trends, best practices, and preferable characteristics of industry-ready solutions. Although we focus on visual mapping solutions, in this survey we also cover other widely used integration solutions even if they do not belong exclusively to this category.

To our knowledge there are no existing studies that evaluate and compare integration software based on the criteria proposed in this thesis, taking care of their domain coverage, language complexity, supported functions, reuse ability, availability, and other non-technical characteristics. In [164], Rathinasamy compares only several tools for the domain of asset management. This survey includes Altova MapForce which is evaluated in this chapter. Do et al. [58] compare several matching tools based on the matching algorithm performance. However, these tools are either still in the prototype phase or not maintained any more.

A framework aimed at generating test cases for visual mapping system evaluation was developed by Alexe et al. [9]. The framework generates test cases that are used to test if a mapping tool language covers a set of predefined functionality, i. e., basic mapping scenarios, such as copying a value from a source to a target model, constant value generation, and horizontal and vertical partitioning. Most of these mapping scenarios come from the domain of database schema integration. A full list and a detailed description of the proposed mapping scenarios may be found in [9]. Although such a framework could provide more insight in the domain coverage and the variety of functions a tool possesses, we opted not to use such a framework as we want to focus on the characteristics other than performance and in-detail analysis of each possible function of a mapping tool. Instead we are focusing on the type of syntax, reuse possibility, and complexity of user interaction as these are the most important characteristics for an industrial scenario.

In Section 4.1, we presented the criteria for choosing integration tools for the surveyed. Details and conclusions about each of the tools are presented in Section 4.2, Section 4.3, and Section 4.4 depending on the category to which a solution belongs. In Section 4.5 we discuss the results of the survey, while in Section 4.6 we conclude this chapter.

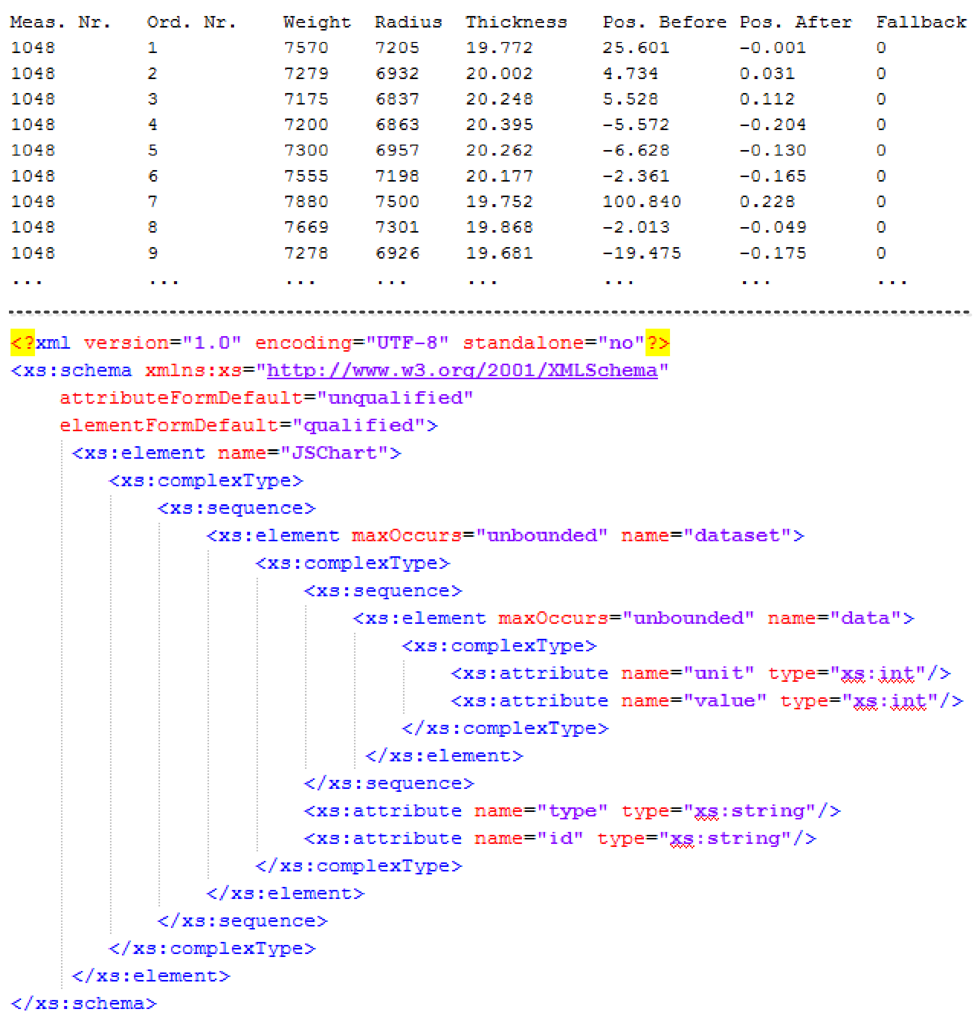
4.1 STUDY PREPARATION AND CHARACTERISTICS

During our initial state-of-the-art study, presented in Chapter 3, we have identified various integration software that follow similar approach to the one we propose. Identified software solutions constituted an initial set of software to be studied. After the initial set was identified, we eliminated solutions that did not fulfill the criteria of being recently updated, currently used, or available for download. Thus we have eliminated the software developed solely for the single or limited use and software developed just to support a scientific paper which was not maintained afterward.

After the literature study, we have searched the World Wide Web for phrases: "integration tool", "schema matching tool", "schema mapping tool", "migration tool", "ectl tool", "etl tool", and "bridging tool". This search resulted in a large number of industrial software solutions that were not identified in our literature study. This also allowed us to classify the solutions by relevance (closeness to the integration domain and number of search hits) and choose the most relevant ones for our study. This lead to the omission of large number of solutions, especially in the area of ECTL processes as ECTL is a very generic notion covering wide range of different processes. Such processes are also known as Extraction, Transforming, Loading (ETL) processes but we opt to use the longer acronym in this paper to avoid the name clash with Epsilon Transformation Language that is also known as ETL.

Most of the found mapping solutions omitted from this survey were used in a narrow application domain not of interest to our study. That makes these tools marginally important to our integration approach. All of the remaining solutions may be categorized into three groups based on their dominant application domain: (i) general mapping tools, (ii) XML mapping tools, (iii) ECTL tools. Each category will be further explained and its tools surveyed in appropriate sections in the rest of this chapter.

Benefits of such a study are twofold. On the one hand side, as we preformed the study prior to the development of our integration approach, we identified advantages and disadvantages of each solution, good practices, and usage patterns. Further, we identified main concepts that these solutions use to implement integration adapters and to allow knowledge reuse. In the



```
Meas. Nr.   Ord. Nr.    Weight  Radius  Thickness   Pos. Before Pos. After  Fallback
1048        1           7570    7205    19.772      25.601      -0.001      0
1048        2           7279    6932    20.002      4.734       0.031       0
1048        3           7175    6837    20.248      5.528       0.112       0
1048        4           7200    6863    20.395      -5.572      -0.204      0
1048        5           7300    6957    20.262      -6.628      -0.130      0
1048        6           7555    7198    20.177      -2.361      -0.165      0
1048        7           7880    7500    19.752      100.840     0.228       0
1048        8           7669    7301    19.868      -2.013      -0.049      0
1048        9           7278    6926    19.681      -19.475     -0.175      0
...         ...         ...     ...     ...         ...         ...         ...
```

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    attributeFormDefault="unqualified"
    elementFormDefault="qualified">
  <xs:element name="JSChart">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" name="dataset">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="data">
                <xs:complexType>
                  <xs:attribute name="unit" type="xs:int"/>
                  <xs:attribute name="value" type="xs:int"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
            <xs:attribute name="type" type="xs:string"/>
            <xs:attribute name="id" type="xs:string"/>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Figure 9: CSV document and XML schema examples

following sections we present each of the surveyed tools, while a comparative analysis is given in in Section 4.5. Another benefit of this study is that it represents a baseline for the evaluation of our approach. Once we developed the approach and the appropriate tooling support, we compare its concepts, performance, and user experience with the solutions that were studied.

The process of evaluating the integration software is based on a single example implemented in all identified solutions. The example emulates the integration between sensors, which send CSV data, and ISs, that can receive XML data. Both CSV document and appropriate XML schema file are presented in Figure 9. The concrete use case is the exchange of data between sensors that measure characteristics in the wafer production process and the IS module for data visualization. We are focusing just on the wafer weight measurements which need to be visualized using JSChart library [96] in a form of a line chart.

The integration process has the following three steps: (i) a sensor machine performs a weight measurement and sends a CSV document containing a measurement and appropriate metadata along with it, (ii) such a CSV document is then transformed to an XML file conforming to the JSChart schema, and (iii) the JSChart library reads the generated XML document and shows a line chart based on a *dataset* element and the data points specified as its child *data* elements of the XML document. The name of the chart is a value of dataset *id* attribute, while the type of the chart is set as the value of its *type* attribute. In this example, the *type* attribute should have the value "line" while the chart name is created from the name of the characteristic being measured, in this case "weight". While the *dataset* element is created once for each measured characteristic, data points are created for each measurement of that characteristics, i. e., for each row in the CSV document. The *value* attribute is mapped from the *weight* column of a CSV row, while the unit is similarly set from the value of the *ordinal number* column from the same CSV row. The abstract view on this transformation algorithm is presented in Figure 10.

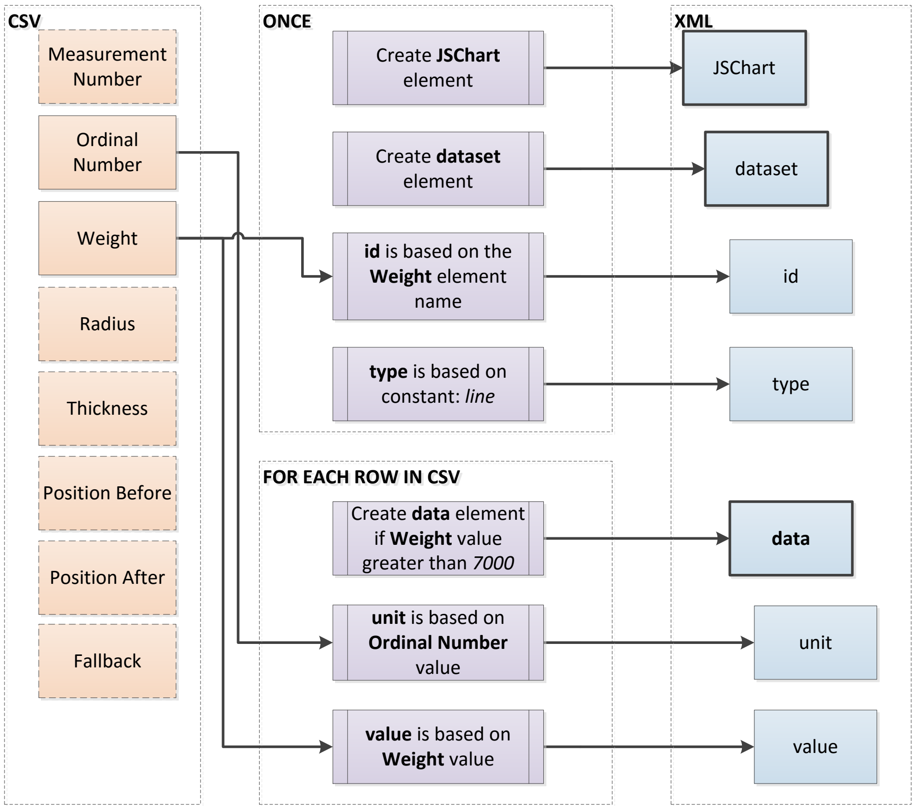


Figure 10: CSV2XML transformation rules

Characteristic	Description	Possible Values
Distribution	application type and license	**type**: desktop, web **license**: commercial, freeware, open source
Recentness	date of the most recent version	x
Domain: application	domain in which the tool is applied	x
Mapping approach	type of the mapping approach	direct, indirect
Language: mapping	type of the mapping language	graphical, textual, configuration-based, tabular
Language: expression	type of the expression language	graphical, textual, configuration-based
Code generation and execution	type of code generation and execution	internal, external
Reuse: concepts	type of reusable concepts	TS specification, function, mapping
Extensibility: technical spaces	TS extension support	yes, no, partial
Extensibility: functionality	type of extensible concepts	mapping language, expression language, code generator

Table 1: An overview of the observed tool characteristics

The reason of choosing this example is that it is one of our target use cases in which we want to apply our approach. The data has been anonymized and scaled to protect the intellectual property of the company that provided us with the data.

During the implementation of the example presented in Figure 10 we have observed a set of characteristics presented in Table 1. Following is the more detailed description of each of the characteristics:

- **Distribution**—software distribution mechanism and license type. Software can be distributed in a form of a *desktop* or a *web* application which can be privately and publicly hosted. Regarding the license type, we categorize applications as *open source*, *freeware*, and *commercial*.
- **Recentness**—software recentness. This characteristic describes the *year* in which the most recent version of the software was published and the *version* of the software which we have used in the survey.
- **Domain: application**—type of the main application domain. The *main application domain* as declared by the software developer or concluded based on the functions, expression language, technical spaces, and generators that the tool provides.
- **Mapping approach**—type of mapping approach which a user perceives while using the tool. Based on the way mappings are created, mapping approach may be classified as: (i) *direct*, in which source and target elements are directly mapped, and (ii) *indirect*, in

which a mediatory TS exists which serves as a central point of the mapping process onto which both source and target TS must be first mapped.

- **Language: mapping**—description of mapping language characteristics. The mapping language is used to establish correspondences between the elements of source and target technical spaces. In the survey we mainly focus on the type of the mapping language syntax which may be one of the following: (i) *graphical* comprising of lines and nodes, (ii) *textual* comprising of formula-like expressions, (iii) *configuration* in which the correspondences are specified through a set of dialogs and forms, and (iv) *tabular* in which the tables are used as the main means of establishing correspondences.
- **Language: expression**—description of expression language characteristics. In addition to defining just the correspondences with a mapping language, expression language must be used to define calculations and rules later used when executing transformations between the source and target technical spaces. The expression language comprises functions and syntactic rules. Based on the type of syntax, expression languages may be categorized as follows: (i) *graphical* comprising of lines and nodes, (ii) *textual* comprising of formula-like expressions, and (iii) *configuration* in which the expressions are specified through a set of dialogs and forms.
- **Code generation and execution**—description of code generation and execution characteristics. Based on the defined mappings and expressions, code generators are used to generate executable transformation code in a chosen programming language. Once the executable transformation has been generated it can be executed *internally*, within the tool, and *externally*, independently of the tool.
- **Reuse: concepts**—description of the reuse granularity, i. e., reusable concepts. Different parts of the integration project can be reused: (i) *TS specifications*, in systems which allow the specification of an arbitrary TS this is a necessity, (ii) *functions*, user-defined functions are stored globally and reused between projects, and (iii) *mappings*, the entire mapping specifications, including correspondences and expressions, are reused in new projects.
- **Extensibility: technical spaces**—possibility of adding a new technical space. This characteristic describes whether a user can extend (fully or partially) an integration tool to support an arbitrary TS or the tool only supports a predefined set of TSs.
- **Extensibility: functionality**—possibility of extending the tool functionality. This characteristic describes if a user can extend any of the following tool functionality: (i) *mapping language*, (ii) *expression language*, or (iii) *code generation and execution strategies*.

In the following sections we present the aforementioned characteristics for each of the surveyed tools. For each tool described in Section 4.2, Section 4.3, or Section 4.4, first we provide a general overview of the tool. After the overview, we present our conclusions about each of the 10 surveyed characteristics after which we give our opinion on main advantages and disadvantages of the tool. Screenshots will be presented only for the tools that served as a direct inspiration for our approach.

4.2 SURVEY OF GENERAL MAPPING TOOLS

In this section we present tools that may be used for a general mapping purpose. These tools are not generally limited to a single or just a few TSs. They often allow for the mapping between a larger number of TSs with a general mapping concepts and functions. Although they can be created for a specific application domain, they may be used in other integration and mapping scenarios. Unlike the tools from ECTL category, the tools from this category often cover just a Transform step of the ECTL process. The following tools are surveyed in this section:

- Altova MapForce [14];
- AnalytiX Mapping Manager [15];
- FME Desktop [76];
- OPC Router [92];
- Open Mapping Software [201];
- MetaDapper [171];
- MuleSoft Anypoint Studio [148]; and
- Vorto [66].

4.2.1 *Altova MapForce*

Altova MapForce is an industry-ready graphical data mapping, conversion, and integration tool. It is a part of a larger tool suite mainly focused on the development of multi-platform information systems. MapForce provides a graphical mapping language which is in our opinion easy to use and understand. Complex transformations can be broken down into smaller pieces which can be then organized in a chain of transformations. Transformation adapters can be generated in Java and C# languages and executed independently of the tool. Further, these adapters can be deployed on a dedicated machine with a sole purpose to execute mappings for provided input data and create transformed data as output. This is especially important for the industrial application of the tool. The observed characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application.

Recentness. Most recent version in 2016. Version: Altova MapForce 2016 Enterprise Edition.

Domain: application. Data integration and Information system development.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical mapping language comprising of mapping lines and operators specified between TS components. TS elements are represented in a uniform way as tree structures inside a floating container. Mappings may be specified between zero or more inputs and one or more outputs. MapForce also supports the mapping between multiple source and multiple target TSs.

Language: expression. Graphical expression language for the specification of operators. Operators and their expressions are built by connecting inputs and outputs of various previously defined functions.

Code generation and execution. Executable code may be generated in following languages: XSLT, XSLT2, XQuery (only for supported TSs), Java, C#, and C++. MapForce provides debugging of the mapping execution as well as the on-the-fly output preview.

Reuse: concepts. Reuse is only available at the level of user-defined functions. User-defined functions are stored in a global repository and can be reused between projects.

Extensibility: technical spaces. Not extensible directly. Built-in *FlexText* utility allows for parsing of the text files with regular expressions. In this way some data schemas from unsupported TS can be parsed but it does not guarantee that every TS can be parsed.

Extensibility: functionality. The tool is accessible from other development environments like Visual Studio and Eclipse and it is possible to extend the existing functionality from outside of the tool in a great extent. New generators can be added in the tool by using their template-based Spy Programming Language (SPL). In SPL templates it is possible to access mapping elements and procedurally specify algorithm for code generation. User is able to define new functions and use them across the projects. MapForce allows for creation of custom function libraries for Java, C# and C++. These functions can be used in the graphical mapping syntax similar to built-in functions.

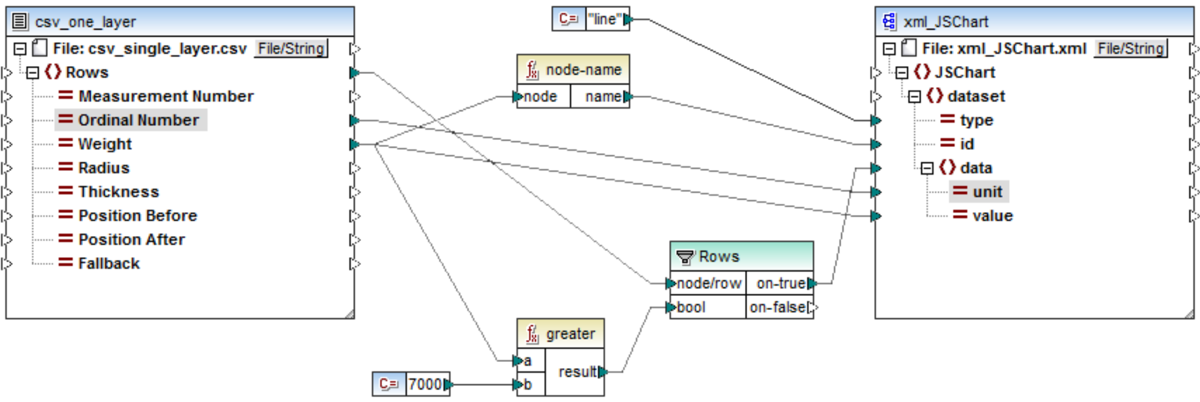


Figure 11: Altova MapForce: CSV2XML example

The main advantages of the Altova MapForce are its stability, functional completeness, customization, and extensibility. MapForce is one of the most stable and functionally complete systems surveyed in this chapter. Automatic execution and debugging of mappings inside the tool greatly increases the productivity of users. Users are able to follow the data flow from a source to a target and in that way identify potential problems with their mappings. Further, the tool provides automatic extraction of schemas from data files for some of the supported TSs. The *FlexText* utility is a convenient addition for parsing of textual files but its graphical way of representing regular expression can be too complex.

The main disadvantages of the tool are the lack of support for an arbitrary TS and its extensive reliance on the graphical syntax for the specification of mappings. Although the *FlexText* utility can be used to import data from a number of unsupported TSs, others cannot be parsed with just a regular expression. On the other hand, the graphical nature of components, mappings, and expressions heavily influences readability of mapping diagrams. Even in our use case where we have a small number of elements being mapped, it was not easy to read the diagram as there were many lines and operators that were overcrowding the diagram. Although there is a solution to group complex functions in a single, user-defined block, this is not practical for the operators that are created only once and never reused again. In Figure 11 we present an example of the mapping in the Altova MapForce.

4.2.2 *AnalytiX Mapping Manager*

AnalytiX Mapping Manager is a pre-ECTL general data mapping tool. It allows users to integrate source and target data schemas and then generate the specification of the ECTL process, i. e., ECTL job code, for a plethora of contemporary ECTL tools. Mappings and data transformation specifications are created in a tabular form that resembles Microsoft Excel spreadsheets. This makes the tool appealing to users, other than software developers, who are used to tabular data representations. The observed characteristics are as follows:

Distribution. Commercial. Distributed as a web application.

Recentness. Most recent version in 2016. Version: AnalytiX Mapping Manager v7.0

Domain: application. Data integration.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Tabular representation of mappings between source and target elements. Each row in a table represents a single mapping. Fine tuning of mappings is possible through configuration dialogs.

Language: expression. Textual syntax. Possibility to use built-in functions or to create own and store them in a global repository. Possibility to create reusable complex functions by writing down a pseudo-code that will be directly incorporated into the generated ECTL job code. The pseudo-code is not parsed by the tool but it is up the ECTL developer to ensure that it runs on a target ECTL tool.

Code generation and execution. Code is generated in a form of files readable by several contemporary ECTL tools. As AnalytiX Mapping Manager provides pre-ECTL mapping creation, once the mappings are created, ECTL job code is generated and run in the chosen environment.

Reuse: concepts. The reuse is possible at the level of expression language functions. User created functions can be saved in a global repository and reused in new projects. Automatic mapping is possible when data source and target elements have the same name. Also, the tool provides a meta-data repository that stores data sources and thus caching their configurations and elements for reuse. Mappings are also stored in a central repository and are kept under version control system which eases development and collaboration in a team.

Extensibility: technical spaces. No.

Extensibility: functionality. It is possible to extend code generators by providing new plug-ins.

The main advantages of AnalytiX Mapping Manager include easy learning curve, especially for people familiar with Microsoft Excel, mapping versioning system, and data analysis tools. Tabular syntax provides a good option to view mappings when a lot of mappings are created for a single project. It solves a problem of overcrowding that is present in tools with a graphical mapping syntax. Built-in mapping versioning system allows easier collaboration between team members and can be used to follow the evolution of a certain mapping. This would be of great help to narrow down possible reuse candidates if the tool would allow for a reuse of the whole mappings. As mappings are stored globally, data analysis tools can provide impact analysis of a data source element changes.

The main disadvantage of the tool is the lack of option to generate standalone data transformation adapters. Although it has its benefits, the tabular representation requires more effort to get an overview of all the mappings and might be somewhat non-intuitive for users that are not familiar with the tool.

4.2.3 *FME Desktop*

FME Desktop is an industry-ready commercial data integration and migration tool. It uses a simple but powerful mapping language to specify transformations between any two TSs. FME Desktop provides a combination of a graphical and configuration-based approach to mapping specification. This improves both readability and ease of use as the less experienced users can easily configure most of the things through the provided wizards. An example of an adapter specification in FME Desktop is given in Figure 12. The observed FME Desktop characteristics are as follows:

Distribution. Commercial. Distributed as a desktop and a web application.

Recentness. Most recent version in 2016. Version: FME Desktop Professional Edition.

Domain: application. Data integration and migration. Main focus on geo-spatial data and relational databases.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Mixture of graphical and configuration-based languages.

Language: expression. Configuration-based approach with the usage of templates and built-in expressions.

Code generation and execution. Code is executed inside the FME environment.

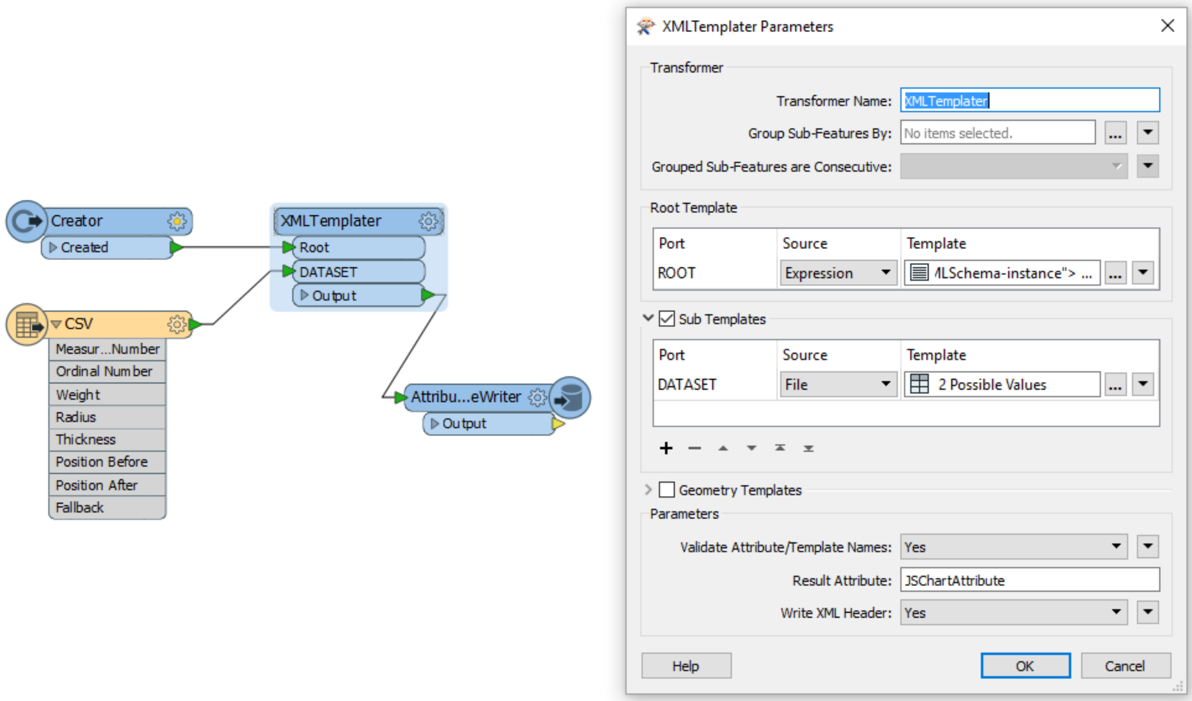
Reuse: concepts. Whole mappings can be reused. As transformations are created using one or more transformers, they may be grouped together thus creating custom transformers. Such transformers are stored in a global repository and may be reused between the projects.

Extensibility: technical spaces. Yes. Support for new TSs can be added and importers can be created based on existing importers or from scratch.

Extensibility: functionality. Almost every aspect of the software can be extended. Although the generators are not present in a way that they generate transformations in an arbitrary execution environment, new generators can be specified by the means of templates. Templates are in charge of creating custom output files.

The main advantage of the FME Desktop is its extensibility in almost every way. User can create new TS importers, new functions and new generator templates. Another major benefit is the ability to create custom transformation elements and thus provide both reusable transformations and more readable diagrams.

The main disadvantage of the solution is the lack of ability to generate external transformation adapters. In order to employ the created transformations in real-time scenarios, it is impractical or sometimes even impossible to use the FME tool for each transformation due to the limited computing resources often encountered in the industrial application. Although using the configuration-based approach helps reducing the number of items on the diagram, it hinders the ability to see the whole picture of the solution at a glance even more than it is the case with the tabular representation. Additionally, FME Desktop has a somewhat steeper learning curve compared to similar tools, mainly Altova MapForce. However, this is to be expected due to the extensibility and more complex nature of the constructs in the tool.



Transformer

Transformer Name: XMLTemplater

Group Sub-Features By: No items selected.

Grouped Sub-Features are Consecutive:

Root Template

Port	Source	Template
ROOT	Expression	ʌLSchema-instance"> ...

☑ Sub Templates

Port	Source	Template
DATASET	File	2 Possible Values

☐ Geometry Templates

Parameters

Validate Attribute/Template Names: Yes

Result Attribute: JSChartAttribute

Write XML Header: Yes

Figure 12: FME Desktop: CSV2XML example

4.2.4 *OPC Router*

OPC router is a tool that focuses mainly on device integration and integration between devices and information systems. It has several connectors developed for connecting Open Platform Communications (OPC)–enabled devices and databases of information systems. Custom transformations between source and target TSs are specified with a graphical syntax but the more complex operations are specified using C# programming language. For each mapping an execution trigger must be specified. Once the mappings and triggers are specified, code is generated in a form of an operation system service. Service is then run in the background and whenever the trigger condition is satisfied mappings are executed. The observed OPC Router characteristics are as follows:

Distribution. Commercial. Distributed as a standalone desktop application.

Recentness. Most recent version in 2016. Version: OPC Router v3.1.11.0.

Domain: application. Device integration.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical syntax comprising of nodes and lines.

Language: expression. Textual syntax. C# can be used for specifying transformation expressions.

Code generation and execution. Code is executed as an operating system service. Once the whole mapping is specified, executable code can be generated and run as a service. A trigger is specified for each mapping. The mapping is executed each time the trigger condition is satisfied.

Reuse: concepts. Mappings between TSs or custom scripts can be stored and reused. Project-level repository exists and can be used to store TS connector configurations, mappings, and user-defined functions i. e., scripts. Chain of mappings can be decomposed and stored as multiple mapping templates to allow easier reuse. Scripts representing user-defined functions can also be stored and reused.

Extensibility: technical spaces. Scripts can be used as a data source. As they are written in C#, any file format can be read and parsed.

Extensibility: functionality. Scripts can be used for the specification of user-defined functions. Custom data generators can also be written in C# as a separate script.

The main advantages of the tool are its advanced OPC and database connectors and the ability to specify triggers and mapping on a single diagram. In this way, the OPC router is well equipped for usage in industrial scenarios. The single diagram can be very useful for understanding the whole process and easier communication of the big picture.

The main disadvantage of the tool is that it requires a lot of manual programming in C# in order to support complex mappings. Also, although the reuse of whole mappings between TS or scripts is possible, parts of the mapping cannot be reused and thus automatic connection and reassembly of mappings from different use cases is not possible.

4.2.5 *Open Mapping Software*

In Open Mapping Software (OMS), all data structures being integrated are first mapped onto a common UML model that should represent semantics of the integrated data. The UML model must be created manually in the way to provide the insight in the data meaning, with no complexities that are inherently part of the data serialization formats. When several data sources are mapped to the same logical model, OMS can automatically generate and run transformations from any data source to any other, preserving all information they have in common, and guaranteeing consistency in round-trip transformations. The observed Open Mapping Software characteristics are as follows:

Distribution. Freeware. Distributed as a set of Eclipse Integrated Development Environment (IDE) plug-ins. Generator and data cleaning plug-ins do not come with the solution and must be bought separately.

Recentness. Most recent version in 2013. Version: Open Mapping Software v1.1.2.

Domain: application. Health system integration and transformation between health message formats.

Mapping approach. Indirect. A common meta-model is created from source and target TSs and then a user must map both source and target TS to this meta-model.

Language: mapping. Mapping is done in a series of configuration views in Eclipse.

Language: expression. XML Path Language (XPath) [170].

Code generation and execution. Transformations are executed in the Eclipse environment and provided by an appropriate plug-in. In order to generate an Eclipse-independent integration adapter, XSLT transformations are generated and then executed between source and target data. The XSLT generator is not free and must be bought separately.

Reuse: concepts. No.

Extensibility: technical spaces. No.

Extensibility: functionality. No.

The advantages of OMS lie in the captured meaning of the data that are being transformed. Once a common information model is created it is just a matter of mapping input and output schemas onto this model.

However, possessing such a model is also a disadvantage of the approach. For each use case, a new model must be created as it is not related to a combination of TSs but to the use case. Therefore, this resembles the manual development of adapters as it is costly and repetitive. Another issue is related to the user interface of the tool. OMS heavily relies on the Eclipse platform and a user must use multiple Eclipse views at the same time to specify a single transformation. This hinders the productivity as a user must focus on different aspects of the transformation at the same time instead of having all changes localized. Another disadvantage of the tool is the lack of support for many TSs and its inability to extend this easily.

As the tool does not support CSV TS that we used for our input, in order to test the functionality of the tool we have created an appropriate XML schema and put the CSV data in a corresponding XML document. XML schema comprised a root *Row* element with each of the CSV columns being its child elements.

4.2.6 *MetaDapper*

MetaDapper is a C# library that aims to mitigate much of the boilerplate code needed for the implementation of data convertors from one TS to another. In addition to the library, a MetaDapper Configurator may be used to create mappings between TSs almost entirely without writing C# code. The mappings are created through a series of configuration dialogs and forms, and are stored in an XML configuration file. Once the mapping configuration is finished, the XML document is executed by a C# engine and the data is transformed. The observed MetaDapper characteristics are as follows:

Distribution. Commercial. Distributed as a C# library and a standalone desktop application.

Recentness. Most recent version in 2015.

Domain: application. Data integration.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Configuration-based mapping through the MetaDapper Configurator. Mappings are specified through a number of dialogs and screen forms. The MetaDapper library may also be used in C# projects and thus the mappings may be specified in standard C# object-oriented, i. e., textual way.

Language: expression. C# language.

Code generation and execution. An XML document is generated which represents a mapping configuration. C# application then executes the transformation based on the generated XML document.

Reuse: concepts. User-defined functions may be specified and stored globally.

Extensibility: technical spaces. No.

Extensibility: functionality. C# functions may be defined and used in a process of data format mapping. New mapping language concepts and code generators cannot be added.

One of the main advantages of this solution is its direct inclusion in the C# projects. Therefore, MetaDapper functions can be used in a project without need to use external tools which is often the case. Another benefit of such a tool is localization of changes when the mapping configuration changes. In this case, only one XML configuration file is changed and the rest of the C# code remains the same.

However, configuration-based mapping specification is a main drawback of the MetaDapper Configuration tool. Once a mapping is created it is put in a tree view structure for easier handling and finding. Such a tree view in the MetaDapper does not contribute to the readability and user efficiency. All mappings are created as sub elements of the same tree and it is very hard to get a full picture of the solution as the user just gets the overview of which mappings are created but not between which elements they are created. The tool supports only XML, Relational Database Management System (RDBMS), Microsoft Excel, and Portable Document Format (PDF) TSs and it cannot be easily extended to support more of them.

4.2.7 *MuleSoft Anypoint Studio*

Anypoint Studio is a tool developed by MuleSoft with a goal to serve as a tooling support for their own ESB. It provides a set of graphical and textual languages that aim to ease the process of integrating different TSs and creating appropriate transformation adapters. Mapping specification is performed in two steps. First, a high level specification of the mapping process is created. In this view a source and target TS importers/exporters are specified with a high level transformation component between them. This way a user may create a chain of transformations. Transformation components are usually created to be used for a specific combination of TSs with the one used in this example being a generic message transformation component. Once the transformation process is specified, mappings between source and target element are created for each element in the source and target schemas. The CSV2XML mapping example created in this tool is presented in Figure 13. The observed MuleSoft Anypoint Studio characteristics are as follows:

Distribution. Commercial. Distributed as a standalone desktop application which is an adapted Eclipse IDE.

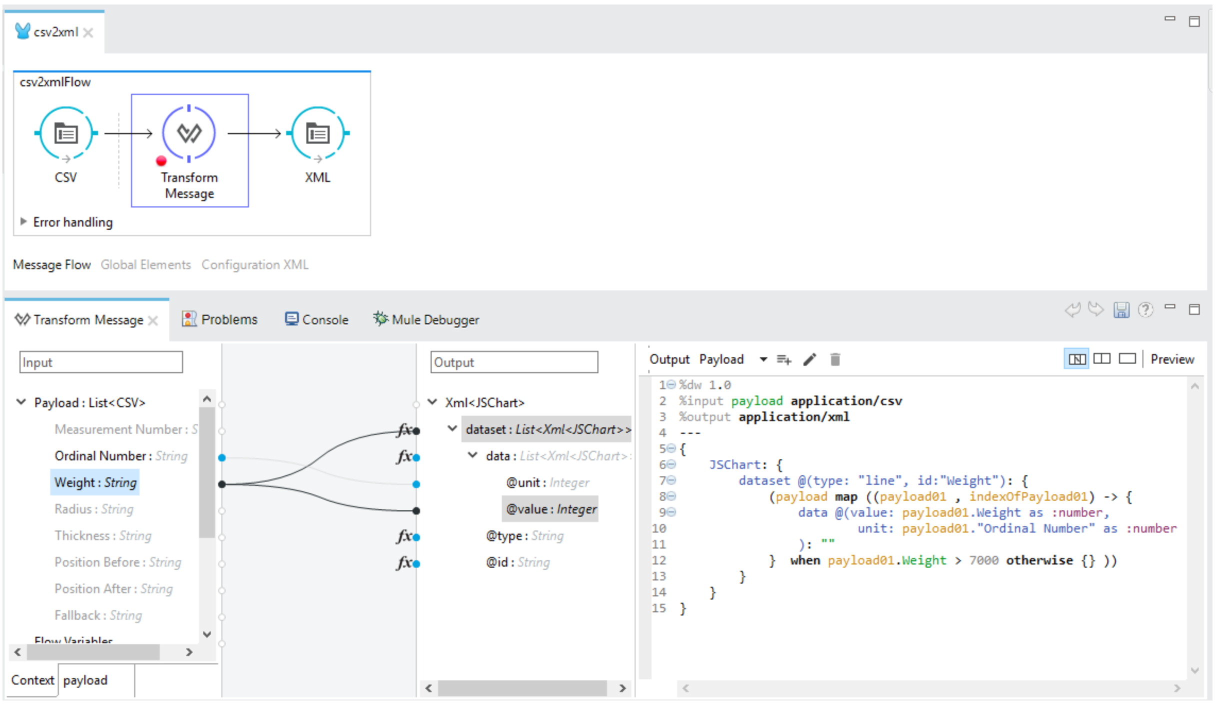
Recentness. Most recent version in 2016. Version: Anypoint Studio v6.0.3.

Domain: application. Data integration.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Both mapping process and mappings are specified using a graphical language.

Language: expression. Functions and other transformation expressions are specified in DataWeaver component that uses a custom textual language that resembles JSON.



```
%dw 1.0
%input payload application/csv
%output application/xml
---
{
    JSChart: {
        dataset @(type: "line", id:"Weight"): {
            (payload map ((payload01 , indexOfPayload01) -> {
                data @(value: payload01.Weight as :number,
                        unit: payload01."Ordinal Number" as :number
                ): ""
            }  when payload01.Weight > 7000 otherwise {} ))
        }
    }
}
```

Figure 13: MuleSoft Anypoint Studio: CSV2XML example

Code generation and execution. Code is generated and run on the Mule ESB, which is provided by the developer of the tool. It can be run inside the tool or as an external integration job.

Reuse: concepts. Custom-built connectors can be reused between projects to provide import/export capabilities for a certain TS.

Extensibility: technical spaces. New connectors can be manually created to provide permanent import/export of the technical space.

Extensibility: functionality. No.

The main advantage of the tool is its master–detail approach to the specification of mappings. It allows for the specification of transformation chains and this way more complex transformations can be decomposed and solved with several intermediary steps.

The main disadvantage of the tool is the lack of a proper reuse algorithm. Developed connectors, that are used to connect new TSs, can be reused between projects, but the mappings between elements in DataWeaver components cannot be reused at all.

4.2.8 *Vorto*

Vorto is an open-source project developed and backed by Bosch [1], which aims to provide a model-driven and technology-agnostic description and integration of devices in the field of Internet of Things. Each device is described in a form of an Information Model (IM) that represents a set of function blocks. Vorto focuses on the knowledge sharing and users are encouraged to store their IMs in an online repository. Vorto also focuses on code generation and these IMs can be transformed into platform-specific representations via one or more code generators. These platform-specific representations are used on specific integration platforms to allow communication with the modeled devices. This way, all the changes to physical devices are applied only at the IM level, allowing automatic propagation to all the different platform-

1 http://www.bosch.com/

specific representations via code generation. The main idea behind Vorto is to bring together device manufacturers that specify IMs for their devices, platform vendors that specify code generators for their platforms, and solution developers that need to integrate a specific device with their solution that is running on a specific platform. The observed Vorto characteristics are as follows:

Distribution. Open source. Distributed as a set of Eclipse plug-ins.

Recentness. Most recent version in 2016. Version: Vorto 0.4.0_M2

Domain: application. Device integration in the domain of Industry 4.0 and Internet of Things.

Mapping approach. Indirect. A common meta-model is created for the source and target devices.

Language: mapping. Rudimentary textual DSL exists for mapping specifications. However, currently mapping seems not to be the main focus of the tool, but it is a focus of the developer who uses the tool to get platform-specific models of the integrated devices to integrate them in a specific platform.

Language: expression. Rudimentary textual DSL exists for mapping specifications. Not many functions are currently supported.

Code generation and execution. Code generators can be easily created to traverse IMs and generate code for arbitrary execution platform. Code generators for many platforms can be already found in a global repository.

Reuse: concepts. Information models and its building blocks can be reused. Global repository of IMs and code generators exists.

Extensibility: technical spaces. Any TS can be modeled.

Extensibility: functionality. As the tool is open source, any aspect can be changed easily. Code generators can also be provided without changing the core code base but just by implementing predefined interfaces.

The main advantage of the solution is its technology-agnostic description of real world devices and the bottom-up approach to describing them. DSLs used to describe IMs are light, easy to learn, and concise enough which makes the specifications readable and easily reusable. Central repository of IMs and generators further contribute to the vision of shared knowledge and reuse. Finally, as Vorto is backed by Bosch and used in their industrial projects, we may expect for this solution to be improved further and to be maintained for a foreseeable future.

The main disadvantage of Vorto is its mapping language. Although Vorto states that one of its main strengths is integration of devices, integration adapters are created manually at the level of an integration platform. Only device interfaces are created automatically from IMs. A rudimentary IM mapping language that can be used to bridge these high-level abstractions of devices and these adapters can be later used in the process of code generation. However, IM mapping language is not yet fully specified, documented, and developed and thus transformations between devices cannot be specified in a transparent and easy to understand way.

4.3 SURVEY OF XML MAPPING TOOLS

In this section we present software solutions that are focused on the analysis, visualization, or design in the XML TS but also provide a possibility to load and map different TS. These tools use XML as a mediatory TS, or in another words as a generic structure all other TSs are mapped onto. After TS schema structures are converted to XML, standard XML languages for the specification of transformations can be used. Such languages are XQuery and XSLT. In this category of tools, we chose the ones that are still maintained and have an active community. The following tools will be surveyed in this section:

- Liquid XML Studio [126] and
- Stylus Studio [180].

4.3.1 *Liquid XML Studio*

Liquid XML Studio is a graphical tool for designing, analyzing, and transforming XML documents. We evaluated its Data Mapper component that provides a way to map different data sources to and from XML. It uses a purely graphical notation for the creation of mappings. The observed Liquid XML Studio characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application.

Recentness. Most recent version in 2016. Version: Liquid XML Studio 2016.

Domain: application. XML document design and analysis.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical mapping language.

Language: expression. Graphical expression language. Functions are represented as graphical blocks with various number of inputs and outputs. Built-in functions can be grouped together to form user-defined functions. The grouping mechanism mostly serves the purpose of making less crowded diagrams and allowing the reuse of user-defined functions.

Code generation and execution. Code can be generated in Java, C#, and XSLT programming languages. The latter is only applicable when the transformation is specified between two XML documents.

Reuse: concepts. User-defined functions can be reused. Although there is no global repository of user-defined functions, they are stored in separate files at the level of a project. Hence, a file can be manually copied into a new project so a function can be shared and reused.

Extensibility: technical spaces. No.

Extensibility: functionality. No.

The main advantage of Liquid XML Studio is its ability to generate C# and Java adapters for transforming the data. This is different from the most of XML mapping tools as they often provide just the generation of standard XSLT transformations. In addition to external execution of the adapters, they can be run within the tool and the transformations can be easily debugged. Another positive characteristic of the Liquid XML Studio tool is the possibility to create custom, user-defined functions that are in fact groups of other previously defined or built-in functions. This way, a user is able to create more concise, cleaner, and more readable diagrams. If it were not for the grouping, all functions would need to be specified in a graphical way and it would lead to serious diagram overcrowding.

The main disadvantage of Liquid XML Studio is the lack of documentation and tutorials. Another drawback of the tool is the lack of a mechanism to extend the tool. The tool supports only a limited number TSs, code generators, and built-in functions which are specific to the XML TS.

4.3.2 *Stylus Studio*

Stylus Studio is an XML IDE comprising various XML tools in one suite and offering Java and .NET components to design and develop applications for data integration. It focuses on the XML TS and uses the XQuery language for the specification of XML mappings. In order to integrate arbitrary two TSs, these spaces need to be represented in a form of an XML document. Custom converters for importing an arbitrary TS into the XML TS, can be created using Java or C# programming languages. Such converters have to provide bi-directional programmatic access to any TS which means that they must allow for the data to be loaded and written to the TS they are created for. In Stylus Studio, such components are called DataDirect XML Converters. One-time data conversions from a limited number of TS can be done in the tool without the need to write external converters. Once data formats are transformed to corresponding XML schemas, the mappings between two XML schemas are created using XQuery language. Stylus Studio provides both textual and graphical syntaxes for writing the transformation code. These syntaxes are a part of the XQuery Mapper component. The observed Stylus Studio characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application.

Recentness. Most recent version in 2016. Version: Stylus Studio X16 XML Enterprise Suite 64-bit.

Domain: application. XML schema integration.

Mapping approach. Indirect. Every TS must be first mapped to the XML TS.

Language: mapping. Graphical and textual syntax in a form of XQuery statements.

Language: expression. Graphical and textual syntax in a form of XQuery statements.

Code generation and execution. Code is executed in the tool. It is possible to generate Java and C# adapters from the XQuery specification.

Reuse: concepts. User-defined added in a form of external Java functions can be reused between projects.

Extensibility: technical spaces. New technical spaces are introduced by converting them to XML TS. Custom converters can be created using Java or C#.

Extensibility: functionality. New functions can be created externally in the Java programming language and then used in the tool.

The main advantage of the tool is very lightweight graphical syntax for defining XQuery mappings that can be easily read and understood. Also, the fact that the tool focuses on the XML TS and maps everything to it allows for the XML-specific tools, languages, and functions to be used.

However, we feel that the graphical representation is given just for visualization and documentation purposes as not all details of XQuery can be specified just by using this representation. Other disadvantages include the lack of up-to-date documentation and overall look and feel of the whole application. Although it may be considered as the advantage, basing the whole data integration approach on the XML TS can lead to some limitations. All other TSs must be converted to an appropriate XML representation that can be limiting in some ways and may hinder some TS specific relationships and elements to be represented properly.

4.4 SURVEY OF ECTL TOOLS

Extraction, Cleaning, Transforming, Loading (ECTL) or, more often, Extraction, Transformation, Loading (ETL) processes comprise complex data workflows, which are responsible for the maintenance of Data Warehouses. Data Warehouses represent a collection of data gathered from different sources, aimed to provide data analysis, decision-support, and other business intelligence tasks in an organization. Unlike general mapping tools, presented in Section 4.2, ECTL tools are not focused solely on the data mapping task, they usually offer the possibility to create the whole ECTL process which includes specification of paralelization, quality assurance tasks, tests, versioning systems, etc.

As the ECTL process is a critical task for a company that in the end results with a system that provides a decision-support for the upper management, the ECTL tool market has become a multi-million dollar industry and thus many comparative studies have been done over the last few years [155]. Such studies often compare performance and functional coverage of an ECTL tool. However we are interested in the characteristics presented in Section 4.1 and as far as we know no other survey has compared these. Although there are many ECTL tools available, we have chosen the ones that we could obtain a copy, trial or full, and install afterward. The following ECTL tools will be presented in this chapter:

- Adeptia ETL Tool [1];
- Clover ETL Tool [156];
- Informatica PowerCenter [91];
- Karma [84];
- Microsoft SQL Server Integration Services (SSIS) [142];
- OpenRefine [187];
- Oracle Data Integrator [153]; and
- Talend Studio [31].

4.4.1 *Adeptia Integration Suite*

Adeptia Integration Suite focuses on three integration domains: business-to-business integration, application integration, and data integration that includes the specification of the whole ECTL process. The tool consists of three distinct components. It has a web-based Design Studio that provides wizard-driven, graphical ability to specify data rules, validations, and mappings. Design studio includes the Data Mapper component for the specification of mappings which has a built-in preview capability to test the mappings. It is worth noting that the Data Mapping component runs as an external Java application and not in the web environment which may cause some performance and user experience issues. The second component is Central Repository where all the rules and mappings are saved. The third component is the Run-time Execution Engine where the mapping rules and data flow transactions are executed on incoming data files and messages. The observed Adeptia Integration Suite characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application.

Recentness. Most recent version in 2016. Version: Adeptia Integration Suite 6.4.

Domain: application. ECTL process specification.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical language. Limited to many-to-one mappings as each mapping may have only one target element and each target element can be produced in only one mapping. Multiple inputs for a mapping can be combined only with a limited number of predefined functions.

Language: expression. Mixture of graphical representation of functions and textual specification of properties in Xpath.

Code generation and execution. Code is executed as an operating system service by the Adeptia Integration Suite.

Reuse: concepts. Everything that is specified, such as mappings, triggers, and schema definitions are stored in a global repository and can be reused. However, there is no automatic adaptation of mappings when an input schema is changed. In that case, a new mapping must be specified from scratch. At least, the reuse algorithm allows combining different elements together, such as changing triggers on the fly without the need to recreate the whole specifications.

Extensibility: technical spaces. No.

Extensibility: functionality. A custom method can be created as a Java class and then used in the data mapping process.

The main advantage of the tool is its modularity in creating the data transformation solutions. All components, such as data sources and targets, trigger specifications, and mapping specifications are stored in a central repository and can be reused as individual components in new scenarios.

The main disadvantage of the tool is its lack of a good extension mechanism. As far as we know, it is not possible to introduce new data interfaces to the tool which greatly hinders the possibility to customize the solution based on customer needs. Another disadvantage of Adeptia Integration Suite is the lack of automatic adaptation of mappings to a new, slightly changed, integration scenario. Although it is one of the rare tools that stores the mappings in a global repository and allows their reuse, mappings must be created from scratch if the source or target data schemas change even slightly. There is no (semi-)automatic way to provide a list of possible candidate expressions based on previous solutions. However, although the automatic adaptation is not supported, this modularity of reuse offers more flexibility than the reuse of entire components.

4.4.2 *CloverETL*

CloverETL is an ECTL tool that provides data integration, data migration, specification of data warehousing logic, reporting, and data quality assurance. It uses a mixture of graphical and textual syntax for the specification of ECTL processes, i. e., ECTL jobs. Once a job is specified, it can be run within the tool. The observed CloverETL characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application which is an adapted Eclipse IDE. It is based on the open-source engine which is available for usage in Java projects but without the accompanying graphical user interface.

Recentness. Most recent version in 2016. Version: CloverETL Designer 4.3.0.

Domain: application. ECTL process specification.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical language. Two views are defined with different degree of granularity. The master view, with the lower degree of granularity, is used to specify the transformation logic between different components of the system at the high level of abstraction. The detail view with a higher degree of granularity, is used for the specification of mappings between concrete elements of the mapped TS.

Language: expression. Graphical and textual. High level operators, such as filters and mappers, are created graphically on the master view. Expressions that specify concrete values, e. g., filters and constants, are specified in a textual way. A plethora of languages are supported including a custom, built-in language that is specifically tailored for CloverETL, and also Java, Python, and JavaScript.

Code generation and execution. All transformations are executed internally in the CloverETL tool.

Reuse: concepts. TS specifications can be reused. They are stored in a form of meta-data specifications where the schema elements are extracted and used in the specification of transformation logic as a source or target elements. Also, different functional blocks may be grouped together with an interface clearly defined in order to form a new, reusable, function block. All specifications are stored in a global repository.

Extensibility: technical spaces. Custom Java readers and writers may be specified, allowing a user to create an importer and an exporter for an arbitrary TS.

Extensibility: functionality. Custom Java transformers may be specified. This allows a user to extend the core functionality of the tool by creating custom functions to be used in mapping specifications.

The main advantage of the tool is its extensibility and number of supported languages in which the ECTL logic can be implemented. Custom Java components provide the extension of the expression language and TS importers and exporters. This also greatly improves reuse of elements as the custom function blocks can be created in Java. Additionally, mapping expressions do not have to be specified in the built-in language or Java. CloverETL provides mechanisms that enable usage of JavaScript, Python, and other programming languages for the specification of expressions. This way, existing and legacy codebase of a company can be greatly reused.

The main disadvantage of the tool is its inherent complexity. By using custom code components and complex built-in components whose purpose or structure is not immediately obvious when looking at the graphical process specification, it may be hard for a user to comprehend the logic behind specifications. User has to have a deep knowledge of internal structure of functions and transformers in order to really comprehend what is happening and why.

4.4.3 *Informatica PowerCenter*

Informatica PowerCenter is a tool suite that can be used for the development of data integration solutions. It provides mechanisms for data analysis, data quality assurance, and data integration. PowerCenter comprises two applications, Developer and Administrator tools, and several services and repositories that are usually installed on a single machine and represent central point of knowledge. Developer and Administrator tools can be used by multiple users on their own machines to create integration solutions and administer them respectively. In addition to the creation of mapping, Developer tool allows for the specification of ECTL workflows that can be run to perform a sequence of events, tasks, and decisions specific to business process requirements. The observed CloverETL characteristics are as follows:

Distribution. Commercial. Distributed as desktop and web applications. Freeware option when using PowerCenter Express. Desktop application is realized as an adapted Eclipse IDE.

Recentness. Most recent version in 2016. Version: PowerCenter Express v9.6.1.

Domain: application. Data integration.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Graphical language.

Language: expression. Graphical and textual specification of expressions. Data transformation scripts can be specified graphically in a form of nested tree elements where each element is an operation. Textual syntax includes both built-in expression language that can be used to specify conditions in filters, and Java language that can be used to specify custom transformation components.

Code generation and execution. The tool provides both internal execution of ECTL tasks with debugging option and external execution in one of the supported integration platforms.

Reuse: concepts. Everything that is created in the Developer tool, e. g., mappings, TS specifications, functions, and custom elements, is stored in a common repository which is serialized into a database. Stored elements can be later reused only as a whole, for the same inputs and outputs.

Extensibility: technical spaces. A custom Java data transformer can be created in order to import and export data in a TS. The import components transform data from a generic data source and load it as a single data element in a system. Therefore, further parsing and cleaning of the data is done in the tool itself, using a set of predefined functions. Export to the unsupported TS is done via custom Java transformer that outputs a single file with the data formatted according to the rules of the TS.

Extensibility: functionality. A custom Java transformer can be created and used together with built-in functions thus extending the expression language.

The main advantage of the tool is its user friendliness and approach to users with little or no software development experience. In addition to a graphical syntax for the specification of ECTL workflows and data integration solutions, there is also a tree-based graphical syntax for the creation of complex, procedural transformation scripts. This way, a user with no software development background can specify functions and procedures in an iterative way, testing and debugging the graphical specification in the tool.

The main disadvantage is the quality of the extension mechanism. When adding a new TS, whether as an input or an output, custom Java data transformation must be created which can be somewhat counter-intuitive. For example, in our use case, we needed to output a result in an XML document. As there is no XML exporter built-in, a custom way to produce the data is needed. In most of the similar tools, the solution would be to create a data exporter that takes field values as parameters, formats them according to an XML schema and stores them in a file. In PowerCenter, a transformation must be created that receives inputs from other transformations and outputs a single string element whose content has been manually formatted using strings and placeholders for values. Afterward, this string element is serialized by a single-input flat file connector that just writes this content to a file. This way, no syntax and semantic analysis on the XML document can be done by the PowerCenter transformation components as they view the XML content just as regular text content.

4.4.4 *Karma*

Karma is an information integration tool that takes an ontological approach to integration of different data sources and represents data schemas and instances using Resource Description Framework (RDF). The integration process is centered on a common ontology that describes knowledge represented by both source and target data files. Mappings are specified by connecting instance data from data files to appropriate ontology classes that are manually specified in advance. Karma provides a self-adapting and learning mechanism that is able to suggest instance-to-class mappings based on the acquired knowledge from previously defined mappings. The main purpose of the Karma tool is to facilitate data analysis based on the ontology that describes the knowledge in the domain. The observed Karma characteristics are as follows:

Distribution. Open source. Distributed as a web application.

Recentness. Most recent version in 2016.

Domain: application. Data analysis. Ontological analysis of data.

Mapping approach. Indirect. Source and target technical spaces must be first mapped to a common ontology which represents a single point of integration.

Language: mapping. Mappings between data from TS and a central ontology are created using a mixture of a graphical syntax and configuration dialogs. Data is represented in a tabular way and each column is linked to an element of the ontology.

Language: expression. Data can be transformed both by using built-in functions through a series of menus or by using Python programming language to write transformation scripts. Python scripts can be of an arbitrary complexity thus providing highly expressive tool to transform data.

Code generation and execution. RDF files are generated and used for ontological data analysis.

Reuse: concepts. Mappings between data instances and ontologies are stored in the system repository. System is trained based on the stored mappings and it is able to offer suggestions for data-to-ontology mappings based on data element name. The system then automatically adapts the chosen suggestion to fit the new scenario.

Extensibility: technical spaces. As the tool is open source, any technical space can be added. However, it is our opinion that it would be hard to add a new TS as the authors of the tool did not offer any interfaces or out-of-the-box solution for extending it.

Extensibility: functionality. As the tool is open source, any function or code generator can be added. However, just like the extension of technical spaces it requires good knowledge of code as the tool architecture is not modular.

The main advantage is the ability to analyze data from the ontological viewpoint. Karma generates RDF files that can be imported to ontology analysis tools for further analysis. Another advantage is the usage of Python to provide advanced cleaning and transformation functionalities thus allowing data to be properly processed before it is mapped to an ontology.

The main disadvantages of the solution are the lack of real generators that can produce output files based on the input and its non-modular architecture. Furthermore, as it can be seen on the software GitHub [2] page, the most intensive development of the tool was in 2014. Since then, number of actual improvements and code contributions has diminished drastically.

4.4.5 *Microsoft SQL Server Integration Services*

Microsoft SQL Server Integration Services (SSIS), more precise their SQL Server Data Tools, are intended to be used for data integration and specification of integration flows and packages. The whole tool suite is built around the Microsoft SQL Server (MSSQL) [3] database and it uses the database as a mediator in the integration of different TSs. However, there are ways to create integration packages without the use of the SQL Server database. The observed SSIS characteristics are as follows:

Distribution. Freeware. Distributed as a desktop application. Although it requires MSSQL to have all functionality enabled, SSIS may be used without it which makes it a freeware solution.

Recentness. Most recent version in 2016. Version: Microsoft SQL Server 2016 Integration Services.

Domain: application. ECTL process specification.

Mapping approach. Indirect. RDBMS TS is used as a mediatory TS.

Language: mapping. Graphical and tabular. High-level mappings between components are done using a graphical syntax. Detailed mappings are performed in a mixed graphical and tabular syntax inside high-level mappings.

Language: expression. C# and Structured Query Language (SQL) are used for the specification of transformation expressions.

Code generation and execution. Externally as a C# application or internally within the SSIS tool.

Reuse: concepts. Defined transformation components can be reused for different input and outputs. However, there is no way to reuse single mappings, parts of the transformation component, on their own.

Extensibility: technical spaces. Yes, by the means of C# code.

Extensibility: functionality. Yes, custom components can be developed in C#.

The main advantages of this tool are its integration with other Microsoft products and its reliance on the database as a mediatory component. First advantage is important for the

2 https://github.com/usc-isi-i2/Web-Karma
3 https://www.microsoft.com/en-us/cloud-platform/sql-server

users that are dealing mainly with Microsoft technologies, databases, data warehouses, and programming languages. For them, this tool is the best option that integrates very well with other tools from the same vendor. SSIS provides easy-to-learn graphical syntax and the developers that need to specify transformation expressions are usually familiar with the C# and SQL languages. Using a database as a mediatory TS allows usage of SQL as a means to easily get the data to be written to the output.

However, although we have considered the mediatory database as an advantage, this is also the main disadvantage of the SSIS tool. SSIS lacks the options to transform any two TSs without using the database. In our example, we were not able to transform CSV data to XML data directly. We had to transform CSV to the database, and afterward use SQL statements enriched with XML expressions in order to produce XML result. This leads to not so obvious mappings and requires too much effort. It is needed to map a source TS to the database and to map the database to a target TS instead of mapping TSs directly.

4.4.6 *OpenRefine*

OpenRefine, previously known as GoogleRefine, is a tool for exploring, transforming, augmenting, and reconciling data with other data sources. It has powerful mechanisms for exploring data with various filters and aggregate built-in functions. Transformations are created using a custom-built expression language that can use both data and meta-data characteristics. Augmentation and reconciliation mechanisms provide a way to dynamically add or refine data from other sources to the data under study. Custom template engine allows generation of data in any format. The observed OpenRefine characteristics are as follows:

Distribution. Open source. Distributed as a web application.

Recentness. Most recent version in 2016.

Domain: application. ECTL process and data analysis.

Mapping approach. Direct. No mediatory TS is used.

Language: mapping. Configuration based mapping language.

Language: expression. GoogleRefine Expression Language (GREL), a built-in textual expression language. Jython and Clojure can also be used as expression languages.

Code generation and execution. Generators are specified by using custom template engine. Once the data is cleaned and transformed, templates are used to generate data in the desired target format. The solution does not generate adapters or any executable code.

Reuse: concepts. All transformations in a project are stored as a single JSON file. They can be reused in new projects but all adaptations to transformations must be done manually. Any NoSQL database that stores JSON documents could be easily used as a repository.

Extensibility: technical spaces. OpenRefine is an open-source project and is built in a modular way. Therefore, new importers for TSs can be added easily.

Extensibility: functionality. New language constructs can be added easily as GREL is also an open source expression language. Arbitrary code constructs can also be written in Jython and Clojure. New generators are added by creating new templates without the need to modify the core code base.

The main advantage of the solution is its completeness, functional richness, and shallow learning curve. It has a powerful GREL expression language that provides a user with general purpose constructs such as if statements and standard string operations, and also with some constructs specific to the domain of data ECTL processes. Both Jython and Clojure can also be used as expression languages. OpenRefine is also well documented and easily extendible with custom functions.

The main disadvantages of the tool, that are related to our use case, are the implicit nature of transformation specifications and the lack of adapter generators that can generate executable adapters which transform data in real time. Transformations between TS elements are manually created one by one through a set of configuration dialogs. After each transformation, transformed data is shown and it presents the base for the next transformation and so on. Although all transformations are stored in a single JSON document, the overview of the transformations is hard to get. Also, data transformations are done manually and there is no way to transform data in real time as they are gathered.

4.4.7 *Oracle Data Integrator*

Oracle Data Integrator (ODI) is a data integration platform that heavily utilizes other Oracle products in the development of data integration solutions. ODI uses Oracle database as a mediatory space to perform transformations. Every source and target TS are mapped to a separate set of database tables and the transformations are specified between these separate sets of database tables using SQL. The observed ODI characteristics are as follows:

Distribution. Commercial. Distributed as a desktop application.

Recentness. Most recent version in 2016. Version: Oracle Data Integrator v12.1.3.

Domain: application. ECTL process specification.

Mapping approach. Indirect. RDBMS TS is used as a mediatory TS.

Language: mapping. Graphical mapping language.

Language: expression. Graphical function symbols with textual parts written in SQL and Procedural Language/Structured Query Language (PL/SQL) are used to specify data transformations. PL/SQL procedures and functions allow fine grained control over the execution of the transformations.

Code generation and execution. Transformations are executed inside the ODI tool.

Reuse: concepts. Connectors to different TS can be reused as well as the schema specifications that represent the structure of the connected TS. User-defined functions can be reused as they are stored in the Oracle database of the ODI solution. Mappings can be reused by defining the transformation logic and providing special input and output elements that are later bound to a concrete input and output data.

Extensibility: technical spaces. New Java Database Connectivity (JDBC) drivers can be developed to provide connectivity to a new TS.

Extensibility: functionality. Custom PL/SQL procedures can be developed and used while specifying the transformations between TS.

The main advantage of the tool is its reliance on JDBC technology for importing and exporting data of a TS. JDBC is a well-defined and widely used mechanism for connecting to various sources. If a new TS needs to be processed, a developer needs to create a new JDBC driver which is not made for ODI exclusively, but is a standard way of accessing data sources in Java programs. This makes the development and reuse of such a connector easier than it is the case with custom API commonly used by other ECTL tools. It even allows reuse of the connector outside of the tool. Further, the advantage of ODI is in the fact that it seamlessly integrates with other Oracle products, making it the good choice for companies that have Oracle product landscape.

The main disadvantage of the Oracle Data Integrator is its complexity and steep learning curve. When specifying the ECTL process, a set of configuration steps must be performed which is sometimes counter-intuitive and does not intuitively lead a developer to a solution like it is the case with other surveyed tools. Many different dialogs and views need to be consulted in order to specify a single integration step. Therefore, sometimes it is hard to see the overview of the whole transformation process at a glance.

4.4.8 *Talend Studio*

Talend Studio provides means to build, maintain, and deploy ECTL processes. It heavily relies on a graphical syntax for the creation of data integration jobs. A project level repository allows for multiple elements to be reused and in this way to speed up the development of ECTL jobs, although just in a single project. Commercial version further provides code versioning, better collaboration between team members, and easier importing of meta-data and job description from other contemporary ECTL tools. The free version (Talend Open Studio) is an open-source tool which makes it a good candidate for adaptation and it can be easily extended. The observed Talend Studio characteristics are as follows:

Distribution. Commercial. Distributed as an standalone desktop application which is an adapted Eclipse IDE. Open-source version (Talend Open Studio) is also available.

Recentness. Most recent version in 2016. Version: Talend Studio v6.1.1.

Domain: application. ECTL process specification.

Mapping approach. Direct. No mediatory TS is used.

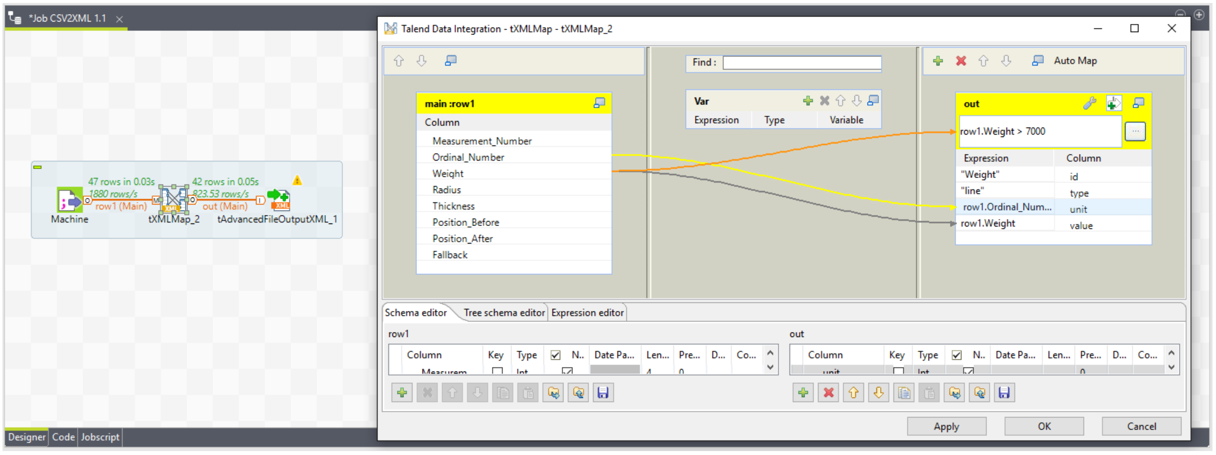
Language: mapping. Graphical language that is split into two parts. Graphical overview in the job creation window, where various TSs are connected together and processed or transformed by mapping operators. Such an overview allows for the creation of transformation chains and also getting a big picture of the ECTL process (left-hand side of Figure 14). For each mapping operator, a detailed mapping view is used to specify more detailed mappings between TS elements (right-hand side of Figure 14).

Language: expression. Java programming language.

Code generation and execution. Java adapters are generated and can be executed within or outside the tool.

Reuse: concepts. Data importers, i. e., data source and target meta-data specifications, and functions can be reused. It is possible to reuse the whole mapping just by choosing different data sources and data sinks, however individual element mappings cannot be reused. Each ECTL project is a repository that provides reuse of elements between different data integration specifications (jobs) inside a project.

Extensibility: technical spaces. It is possible to easily add new TS importers based on existing flat file configurations and regular expressions.



main :row1
Column
Measurement_Number
Ordinal_Number
Weight
Radius
Thickness
Position_Before
Position_After
Fallback

out: row1.Weight > 7000

Expression	Column
"Weight"	id
"line"	type
row1.Ordinal_Num...	unit
row1.Weight	value

Figure 14: Talend Studio CSV2XML example

Extensibility: functionality. It is possible to add new generators and expression language functions as the tool allows for the specification of Java functions and modules. Although it is possible in the free version of the tool, it is not easy to extend the graphical mapping language as the code base of the tool needs to be changed.

The main advantage of the tool is its functional completeness and shallow learning curve. Of all surveyed solutions, we feel that this tool has the most complete function and tool set. It is well documented and has a large community of users, mostly thanks to its open-source version. The open-source version is also easily extendible which makes the Talend Open Studio a very good foundation for custom tool development.

The main disadvantage of the tool is the lack of options to reuse individual element mappings in addition to reusing complete job specifications. A user is able to specify joblets, partial jobs which can have different data sources and data sinks. However, the whole mapping specifications must be specified inside a joblet and only as the whole may be reused. This is however only useful if new data sources and data sinks are subset or equivalent with the original ones. In Figure 14 we present our example implemented in Talend Open Studio.

4.5 SURVEY RESULTS

In this section we disseminate and compare the results of the survey. We also present the conclusions about each of the characteristics that we observed. The overview of the surveyed characteristic is split into two tables as to better fit the pages. In Table 2, we present the characteristics of the general mapping tools, while in Table 3 we present the characteristics of XML and ECTL tools.

4.5.1 *Software license, distribution, and recentness*

Although not of the direct importance, software license, distribution, and recentness can all give a context to other characteristic and conclusions.

All the tools that we have surveyed had one or more of the following licenses: (i) **proprietary/commercial license**, (ii) **freeware license**, which makes a tool free to use but does not provide its source code, and (iii) **open-source license**, which makes both the tool and its source code as free to use. Most of the tools, 72% (13/18), have a commercial license. This is not surprising given the fact that the tasks such as data integration, analysis, and specification require advanced algorithms and tool support to function properly. Also, these tasks are necessary in day-to-day operations of a company and also as an enabler of decision support systems used by company management. With a current increase in amounts of produced data and current trends such is the Big Data mining and analysis, these integration tools will only become more important as there is more and more data to process. 11% (2/18) of the surveyed tools have only a freeware license, while 17% (3/18) have an open-source license. It is worth mentioning that three of the commercial tools, Clover ETL, Informatica PowerCenter and Talend Studio, has also a freeware or open-source editions available for download in addition to their commercial edition. Nevertheless, we have chosen to perform our survey on the trial versions of the commercial editions as free versions were too limited with their functionalities.

All of the surveyed tools were offered as desktop solutions, web solutions, or in both forms. 67% (12/18) of the tools are distributed as desktop applications including the tools that are distributed as Eclipse IDE plug-ins and code libraries. 22% (4/18) of the tools are distributed as web applications, while only 11% (2/18), Informatica PowerCenter and FME Desktop, have both desktop and web installations available for download. It is interesting to note that none of

Tool name	License and Distribution	Application Domain	Mapping Approach	Lang.: mapping	Lang.: expression	Reuse: concepts	Code execution	Ext.: TS	Ext.: func.
Altova MapForce	Commercial, desktop	Data integration	Direct	Graphical	Graphical	Functions	All	Partial	All
AnalytiX Mapping Manager	Commercial, web	Data integration	Direct	Tabular and Config.	Textual	Functions	External	No	Code generation
FME Desktop	Commercial, desktop and web	Data integration	Direct	Graphical	Config.	Mappings	Internal	Yes	All
OPC Router	Commercial, desktop	Device integration	Direct	Graphical	Textual	Functions, Mappings	External	Yes	All
Open Mapping Software	Freeware, Eclipse plug-in	Health systems integration	Indirect	Config.	Textual	None	All	No	None
Meta Dapper	Commercial, desktop and library	Data integration	Direct	Config. or Textual	Textual	Functions	External	No	Expression language
MuleSoft Anypoint Studio	Commercial, desktop	Data integration	Direct	Graphical	Textual	TS spec.	All	Yes	All
Vorto	Open source, Eclipse plug-in	Device integration	Indirect	Textual	Textual	TS spec.	External	Yes	All

Table 2: Survey results: General mapping tools

Tool name	**License and Distribution**	**Application Domain**	**Mapping Approach**	**Lang.: mapping**	**Lang.: expression**	**Reuse: concepts**	**Code execution**	**Ext.: TS**	**Ext.: func.**
Liquid XML Studio	Commercial, desktop	XML schema integration	Direct	Graphical	Graphical	Functions	All	No	None
Stylus Studio	Commercial, desktop	XML schema integration	Indirect	Graphical and Textual	Graphical and Textual	Functions	All	Yes	Expression language
Adeptia Integration Suite	Commercial, web	ECTL process specification	Direct	Graphical	Graphical and Textual	TS spec., Mappings	Internal	No	Expression language
Clover ETL	Comm. and O.S., desktop	ECTL process specification	Direct	Graphical	Graphical	Functions, TS spec.	Internal	Yes	All
Informatica PowerCenter	Comm. and Free., desktop and web	ECTL process specification	Direct	Graphical	Graphical and Textual	All	All	Yes	Expression language
Karma	Open source, web	Data analysis	Indirect	Graphical or Config.	Config. and Textual	Mappings	None	Partial	All
Microsoft SSIS	Freeware, desktop	ECTL process specification	Indirect	Graphical and Tabular	Graphical and Textual	All	All	Yes	All
OpenRefine	Open source, web	Data analysis	Direct	Config.	Textual	Mappings	Internal	Yes	All
Oracle Data Integrator	Commercial, desktop	ECTL process specification	Indirect	Graphical	Graphical and Textual	All	Internal	Yes	Expression language
Talend Studio	Comm. and O.S., desktop	ECTL process specification	Direct	Graphical	Textual	All	All	Yes	All

Table 3: Survey results: XML mapping tools and ECTL tools

the companies publicly offered or advertised a software as a service (SaaS) way of using their tools. This may be justified by the data security concerns in the field of data integration and analysis. Therefore, all the tools are offered either as desktop application or web applications that are run exclusively at the customer premises.

It is also worth noting that one of the popular ways to distribute the surveyed systems was through the Eclipse IDE. As the Eclipse platform is stable, open source, widely adopted, and easy to extend and adapt, it represents an easy way of implementing necessary logic without the need to implement generic editor support. Informatica PowerCenter and Talend Studio are distributed as fully adapted Eclipse IDEs, while Vorto, Clover ETL Designer (a part of Clover ETL) and Open Mapping Software are distributed as a set of Eclipse plug-ins that a user can include to the existing Eclipse instance along with existing plug-ins.

As one of the criteria for choosing a tool for this survey was that it is maintained and used in real-world use cases, most of the tools have had a major release within the last year. The only exception is the Open mapping software which had its last major release in 2013. The tool is still functional and maintained, however it only had some minor releases that included bug fixes in the past year.

4.5.2 *Application domain*

We identified application domains of surveyed tools by relying on the official documentation and on our opinion after we implemented test example. Official documentation was first used to assign a tool to one of our main three categories: (i) **general mapping tools**, (ii) **XML mapping tools**, and (iii) **ECTL tools**.

General mapping tools, 44% (8/18), are usually declared as data or device integration tools. They provide means for creating mappings between source and target data or devices. OPC Router and Vorto are device integration tools as their functions and approach to integration are tailored to provide easy integration in the domain of IIoT and I4.0. Application domain of all other tools in this category can be identified as a general term of data integration. One exception of this general term is the Open Mapping Software which focuses on health data and protocols and falls short when integrating other kinds of data.

XML mapping tools, 11% (2/18), are declared to be XML schema integration tool. Although they provide means to integrate other TSs, this is usually done through a mediatory TS, in this case XML TS. Therefore, these tools favor XML TS over other TSs.

44% (8/18) of the tools belong to the third category: ECTL tools. While these tools are declared as ECTL process specification tools where a user is able to define the whole ECTL process, this is not completely true for Karma and OpenRefine. These tools may be considered as tools just for data analysis. Although these two tools support specification of the ECTL process, it is done implicitly with a lot of standard quality assurance, code generation and process automation functions missing.

4.5.3 *Mapping approach*

While surveying the tools and we have identified two approaches to the creation of mappings between source and target TSs: (i) **direct** and (ii) **indirect**.

67% (12/18) of the tools implement the direct mapping approach. In the direct approach to mapping, users create mapping elements between elements of the source and target TSs. This way, a user only sees the two TSs that are being mapped while an internal TS of the tool is hidden from the user. The internal TS of the tool is used just for the uniform visualization of imported technical spaces and for easier execution of the mappings by the tool. In our opinion, direct mapping allows a user to be more focused on the task and to handle source and target structures without losing focus. On the other hand, making generators for this kind of a solution is a complicated job as the generator has to be able to read the data directly from the source TS and create data in the format supported by the target TS while executing the mapping rules defined in the tool.

Indirect approach to mapping implies using a mediatory TS in the process of creating correspondences between source and target TSs. Users are required to first map the source and target TS to the mediatory TS and then create mappings between two structures in the mediatory TS. Sometimes, such a structure limits the overall expressiveness as the source and target can lose some of their semantics. The issue arises if the mediatory TS is semantically poor, it can be difficult to map all original source and target TS elements and their relationships. However, a mapping language used in an indirect approach is custom built for the mediatory TS and therefore it is easier to generate adapters that are less complex and perform better than their counterparts of the direct approach.

33% (6/18) of the tools, Vorto, Karma, Microsoft SSIS, Oracle Data Integrator, Stylus Studio, and Open Mapping Software implement the indirect mapping approach. In Vorto, users specify models of the real world devices thus abstracting from the pure data and schema structures. Afterward, mappings are created at a higher level of abstraction using only device models while the tool should be able to automatically generate adapters at the data and schema abstraction level. Karma and Open Mapping Software map TS data to a common representation created by analyzing common syntactic and semantic elements of source and target structures. Based on such a mapping, Open Mapping Software provides generation of integration adapters, while Karma only allows analysis of data based on the common ontology. As Microsoft and Oracle are well-known database system vendors, SISS and ODI use RDBMS TS as a mediatory TS and SQL language as an expression and transformation language. All structures specific to a mapped TS, must be first mapped onto database structures by a user before creating source-to-target mappings. Stylus Studio requires that all TS are first mapped to XML documents and only then the mappings can be specified using the XQuery language specific to the XML TS.

4.5.4 *Mapping and expression languages*

In this survey, we have identified four different syntaxes of mapping and expression languages: (i) **graphical**, (ii) **textual**, (iii) **tabular**, and (iv) **configuration-based** syntax.

72% (13/18) of the tools provide single mapping language syntax. Usually, in 56% (10/18) of the tools only the graphical syntax is provided as the data integration languages have traditionally used such syntax. 11% (2/18) of the tools tools have a configuration-based syntax, while only Vorto, 6% (1/18), has a textual mapping language. Tabular syntax was always used in combination with other syntaxes and no tools provided just this concrete syntax.

Combination of two mapping language syntaxes is present in the 28% (5/18) of the tools. MetaDaper and Karma, 11% (2/18), offered two syntaxes in which a user is able to specify the whole mapping using only one of them. On the other hand, Analytix Mapping Manager, Stylus Studio, and Microsoft SSIS, 11% (2/18), provide two syntaxes that must be used together in order to create a valid mapping specification.

Similar situation can be seen when analyzing the expression language syntax. 67% (12/18) of the tools provide a single syntax, most often it is a textual syntax (33% (6/18)) but also graphical (22% (4/18)) and configuration-based (11% (2/18)) syntaxes are also present. In the case of 33% (6/18) tools, multiple syntaxes are provided. The most frequent combination of expression language syntaxes is the combination of graphical and textual syntaxes (28% (5/18)), while only Karma (6% (1/18)) has a mixture of configuration-based and textual syntaxes.

While using the tools, we felt the most comfortable in using the tools that had the following combination of syntaxes: graphical syntax for the mapping language and a textual syntax for the expression language. Graphical syntax provided us with a good overview of the whole mapping specification, while the textual expression language was similar or exact syntax of the programming languages we regularly use. Also, such tools, especially the ones from the ECTL tools category, provided a master–detail view for the specification of mappings. In a master view, a general algorithm or a ECTL process is specified where the abstract components such as data readers, data writers, and transformers are specified. Once users open the transformer block they can specify the mappings in a usual, graphical way.

We found graphical syntax to be the most suitable for the integration of small to medium size data schemas such is the case with our example. This syntax provides the best readability, the shallowest learning curve and the user is able to start integrating TSs very fast. However, the graphical syntax used can lead to overcrowded diagrams if many elements are mapped at once. Too many lines intersect and it is hard to get an overview in a glance. If this syntax is combined with a graphical syntax of the expression language, like it is the case with Altova MapForce, Liquid XML Studio, and Clover ETL, the diagrams become almost unreadable. These tools tackle this issue with the ability to group all expression elements together into one, but it does not solve the issue entirely.

Tabular representations of mappings can alleviate the problem of overcrowded diagrams. Mapped elements are shown in adjacent cells and users can easily sort, scroll, and search cells to get a desired information. Analytix Mapping Manager relied on the tabular representation of mappings with some of the mapping configured using a sequence of dialogs, while Microsoft SISS provides a mixture of a graphical and tabular mapping language. Tabular representations are often used with textual expression languages. Expressions are either written in a cell between the mapped elements or in a pop-up dialog. The major drawback of the tabular representation is that is somewhat hard to easily get an overview of the whole mapping specification without taking a detailed look at the table context. Additionally, users that are not experienced with a tabular data representations, such as ones that are present in RDBMS and Microsoft Excel, can find the syntax pretty hard to get used to and to work with.

Unlike tabular and graphical representations of mappings, the tools that use configuration dialogs and forms to define mappings between source and target elements do not show the mappings between elements directly but only show the end result of the last configured transformation step. The tools that mainly use this approach are OpenRefine, Open Mapping Software, and Meta Dapper. The main benefit of such a solution is that a mapping is done in discrete steps and for each element mapping the result is immediately visible. However, this kind of an approach greatly hinders the possibility of teamwork and the lack of support for agile development of mappings. Also, we found out that it was much harder for us to get an overview of the whole mapping without going into details of every single element mapping that was created.

4.5.5 *Reuse*

We have identified that the tools support the reuse of three types of concepts: (i) **user-defined functions**, (ii) **mappings**, and (iii) **TS specifications**. The reuse of only one concept was provided by 56% (10/18) of the tools, most often the reuse of user-defined functions (28% (5/18)), the reuse of two or more concepts in 39% (7/18), while only Open Mapping Software (6% (1/18)) did not have reuse mechanism implemented.

User-defined functions are elements that are reused most often, in 61% (11/18) of tools. Often, the user is able to group existing functions in a new unit, store it in a repository and reuse the new functionality in the current project or globally, in other projects. Although this kind of reuse is frequent, it is a basic one. It contributes to the customization of the product and speed of mapping creation, as users may build a custom function library over time that helps them in the problems from their domain. However, creation of custom functions in the expression language does not contribute to the automatic adaptations of solutions and to the (semi-)automatic creation of mappings.

TS specification reuse provides a way of reusing TS importers and exporters between projects. It is also a kind of reuse that contributes to the automation of the mapping process as a tool may have a repository of TS binders at its disposal and reuse the most appropriate one when it is needed. Although 61% (11/18) of the tools allow for the custom TS to be connected to the application, most of these tools provide this option in a one-time-only manner. For example, Altova MapForce FlexText utility provides a way to import a custom TS data schema by using a combination of a graphical schema editor and regular expressions. Such a FlexText importer is not reusable, and the next time a TS needs to be imported the whole process has to be performed from the beginning. On the other hand, MuleSoft Anypoint Studio, Vorto, Adeptia Integration Suite, and Clover ETL provide a mechanism for reusing custom TS importers and exporters.

The reuse that can contribute to the automation of mapping specification is the mapping reuse. Once created, ideally, mappings should be reusable in new, similar context with just a little or no user intervention. 28% (5/18) of the tools, FME Desktop, OPC Router, Adeptia Integration Suite, Karma, and OpenRefine, provide the ability to reuse the whole mapping specification when the same combination of source and target data schemas are encountered. Karma is the only tool that provides reuse on the level of single element mapping, while all other tools provide reuse at the level of the whole mapping specification. Reuse at the element-level mappings provides a possibility of mapping automation as previous knowledge can be consulted and element mappings recreated from several different specification to form a new, adapted mapping specification applicable in the current scenario. That is why it is unfortunate that only one tool out of eighteen that were tested, offers a sort of intelligent integration, i. e., integration based on the previously stored knowledge.

4.5.6 *Code execution*

Once the mappings are specified they have to be executed in order to generate target data from the source data. We have identified two approaches to code execution (i) **internal**, in which a tool executes mappings internally, without creating external adapters, and (ii) **external**, in which external adapters are created and executed independently of the tool. 50% (9/18) of the tools implemented only one approach to code execution, while 44% (8/18) had both approaches implemented and only Karma (44% (8/18)) had no code execution capabilities. Karma only provides the generation of RDF files that could be included in other tools for ontological analysis. This is due to the fact that Karma is not a usual ECTL tool, but a tool that only provides mappings between TS and a central ontology and then provides analysis capabilities on this ontology.

Internal code execution is provided by 72% (13/18) of the tools and for 28% (5/18) this is the only implemented approach. Internal generators provide a convenient way to test the mapping specification without user leaving the tool. However, the tools that only implement this approach must be always used as integration adapters. This may cause a problem due to the fact that computers, on which the integration is run in industrial cases for example, often have limited resources and the integration tools often require a lot of resources to run. Therefore this option is not preferable in real-time use cases.

External code execution is provided by 67% (12/18) of the tools and 22% (4/18) of the tools provide only this approach. Tools that provide this option generate executable adapter which implements the specified mapping and is able to be run independently of the tool. There are few different ways that the surveyed tools generated adapters and executed them. First, some tools generate a new adapter for each new mapping specification that is created. These are a majority of tools which include: Altova MapForce, OPC Router, Vorto, Liquid XML Studio, Stylus Studio, Informatica PowerCenter, Microsoft SSIS, and Talend Studio. This way, the best performance of the generated adapter can be achieved as it is adapted to the concrete combination of TSs. Such adapters can be run independently as separate applications or as operating system services like it is the case with the OPC Router. Second mechanism comprises the generation of a configuration file that configures a generic adapter. This way of code generation is provided by: AnalytiX Mapping Manager, Open Mapping Software, MetaDapper, and MuleSoft Anypoint Studio. However, the drawback of these generic integration adapters is that they must contain means to execute every possible configuration that can be created. Therefore, they tend to be larger in size and, in our opinion, they are slower than the specially tailored generators provided by other tools.

4.5.7 *Extensibility*

We consider the two main aspects of customization and extensibility for each of the surveyed tools: (i) **TS extensibility**, which includes adding the support for new TSs, and (ii) **functional extensibility**, which includes the possibility to extend the current expression language and code generators.

TS extensibility is one of the most important characteristics we have observed. As different application domains usually have different TS in which they operate, users want to extend the tool to work in their application domain. Thus, users must be able to import data and data structures from that TS to the tool and afterward work with the loaded structures. 61% (11/18) of tools provide the mechanism for adding new TSs. On the other hand, 28% (5/18) of the tools do not provide such a mechanism, while 11% (2/18), Karma and Altova MapForce, provide only a partial mechanism for extending TSs.

Extending other functionalities of the tool, except the TS import and export, allows for the fine-grained customization of the tool to a specific application domain. Sometimes, if a TS is imported, it might be useful to provide a specific set of function and generators to fully utilize the specificness of that TS. 22% (4/18) of the tools provide only the mechanism for extending the expression language, mainly in the form of user-defined functions. Only AnalytiX Mapping Manager (6% (1/18)) provides just a mechanism for extending generators. Both expression language and generators are extensible in 61% (11/18) of the tools while 17% (3/18) do not provide an extension mechanism.

It is worth noticing that in addition to inherently good extensibility of open-source projects, commercial projects also provide interfaces and extension points for users to customize their tools according to the needs in an application domain. Of all commercial tools, we have to single out FME Desktop as the tool that provides the highest level of extension, and as such, our opinion is that it can be applied in the most application domains.

4.6 SUMMARY

In this chapter we have presented the setup, execution, and the results of our mapping and integration tools survey. We have observed a set of 10 functional and nonfunctional characteristics of the tools and identified their values while implementing the example from Section 4.1. This example was tailored so as to provide us with the best insight in the tool capabilities of interest to us while not requiring deep knowledge of tool internal mechanisms. Although we are aware that with this example we cannot check the domain and functional coverage of these tools nor the performance of the mapping execution, the example was good enough to draw conclusions about the set of characteristics we have established in Section 4.1.

Based on the most frequent characteristic values of the surveyed tools, we can construct the profile of a generic integration tool with the most desirable characteristic values. By the results of our survey, a generic integration tool can be seen as the tool that:

- has a proprietary/commercial license;
- is distributed as a desktop application;
- provides means to solve problems in the data integration domain;
- uses the direct approach to mapping specification without requiring for the source and target TS elements to be mapped to a mediatory TS by user;
- provides a graphical syntax for the creation of mappings between source and target elements;
- provides a textual syntax for the creation of mapping expressions used for the fine-tuning of the mappings;
- facilitates both internal and external mapping execution;
- is extensible with new TSs as required by new application domains not anticipated while building the tool; and
- provides mechanisms to extend both expression language and code generators and thus allowing for the whole tool to be adapted to a new application domain.

After using all the tools, we found that the tools with these characteristic values provide the most flexibility and functions to users that want to solve problems in the data integration domain. While implementing the tool that supports our approach to TS integration, we have tried to implement the aforementioned characteristic values. However, there are some slight differences.

As we follow MDSD principles to create mapping and expression languages, we can specify more than one concrete syntax for the same abstract syntax. This has been one of our major remarks for the surveyed tools. Most of the tools provide just one concrete syntax, with no abstract syntax explicitly defined. This leads to issues with different types of TS data, where the used syntax is not flexible enough to be adapted to the new use case. The graphical syntax is suitable for integration scenarios with low to medium complexity, while tabular syntax is suitable for scenarios with high complexity. By having multiple syntaxes, we would be able to provide different views on top of the same abstract syntax and thus the tool can be used in a variety of scenarios. Though we are aware of this advantage, for this thesis we have implemented just the graphical concrete syntax. This is due to the fact that our case studies comprise of low to medium complexity TS schemas and this syntax proved to be the most useful. However, we tried to keep the graphical syntax as simple as possibly by a fully textual expression language. This solves the problem of rapidly overcrowded diagrams like it is the case with tools like Altova MapForce which use graphical syntax for expressions as well.

MDSD principles were followed by Vorto and Open Mapping Software but in a slightly different manner compared to our approach. They have incorporated MDSD principles in a way that each integrated TS is represented with a highly abstract model and the integration is achieved at the model level. However, this way, some of the specificness of data schemas from the integrated TSs that need to be considered in the integration, especially in the real-time device integration, are overlooked.

Another main difference between our tool and the generic integration tool presented in this section is the granularity level of reusable components. All the tools provided the reuse of user-defined functions, TS specifications, or mapping specifications. The most important type of reuse is the mapping specification reuse although it is the hardest to accomplish. With this type of reuse, whole mappings could be reused and adapted to a new integration scenario, resulting in the (semi-)automation of the integration process. This, however, requires the reuse and adaptation at the level of individual element mappings, not just at the the level of the whole mapping like it is the case in majority of surveyed tools. Karma provided mapping reuse at level of element mappings, however this was mostly dependent on the ontological reasoner and the automatic deduction process incorporated in the tool. Further, as Karma does not provide mapping execution engine, this kind of reuse is not so useful in the data integration domain.

In the following chapter, we will present our approach and the developed tool taking into consideration our experience and conclusions drawn from tool survey presented in this chapter.

5 THE MAPPING APPROACH

In this chapter we present the mapping approach which is a central point of our research. First, in Section 5.1 we give an overview of main steps preceding the use of our mapping approach. These preceding steps are aimed at customizing the integration tool and providing the necessary means to facilitate the application of the mapping approach. By giving the overview, we want to emphasize the importance of the steps that are often overlooked when an integration adapter is implemented by using an integration tool. The described process is based on our previous experience in adapter implementation and on the conclusions of the integration tool survey presented in Chapter 4.

Tool customization process is identified in almost all of the surveyed tools. However, each approach supported by these integration tools, has a different process for implementation of integration adapters. In Section 5.2 we present our vision of such a process. With this approach, we tackle often overlooked problem of transformation rules reuse. Our goal is to provide an approach that can be applied on any TS and allow abstract specification of transformation rules which would ease the reuse and make it more obvious to factory engineers that are not experienced programmers.

In Section 5.3 and Section 5.4, we present main notions of our mapping approach: (i) the mapping language and (ii) the automation process. After all conceptual elements are introduced, in Section 5.5 we present the tool named AnyMap which we have built in support of our mapping approach. The tool architecture is presented together with the brief overview of the tool modules and technologies used in their implementation.

5.1 INTEGRATION PROCESS

Our mapping approach can be seen as a process for the specification of integration adapters which in turn is just a sub-process of a larger, generic integration process. There are several important steps each developer of integration adapters performs either implicitly or explicitly before the adapter specification. These often overlooked steps can greatly influence the overall time needed to develop an adapter. Although the execution of these steps is not essential to our mapping approach, by explicitly describing them we provide a context in which our approach is intended to be used. Further, by explicitly acknowledging the importance of these steps, we can provide a rationale behind some of the decisions made during the development of the AnyMap tool. Main steps of such a generic integration process are presented in Figure 15 and each of these steps is explained in more details in the rest of this section. In the text, names of process steps and sub-processes are written in italics.

As an input for the process, a developer provides source and target TS definitions, integration tool and method. Prior to starting with the integration, the developer identifies one or more source and target TSs and defines their scopes based on previous knowledge, experience, and documentation of elements being integrated. After TSs are identified, the developer is presented with a choice: either to use an integration tool or to implement the integration adapter manually in a preferred programming language. As we already discussed in Chapter 1, the latter option has some significant drawbacks such as time-consuming and error-prone development and reuse not being at the satisfactory level. As our approach is supported by our AnyMap integration tool, steps of the manual adapter development process are not of interest

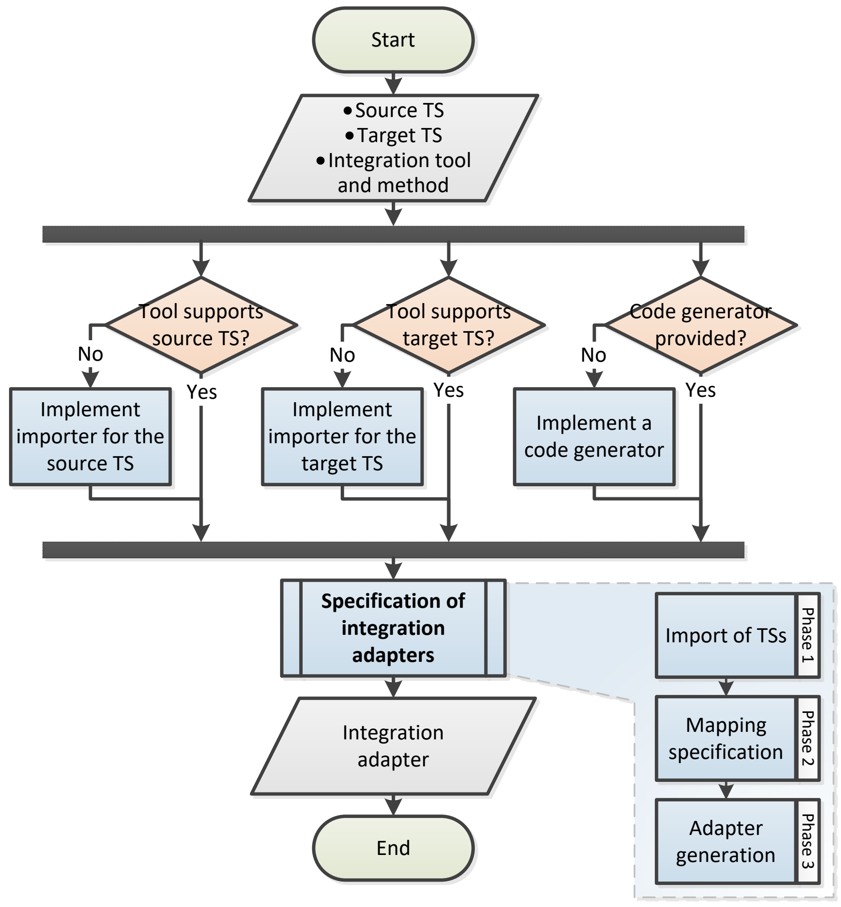


Figure 15: The main steps of the integration process

in this chapter and therefore are omitted from Figure 15. Therefore, the chosen integration tool and the integration method it supports are given as an input to the process.

Both selection of an integration tool, prior to the process, and the tool usage are influenced by the number and type of identified TSs. For the developer, the best outcome is that he or she finds an integration tool which supports all TSs that participate in the mapping. In such a case, the developer can immediately start to develop integration adapters in the selected tool without having to extend it in any way. However, often it is a case that integration tools do not support all the participating TSs. Further, an issue may arise when the tool does not support a TS at a particular abstraction level. To explain the latter issue let us consider RDF and XML TSs while having in mind the inherent broadness of the technical space notion. Although these TSs may be considered as completely separate spaces, RDF documents are often serialized in the form of XML documents. If this is the case then they are called RDF/XML documents. Therefore, we may put these RDF/XML documents in either TS depending on the integration problem being solved. The integration of these documents requires that the tool has different TS importers to be used in different use cases. Furthermore, if we consider the content of an RDF/XML document, it might be necessary to create such an TS importer that would take advantage of specific constructs such is the representation of SUS entities in triplets (Subject, Predicate, Object) and contents of which a generic XML importer cannot make advantage. Therefore, such a family of RDF/XML documents that is important to a specific integration problem, can be considered as an own TS which is in fact a sub-space of both XML and RDF TSs.

Extending an integration tool with new TS importers (*Implement importer for the source TS* and *Implement importer for the target TS*) can be a beneficial in the long-run. TS importers need to be implemented only once for a specific TS and can be reused afterward if the same TS needs to be integrated again. Such recurrence of TSs is often encountered in many domains, and once an integration tool is extended for a domain it is highly probable that it can be used successfully with no or little future extensions. Of course, in order for the developer to implement these TS importers, the tool must provide the means to make such an extension. Although not all of the tools support this, in our survey we have identified that a majority of the tools support TS extension. In this section we are considering only extensible tools that can be adapted to support the integration process in any domain. Therefore, if developers do not find appropriate tools with which they are able to create an adapter without extending it, they must first create appropriate importers for source and target TSs. These importers can be created from scratch or based on existing importers. Also, these process steps can be performed in parallel as they are not mutually dependent. It is even possible that different developers implement different importers in parallel and thus speed up the entire process.

In addition to supporting source and target TSs, an integration tool must provide appropriate code generators that will provide means to generate integration adapters that are to be executed on a desired platform, i. e., execution environment. In the process of selecting an integration tool, availability of desired code generators and execution engines plays an important role. The developer needs to select an appropriate code generator that is the most suitable for the problem domain and the current tool and technology landscape at developers' disposal. If there is no appropriate generator, the developer needs to implement such a generator. It is done as a part of the *Implement a code generator* process step. Again, extending a tool with a new code generator is possible only if the tool supports such an extension. In this thesis we are considering only extensible tools that can be extended with a new code generator.

In the case of integration process supported by an integration tool, the *Specification of integration adapters* sub-process can be started only after the importers for both TSs are provided and appropriate code generators are available. The output of this sub-process is an integration adapter which is also an output of the whole process. The steps of the *Specification of integration adapters* sub-process are discussed in detail in the following section.

According to the four-level architecture, in Figure 16 we present the most important activities of the integration process and their classification according to meta-levels. The names of activities are bolded for the activities that are performed by the users.

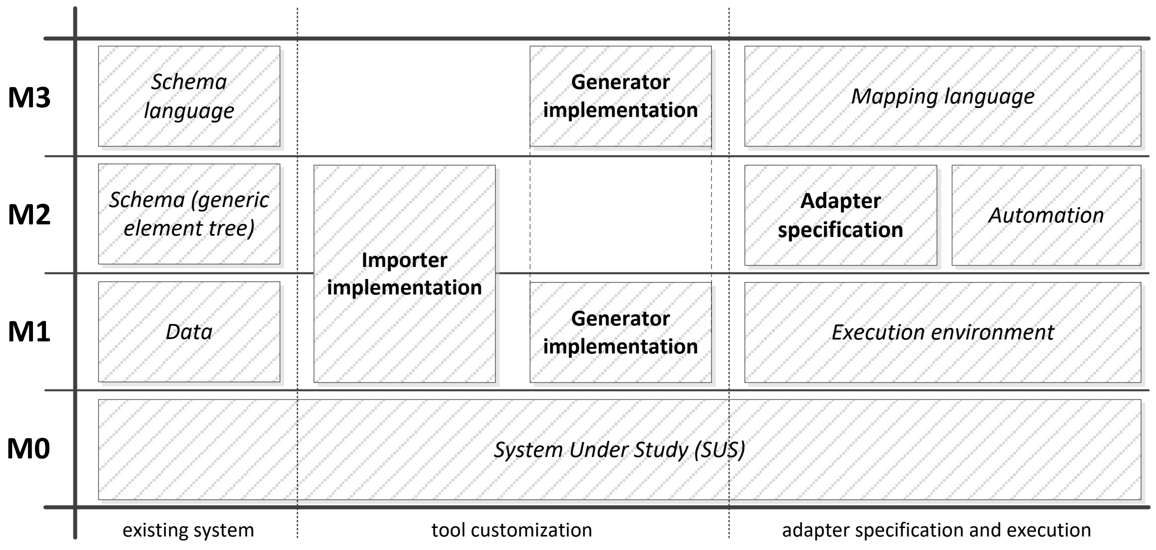


Figure 16: Main integration activities according to meta-levels

For the implementation of the TS importers, a developer needs to be familiar with the internal rules and concepts of a TS at both *M1* and *M2* meta-levels. There are two scenarios for importing a schema from a TS: (i) importing a schema directly from an existing schema file and (ii) importing a schema from an example data file. An importer has to be able to read existing TS schemas (*M2*) if they exist and transform them into a structure supported by the tool. In the latter scenario, importer should be able to construct schemas from existing data files (*M1* and *M2*) as the schema is needed for the adapter specification. Further, as data files are needed for the execution of adapters, importers need to be able to read and write to these files (*M1*). Due to this need, importers are often bundled with a generated adapter.

A generator developer needs to be aware of the concepts of both mapping language (*M3*), used for the specification of adapters, and the execution environment internals (*M1*) so as to be able to generate executable code for that environment. Code generator developer needs to be familiar with the mapping language elements as the code generator needs to parse mapping specifications in order to provide an output, i. e., to generate executable code for executing the mapping. Also, knowledge of *M1* level is needed as the developer must provide code templates that would be filled with the parsed values from the mappings. These code templates and parsed values together provide executable code constructs that are executed by the execution environment to transform data files from source TSs. Code templates are custom made for the execution environment for which the generator is built.

Adapter specification can be done by a factory engineer that is not experienced in programming as he or she only needs to be trained in the domain-specific mapping language and needs to know transformed schemas. The adapter specification is only performed at the level of schema elements (*M2*). In order to speed up the process, an engineer can use one of the provided automation mechanisms. Automation is also executed at the (*M2*) meta-level as it needs only to access schema elements in order to provide a list of element mapping candidates to be created in the current mapping. Based on the degree of automation in the candidate selection process it can be performed automatically or semi-automatically. Automation is described in more detail in Section 5.2.2 and Section 5.4.

After an adapter has been specified and generated, it is executed by the execution environment on the provided source data files. Therefore, adapter execution is performed at the *M1* meta-level.

Therefore, an integration tool needs to be set-up to support TSs being transformed and this is usually being done by developers that know the tool and the mapping language. Once the tool is set-up, an adapter developer can perform all the tasks at the *M2* level.

5.2 SPECIFICATION OF INTEGRATION ADAPTERS

In the previous section we have given the overview of the generic integration process and we have presented in detail the steps for the customization of an integration tool which precede the adapter specification step. Such a generic process can be supported by many integration tools as it can be concluded from the results of the tool survey presented in Chapter 4. Although most of the tools support these steps in a similar way, the main difference between the tools and their approaches is the *Specification of integration adapters* sub-process. This sub-process is often implemented differently and usually represents a main selling point of the tool. All other steps and sub-processes are performed just to provide facilitators for the adapter specification.

In this section, we present our view of the *Specification of integration adapters* sub-process structure. Later, in Section 5.5 we present the AnyMap tool which we have developed to support the approach.

The *Specification of integration adapters* sub-process comprises steps that provide the means through which a developer can create source-to-target mappings between source and target TSs. Mapping specifications are created either manually or (semi-) automatically. From these mapping specifications, executable integration adapters are automatically generated. We divide the *Specification of integration adapters* sub-process into three distinct phases: (i) *Import of TSs* (Section 5.2.1), in which a developer imports all participating TSs into an integration tool, (ii) *Mapping specification* (Section 5.2.2), in which a developer specifies source-to-target mappings between TSs imported in the previous phase, and (iii) *Generation of integration adapters* (Section 5.2.3), in which the executable adapter code is generated based on the mapping specification created in the previous phase. All of these phases are presented in more detail in the following subsections.

As the presented sub-process is supported by the integration tool, each process step can be performed either manually by the adapter developer, automatically by the tool, or semi-automatically where both the developer and the tool are participating in the activity execution. Therefore, all figures with diagrams depicting process steps in this section comprise of two swimlanes: (i) the developer swimlane, in which all the process steps are performed by a developer, and (ii) the AnyMap tool swimlane, in which all the process steps are performed by the AnyMap tool automatically, without developer's intervention. The process steps that can be performed either by the developer or by the tool, are drawn at the border between the two swimlanes. The meaning of such notation is that the process steps can be performed entirely by the developer, entirely by the AnyMap tool, or partially by the tool. In order for the tool to partially perform process steps, a user must provide a configuration or an input to the tool.

In addition to the swimlanes, all process diagrams depict both process flow and data flow. All process steps are given with the full-line border, and connected with full-line arrow connectors. On the other hand, data flow and data elements which represent inputs and outputs of the process steps are given with dotted-line borders and connected to process elements with dotted-line connectors. In the rest of the section text, the names of process steps, data artifacts, and sub-processes are written in italics.

5.2.1 *Phase 1: Import TSs*

The *Specification of integration adapters* sub-process starts with the acquisition of data schemas and data files in the integration tools. The *Import TSs* phase of the *Specification of integration adapters* sub-process is presented in Figure 17.

Our approach allows for the integration of software tools, systems, and devices, by transforming the exchanged data to the appropriate format. Therefore, in order to start creating integration adapters, a developer must first collect data schemas and data files from TSs that are being integrated. Data files can have a twofold purpose. First, data files are used during the adapter execution as they represent an input of the transformation. Second, if there is no data schema available, data files can be used as examples based on which a schema can be partially or fully recreated. Such data files are called example data files. All collected data files and appropriate data schemas are considered as an input for the *Specification of integration adapters* sub-process.

Mappings between source and target TS elements are specified at the schema level (*M2* level in Figure 16). By specifying the mappings at the schema level, the developer is specifying instructions on how to transform concrete data which are instances of the mapped schema elements. Therefore, an element mapping specified at the schema level is applicable for every instance of that schema element, i. e., for every piece of data conforming to the schema element

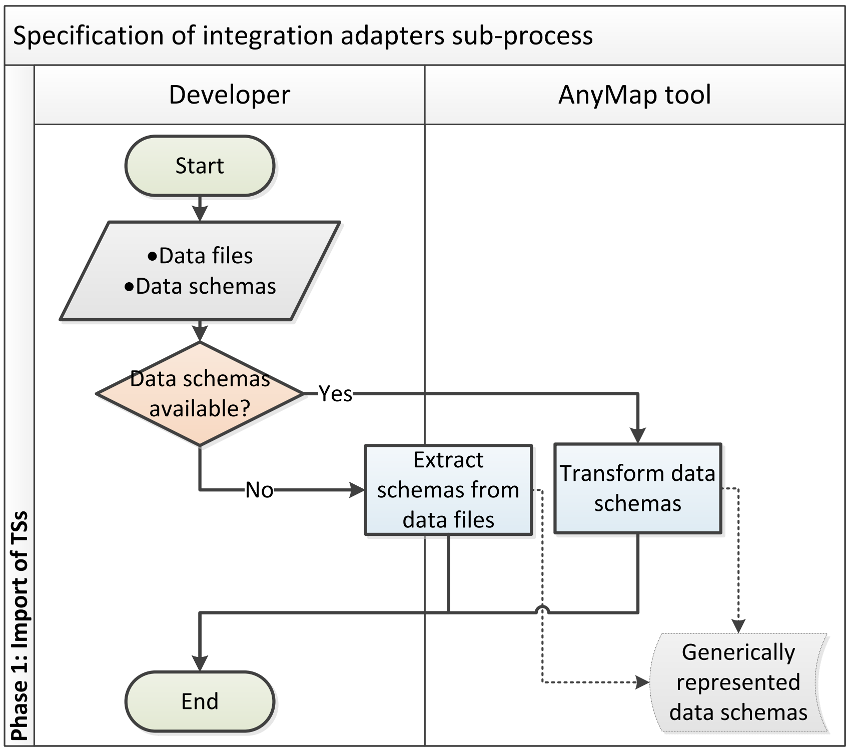


Figure 17: Specification of integration adapters: the *Import TSs* phase

being mapped. This corresponds to the notion of model-to-model transformations in MDSD where data schemas can be seen as meta-models and data files are viewed as models. Data schemas, example data files, and appropriate mappings represent a baseline or a recipe for the creation of integration adapters. During the execution, adapter receives data files from source TSs and, based on the internal rules generated from the specified mappings, produces data files in target TSs.

In some cases it can be hard to acquire a data schema for a source or a target TS. This could be due to a number of reasons, e. g., a data schema does not exist or it is not available to the developer, or the TS does not provide means and appropriate languages for schema specification. Regardless of the reasons, data schemas are necessary for the mapping specification and they must be provided. In such a case, a developer can use example data files from a TS as an input for a (semi-)automatic extraction of schemas (the *Extract schemas form data files* step). An example of a TS that does not have a dedicated schema specification language is the CSV TS. In this TS, schemas are implicitly specified in the data file itself in the form of the data header. In the case of CSV TS, sometimes a tool can automatically recognize data schema, e. g., if common separators and value encapsulations are used. If the proprietary separators and value encapsulation characters are used in a CSV document, manual intervention is needed in order to extract the schema. On the other hand, in TSs like JSON and XML, data schemas may not be created or the developer may not have rights to access them. In both of these cases a schema must be specified manually, in JSON Schema or XML Schema languages, or extracted from one or more example data files in a (semi-)automatic way, e. g., using JSON Discoverer [1] and Trang [2] tools. Therefore, for all of the source TSs, both data files, that can be used as example files too, and data schemas need to be acquired so the developer is able to create integration adapters

1 http://som-research.uoc.edu/tools/jsonDiscoverer/
2 http://www.thaiopensource.com/relaxng/trang.html

and execute them. For target TSs, only data schemas need to be identified and just data files could be necessary if a data schema does not exist and needs to be recreated.

In order for the integration process to be applied in the same way regardless of the TSs being integrated, all imported data schemas must be represented in a same, generic way. Such a generic representation is the result of both *Extract schemas form data files* and *Transform data schemas* process steps. In the former step, the generically represented data schemas are a product of the (semi-)automatic process of data schema extraction. In the latter step, the appropriate importer takes the collected data schema and converts it automatically to the generic schema representation. Such a generic representation is then presented to mapping developers throughout the remaining process steps.

We consider every data schema as a graph comprising of elements (nodes) and relationships between elements (links) as they are also considered in [134]. Every schema can be represented as a single-rooted graph, with a single root element and many child elements. A single root element may be explicitly specified in the schema or the schema document, i. e., file, can be considered as a root element in the graph. Many schema languages can be used to specify schemas as trees (e. g., SEMI Equipment Communications Standard/Generic Equipment Model (SECS/GEM)and CSV) but more often it is the case that schemas are specified as general graphs due to the shared substructure and referential constraints (e. g., XML and RDBMS).

In order to allow easier representation and handling, as well as to improve readability and storage, we have decided to use a tree representation of schemas for all schemas. The general graph schemas are converted to tree schemas by flattening the graph structure. Relationships between elements at the same level or between lower and upper levels of the tree, during the flattening can be represented as tree elements (nodes) with non-trivial type instead of considering them as elements. This can be achieved by copying referenced structure to the referenced place (e. g., complex types in XML) or by introducing a special reference element type for representing references where copying is not an option as it would introduce recursive and infinite structures.

The flattening is done at the level of TS importers and it is up to a developer of an importer to provide the transformation of the original schema structure to the generic tree representation. Flattening approaches can be implemented by following the research presented in [134] and the corresponding Cupid tool.

It is worth noting that this step, in which the schema is transformed into a generic tree structure, must ensure the two way connection between the original data schema and the generic representation. While the mapping specification process is performed on the generic representation, integration adapters that are generated based on the mapping specifications must be executed on the data files considering the original schema definitions. Therefore, in order for the adapter generator to be able to access original schema elements and execute operations specified at the schema level, links from the generic schema element to the original schema elements must be followed. We have named these links as bindings and TS importers as binders. In the rest of this chapter we will use the terms importers and binders interchangeably. The process of creating the bindings is described in more detail in Section 5.3.

5.2.2 *Phase 2: Mapping Specification*

After data schemas are identified and represented in a generic way, a developer can start with the creation of mappings. The *Mapping specification* phase of the *Specification of integration adapters* sub-process is presented in Figure 18. Therefore, the only required inputs for this process phase are generically represented data schemas from the previous process phase presented in Section 5.2.1. There are three approaches to mapping creation between these generic schema representations: (i) manual, (ii) semi-automatic, and (iii) fully automatic approach. The **manual** approach allows a developer to manually connect source and target schema elements and provide additional transformation expressions that will be later used for the execution of mappings. On the other hand, the automatic mapping approaches imply the usage of a reuse or an alignment algorithm for mapping automation. All approaches result in the mapping specification that is later used for the adapter generation.

If a developer decides to use the **semi-automatic** approach to mapping creation, the first thing that the developer needs to do is to select source and target schema elements that are to be mapped (*Select schema elements*). Although this is not a necessary step, in the case of large schemas, where only a small subset of elements needs to be mapped, such an element selection process can lead to a great increase in speed of the automation process. There are two possible automation algorithms: (i) reuse algorithm and (ii) alignment algorithm. In short, the reuse algorithms take previously created mappings into consideration when calculating element mapping candidates, while alignment algorithms take into consideration only the current source and target schema elements. These algorithms are presented in more details in Section 5.4

Once schema elements are selected, a developer needs to choose an automation algorithm (*Select a reuse or an alignment algorithm*), which belongs to one of the following two categories: reuse algorithms and alignment algorithms. Regardless of the chosen algorithm, the execution of alignment (*Create mapping candidates based on selected schema elements*) or reuse (*Create mapping candidates based on selected schema elements and content of a reuse repository*) algorithms requires a set of source and target elements as an input and produces a set of element mapping candidates as an output. Afterward, the developer can choose the most appropriate candidates for the current integration scenario. Once appropriate candidates are chosen, they are automatically applied to the current mapping context.

Although, in some cases, the mapping specification process may stop after the element candidates are created in the current mapping, a developer can choose to further manually adapt created mappings (*Manually improve applied element mapping candidates*). This is an optional step as a developer can be satisfied with a precision of the automation process. On the other hand, when the reuse repository contains just a small number of stored mappings the developer must manually fix the applied element mappings to improve the overall quality of the integration solution. However, even in this case the overall speed of development is increased as element mappings that are frequently created can be automatically discovered and applied by the automation algorithm. This requires less work from developers as they need to create just use-case-specific element mappings.

Once a mapping specification is created and the developer is satisfied with the result, he or she may choose to improve the reuse process by adding the created specification to a reuse repository. There are two main types of the reuse repository: local and global. Local repositories may be deployed to a single machine and maintained by a single developer or a group of developers that usually solve the same type of integration issues. Keeping the local repository focused on a single integration domain, the accuracy of the reuse algorithm is improved for that particular integration domain. However, such repositories tend to have small number of mapping specifications and as such, the reuse algorithm cannot provide element mapping candidates for the elements that have just been introduced to the integration domain and had no similar elements before. A global repository may be more suitable in such situations. In this kind of repository, a larger number of developers from different integration domains store their mapping specifications. Depending on the used tool and data security policies, a global repository may be deployed at a level of a company or on a public computer for a worldwide use.

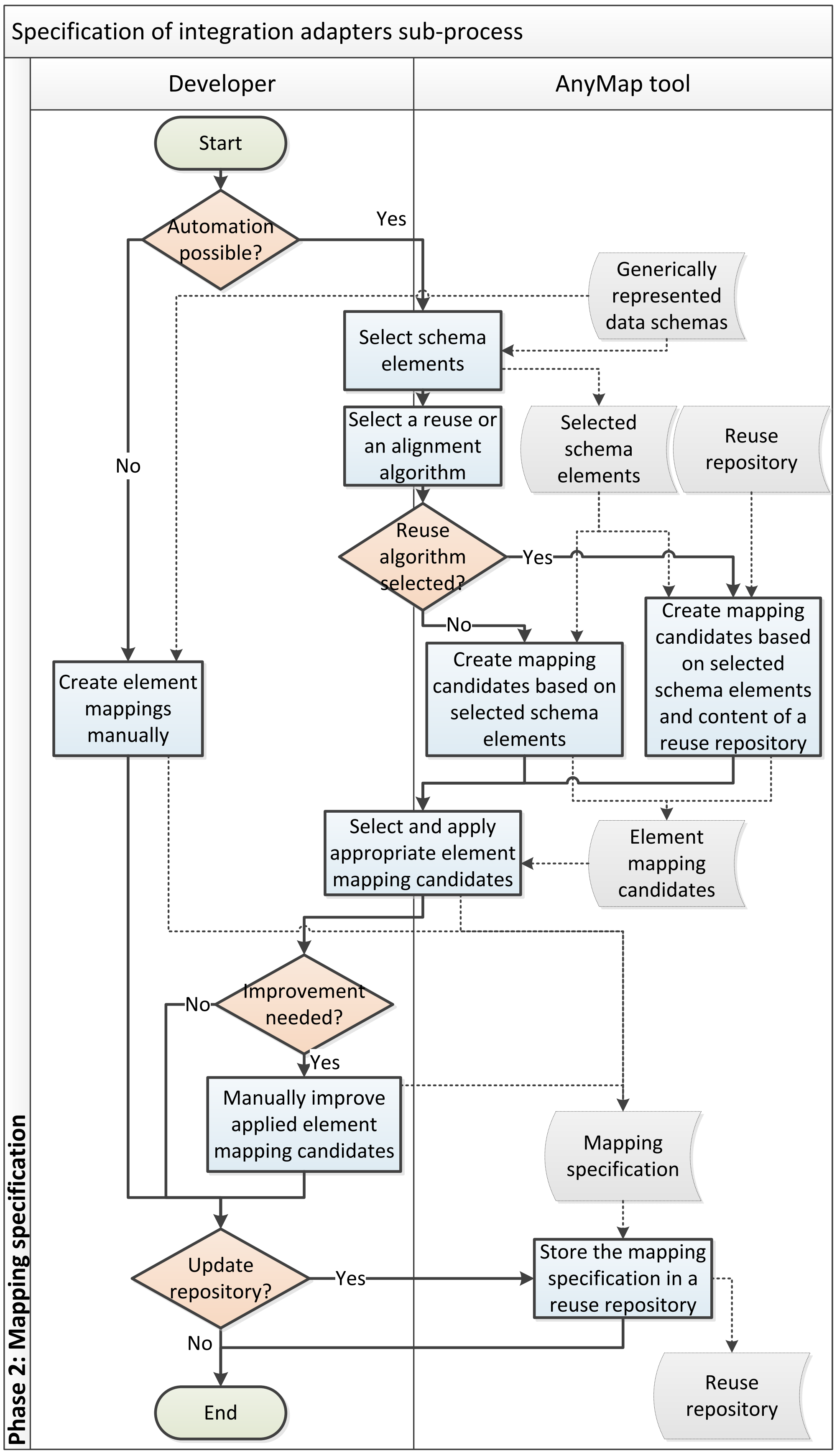

Specification of integration adapters sub-process
Developer
AnyMap tool
Phase 2: Mapping specification
Start
Automation possible?
Yes
No
Generically represented data schemas
Select schema elements
Select a reuse or an alignment algorithm
Selected schema elements
Reuse repository
Reuse algorithm selected?
Yes
No
Create mapping candidates based on selected schema elements
Create mapping candidates based on selected schema elements and content of a reuse repository
Create element mappings manually
Select and apply appropriate element mapping candidates
Element mapping candidates
Improvement needed?
No
Yes
Manually improve applied element mapping candidates
Mapping specification
Update repository?
Yes
No
Store the mapping specification in a reuse repository
Reuse repository
End

Figure 18: Specification of integration adapters: the *Mapping specification* phase

In the case of **fully automatic** mapping specification, the process activities from *Select schema elements* to *Select and apply appropriate element mapping candidates* are performed automatically. Additionally, the *Manually improve applied element mapping candidates* activity is not performed by a user. In the case of full automation, an integration tool may automatically select a subset of source and target elements following a predefined heuristic. Most often the tool would select all the schema elements. Afterward, the automation algorithm can calculate and choose only the candidates with the highest calculated probability of being appropriate for the current integration scenario. These candidates are automatically applied and the mapping specification is passed to the next step of the process. In this case, reuse algorithm is often used with the repository that contains significant number of mappings that are specific to the current integration domain. Therefore, the reuse algorithm is able to precisely reuse elements that are often repeated in these specifications.

The result of performing the *Mapping specification* phase is the mapping specification which is serialized and saved to the storage regardless of the fact if it was used to update the reuse repository or not. The mapping specification is later used in the *Generation of Integration Adapters* phase presented in the following subsection.

5.2.3 *Phase 3: Generation of Integration Adapters*

The mapping specification created after the execution of the previous process steps is just a schema-level abstract specification of the integration adapter. Once a developer is satisfied with such a mapping specification, he or she may choose to generate an executable integration adapter that is to be executed on a desired platform. The *Generation of integration adapters* phase of the *Specification of integration adapters* sub-process is presented in Figure 19.

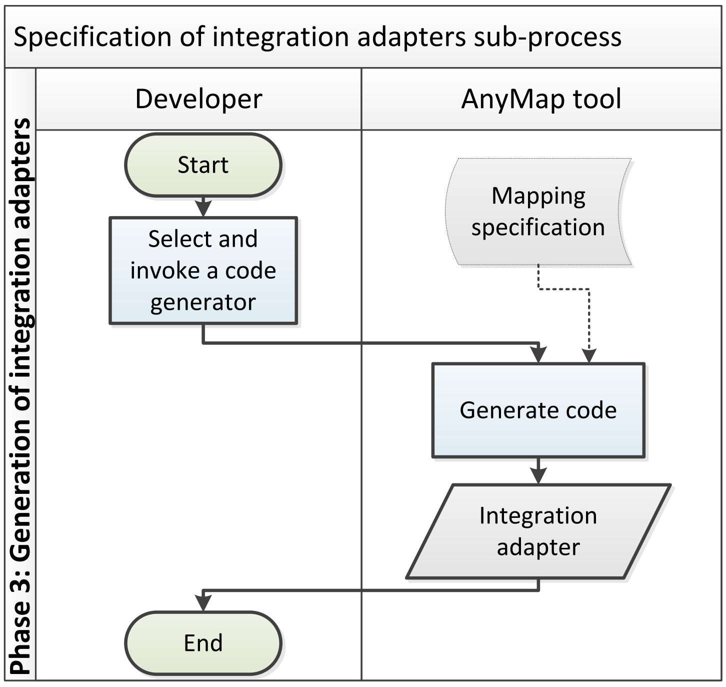

Specification of integration adapters sub-process
Phase 3: Generation of integration adapters
Developer
AnyMap tool
Start
Select and invoke a code generator
Mapping specification
Generate code
Integration adapter
End

Figure 19: Specification of integration adapters: the *Generation of integration adapters* phase

As it is already said in Section 5.1, a single integration tool may provide multiple code generators each in charge for generating code executed at a different execution platform (i. e., execution environment). A developer selects and invokes a code generator that is the most appropriate for his or her current integration scenario (*Select and invoke a code generator*). The code generation process *Generate code* is executed entirely by the integration tool and it takes a mapping specification as an input and produces an integration adapter as an output. The integration adapter also represents a final output of the entire integration process.

The generated integration adapter can be executed inside the integration tool or independently. In both cases, the process results with a data file conforming to the target schema. Such a target data file is produced by the generated integration adapter from the source data file according to the transformation rules specified in the mapping specification. Furthermore, generated adapters may be run, executed and shutdown once per each document that is served as an input or may run continuously and execute each time a predefined event is triggered. As our main focus is on the Industry 4.0 domains, the integration process is focused on the generation of external adapters, i. e., the adapters that are executed independently of the integration tool. Internal code execution may be useful in the development phase and for getting one-time transformations executed.

5.3 META-MODEL OF THE MAPPING LANGUAGE

In this section we present the meta-model, i. e., the abstract syntax of the mapping language which represents the core concept of our integration approach. The meta-model is implemented as a part of the AnyMap tool. The preceding versions of the meta-model were introduced in our previous papers [108, 109, 110]. In this section we provide a detailed description of the latest version.

All meta-model concepts are presented in Figure 20. Concepts used for the representation of data schemas and data values are represented as rectangles with a gray filling and described in more detail in Section 5.3.1. Rectangles with white filling represent the concepts used for the element mapping specification and are described in Section 5.3.2. In Section 5.3.3, we present the expression language used for the specification of imperative transformation rules in the Java programming language and based on the *Value* concept from the meta-model. In the rest of the section, names of meta-model concepts are given in italics.

The root concept of the meta-model is *Mapping*. It represents a single mapping specification which is used in the adapter generation process. Each mapping has its name (*name*), a set of source and target element containers that are used to represent loaded data schemas, and a set of element mapping specifications (*Operator* and *Link*) that are used to specify transformation rules at a higher abstraction level. Element containers and element mapping concepts are described in more details in the following sections.

5.3.1 *Generic Representation of Data Schemas*

The mapping approach aims to provide an abstract mechanism for specifying mappings regardless the underlying data schema technology, i. e., technical space. For this reason, we provide a generic tree representation of schemas (cf. Section 5.2.2). Although the schemas may be represented in a generic way by using a graph structure, we flatten out the structure to a tree as it is easier for the end user to comprehend this kind of structure. Furthermore, it is easier to represent these structures in a GUI tool and it allows easier and seamless user experience from the user interface standpoint.

For each loaded data schema a single element container (*ElementContainer*) is created and its name (*name*) is set in order to differentiate it from other loaded containers. Depending whether the container represents a source or a target schema, the appropriate value of the side attribute (*side*) is set. Possible side values are *SOURCE* and *TARGET* which are both literals defined as

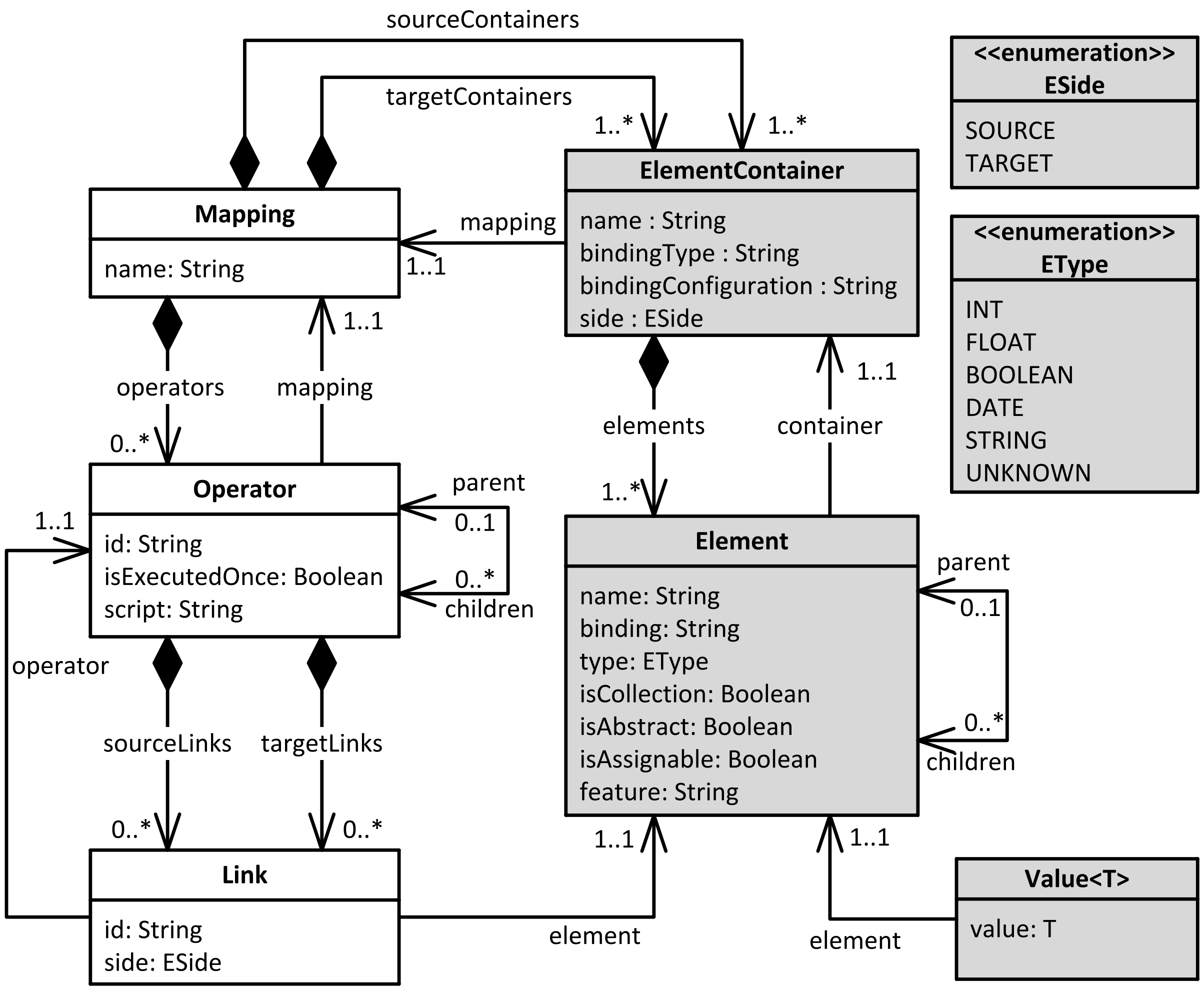


Figure 20: The mapping language meta-model

a part of the *ESide* enumeration. Each container is contained by a single mapping specification (*mapping*) either as a source (*sourceContainers*) or a target container (*targetContainers*).

Depending on the type of the loaded data schema, bindings between the generic representation and the original schema elements must be established in order to provide a uniform mapping specification process and adapter generation (cf. Section 5.2.2). Because the specification is performed at the level of data schema and the execution is done at the level of the data, an adapter needs to know how to read the original data based on the schema elements on top of which the mappings are specified. Therefore, at the level of an element container, a binding type (*bindingType*) and a binding configuration (*bindingConfiguration*) attributes must be set. The binding type determines the responsible binding component which is in charge of reading the data schema and reading and writing of data files. The value of this attribute usually uniquely denotes a binder that can read and write data in the appropriate TS. Based on the binding type attribute, a generator is able to bundle the appropriate binder component with the integration adapter in order for it to be executed. The binding configuration attribute is used by an appropriate binder, which is identified by the binding type, to store schema-level configuration while importing the data schema. Therefore, this attribute is binder-specific and can be formatted differently by different binders. The binding configuration is needed in order to recreate the element tree during the adapter execution as an adapter has only the mapping file at its disposal.

Each element container comprises one or more elements (*Element*) which represent data schema elements. The most important attributes of an element are its name (*name*) and a binding string (*binding*). Element name corresponds to the name of the corresponding schema element from an imported schema. The binding string stores the path, i. e., unique identifier of the schema element in the imported data schema document. The binding string is specific to a binder which sets it and can be formatted differently by different binders. For example, in the case of a CSV binder, a binding string is an ordinal number of column which is represented by the element, in XML binder this is an XPath expression, while in the case of an RDBMS binder may represent a fully qualified name of a table column. In addition to the binder-specific configuration, a binder may set an arbitrary feature (*feature*) for the element which will be later used in the adapter execution process.

For each element, a binder may set the values that specify the element type (*type*) and if an element is in fact a collection (*isCollection*), abstract (*isAbstract*), or assignable (*isAssignable*). By setting these values, a binder limits the total number of functions that can be applied to these elements thus preventing the unnecessary check at the adapter side by preventing the specification of inappropriate mappings at the first place. If an element is a collection, usual collection operators may be applied. Abstract elements cannot be source or target elements of an element mapping and they are usually just placeholders or grouping elements. Finally, non-assignable elements can only be used as a source but non as target elements of an element mapping.

The element type attribute is set just for the elements that are of primitive type and the possible values are defined by the *EType* enumeration. Complex types, such as objects, are represented as sub-trees in their element container. For example, an XML structure is represented as an element with its sub-elements and attributes also being its child elements (*children*) in the element container. Each of these elements that represent the object properties have a parent relationship (*parent*) set to point to the element representing the object. These properties do not depend on a specific binder and must be set by all binders. The *Value<?>* concept is used in the expression language and is described in more details in Section 5.3.3.

5.3.2 *The Element Mapping Specification Language*

Once the source and target element containers are created, the element mapping language may be used to provide a specification of the transformation rules that will be executed by the generated integration adapter. Such transformation rules are specified by the means of the element mapping specification language.

In the heart of each element mapping is an operator (*Operator*) which embodies a single transformation rule. Each operator is connected by links (*Link*) to a set of source and target elements that are being transformed. Depending on the number of source links (*sourceLinks*) and target links (*targetLinks*) of an operator, we classify operators as: one-to-one, one-to-many, many-to-one, many-to-many, and zero-to-any. All operators have an id (*id*) for their unique identification inside of a single mapping specification, and an operator script specification (*script*) for representing the transformation logic using an expression language (c.a. Section 5.3.3).

As it might be necessary to define the order of execution, all operators may be organized in a hierarchy. Each operator may have one operator which has to be executed before it (*parent*) and a set of operators which are executed after it (*children*). It is possible to create multiple operator hierarchies in a single mapping specification. All operators that are at the same hierarchical level are executed in parallel. Furthermore, as the execution of transformation rules depends on a cardinality of source elements, it is possible to specify whether the operator is executed only once (*isExecutedOnce*) for the first input element value, or multiple times, for each value received for the input element.

Each link that connects operators to source and target elements has its id (*id*) which identifies it uniquely in a context of an operator. Together with the connected element name, link id is used to provide variable name for unique identification of elements inside the operator script. In addition to the id attribute, for each link it is specified on which side of the transformation the connected element belongs (*side*).

5.3.3 *The Expression Language*

Although the mapping specification language presented in the previous two sections can be used to specify high-level and abstract correspondences between the source and target elements, a language for manipulating the input values is also needed in order to have a usable approach. Such a language, used for manipulating values received as an input to a transformation, is called the expression language. For each operator which represents a high-level element mapping, value-level transformation logic is specified in the *script* property using the expression language.

Based on our previous experience and on the integration tool survey (c.a. Chapter 4), we identified the two mandatory expression categories any expression language must have: (i) meta-level expressions and (ii) value-level expressions. The first category of expressions is used to access information about the schema during the execution of transformations. For example, in our survey use case, a target element (*id*) depends on the input schema element name (*Weight*) and not on its value received as an input (cf. Figure 10). On the other hand, more frequent is the need to transform the value received as the input of transformation and set it as a value of an appropriate target element.

Regardless of the expression category, an expression language must be a Turing complete language in order to allow computation of target values based on the source values. Therefore, there are two approaches to the expression language definition. One approach comprises creation of a proprietary DSL which may utilize the specificness of the mapping language structures in order to allow easier mapping specification. However, such a language must contain majority of operators and flow control statements already present in contemporary programming languages (e. g., arithmetic operations, string manipulation functions, for and if statements, and type casting) and also it must be sufficiently extensible in order to provide users with enough flexibility and reuse potential. On the other hand, such a language could be created especially with integration domain in its focus and as such it could omit a lot of unnecessary functions and standard libraries present in most of the contemporary programming languages. An example of such a language is the General Refine Expression Language (GREL) [3] language used in the OpenRefine tool [187].

The other approach to expression language definition is through adaptation of an existing programming language. This can be done by specifying one or more integration-specific structures and APIs in such a language and then using the standard language mechanisms on top of these structures. In a way, this process resembles building an embedded or internal DSL [78] which exploit the syntax of the host language while adding domain specific elements (such as an integration-specific structure). The advantage of reusing a language is that all of its mechanisms, statements and expressions are then available for defining the transformation logic. Also, if a host language is widely used, the user of the tool doesn't have to learn another language (i. e., proprietary expression DSL) but only the details of integration-specific structures.

3 https://github.com/OpenRefine/OpenRefine/wiki/General-Refine-Expression-Language

The downside of using an adapted general-purpose language is that it is not tailored specific to an integration domain and therefore contains a lot of unnecessary functions. As such, it may be slightly harder for a user not proficient in the chosen general-purpose language to adapt to it.

In our mapping approach, we chose to create the expression language by adapting a general-purpose programming language. In addition to all of the aforementioned benefits of this way of creating an expression language, another crucial benefit is related to the code generation process. During the code generation, from element mappings transformation rules are generated and from expression language scripts, concrete value transformation expressions are produced. If a custom expression DSL is built, the code generator must be developed in a way to transform these expressions to expressions supported by the execution environment which usually supports a widely used programming language such as Java, C#, Python, or C++. On the other hand, when using a special structure to denote values inside a script, such a structure can be easily created in any of the languages used in the execution environment. Therefore, a user may use the language specific to his or her execution environment and there is a high chance that he or she already knows such a programming language.

In order to provide a structure to be used in any general-purpose programming language and to support both categories of expressions presented in this section, we have created a structure that represents a value encapsulation. This structure is represented with a generic *Value<T>* concept presented in Figure 20. The template parameter *T* represents the type of the value *value* received as an input and which corresponds to the schema element *element*. Therefore, this structure encapsulates values and schema elements together which allows for the expressions to be made at both meta-data and data levels. Also, whichever general-purpose language is needed by the execution environment, *Value<T>* structure and the connected *Element* concept are easily translated to a structure in a target scripting language. The only precondition for the generation is that a target language supports generics or a similar concept (e. g., if a target language is a dynamically typed language).

For each input and output link of an operator, input and output variables are created in the script and can be used by the expression language to specify the transformation logic. Each of these variables is an instance of *Value<T>* where the *T* corresponds to the type of the schema element connected by the link. On top of these variables, transformation rule scripts can be implemented in a desired general-purpose language.

5.4 MAPPING AUTOMATION

One of the central ideas of the approach presented in this paper is the automation of the mapping creation process. Our goal is to automate mapping creation as much as possible by utilizing two related yet slightly different families of algorithms: reuse and alignment algorithms.

Automation is provided at the level of schema elements and not at the data level which makes automation algorithms more efficient and applicable in the domain of manufacturing industry. By applying automation at the schema level, multiple data files may be susceptible to the transformation specified by the same mapping which greatly improves the automation possibility. This is best observed when SECS/GEM, CSV and other schema-less protocols and data serialization formats are observed. These protocols implicitly contain schema definition in the data structures and formatting rules. Therefore, if a developer is to manually write a transformation rule, he or she would have to do it at the data level, limiting the reuse possibility. The reuse is limited as the variability at the data level is significant even for the data conforming to the same schema element. On the other hand, if a mapping is specified at the schema level it can be adapted for another schema element more easily and would be able to transform all instances of the new schema element as well as all instances of the original schema element.

In Section 5.4.1 we give an overview of the automation process with the description of main differences between reuse and alignment processes. The main difference between these processes may be seen in Figure 21. The alignment process is diagrammatically depicted at the left side while the reuse is depicted at the right side of the figure. Alignment process uses a schema matching algorithm to find similarities between source schema elements ($E_{s_i}..E_{s_k}$) and target schema elements ($E_{t_j}..E_{t_l}$). Each pair of source and target schema elements is compared and similarity is returned by the matching algorithm. The matching algorithm may be one of the algorithms from the papers presented in Chapter 3 or a proprietary algorithm developed by the company to fit their needs.

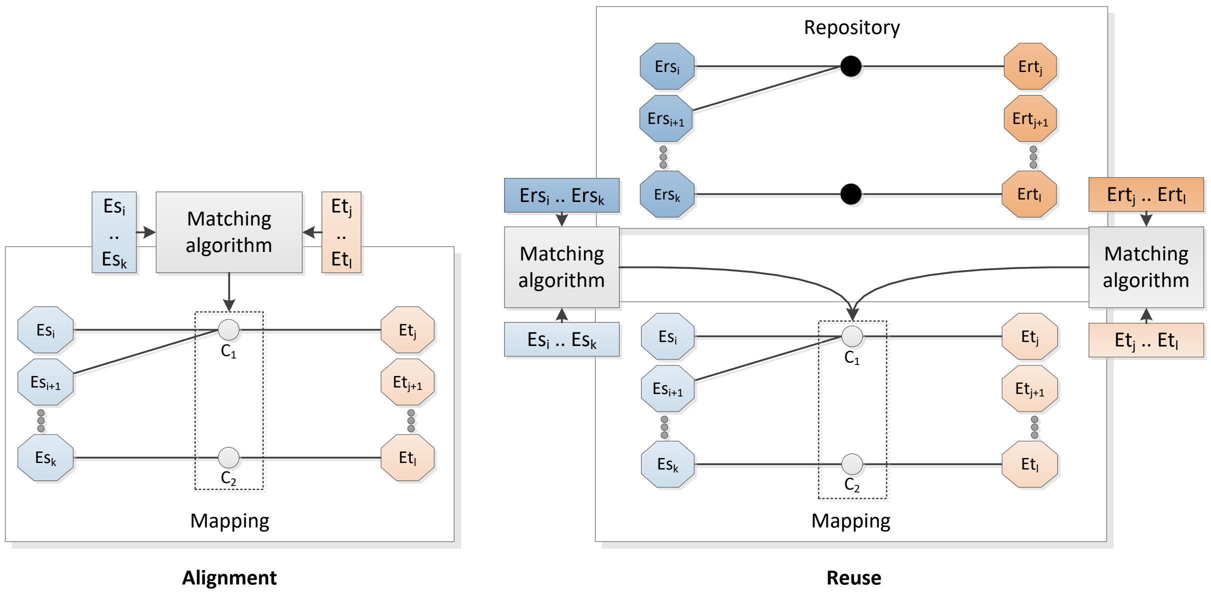


Figure 21: Comparison of the alignment and reuse processes

Same matching algorithms may be used in the reuse process as well. However, their role is somewhat different. Instead of comparing the source ($E_{s_i}..E_{s_k}$) and target ($E_{t_j}..E_{t_l}$) schema elements directly, matching algorithms are used to compare schema elements with the repository schema elements in order to find similar, previously created element mappings that can be reused and applied in the current integration scenario. Therefore, matching algorithms are used to calculate similarities between source schema elements of the current mapping ($E_{s_i}..E_{s_k}$) and source schema elements of repository mappings ($E_{rs_i}..E_{rs_k}$) and also between target schema elements of the current mapping ($E_{t_j}..E_{t_l}$) and target schema elements of repository mappings ($E_{rt_j}..E_{rt_l}$). Once the source and target element similarities are calculated, they are combined and returned for each repository element mapping. Therefore, for each repository mapping a probability that it fits the current integration scenario is returned. In the rest of the text we call it just probability. In order to automatize the process of choosing the right repository mapping, or to ease the choice process for a developer, a minimum probability threshold can be defined, to filter out unwanted element mappings.

Basic automation process, comprising only of reuse algorithm, was introduced in [54, 110]. In this thesis we give a more detailed description of the automation process that also includes alignment algorithms. The reuse and alignment processes and the formulas for calculating the element candidate probabilities are given in Section 5.4.2 and Section 5.4.3 respectively.

5.4.1 *Automation Process*

In both automation process variants, with reuse algorithm and alignment algorithm, the core automation process remains the same.

In the left part of Figure 22 we present the common process of finding element mapping candidates based on the mapping repository. The process has three phases. In the first phase, in the case of semi-automatic reuse or alignment, a tool user may select one or more source and target schema elements from the generic tree representation. These elements are considered as an input for the reuse or alignment algorithm. In the case of full automation, all source and target schema elements can be selected automatically and sent to the algorithm for element mapping probability calculation.

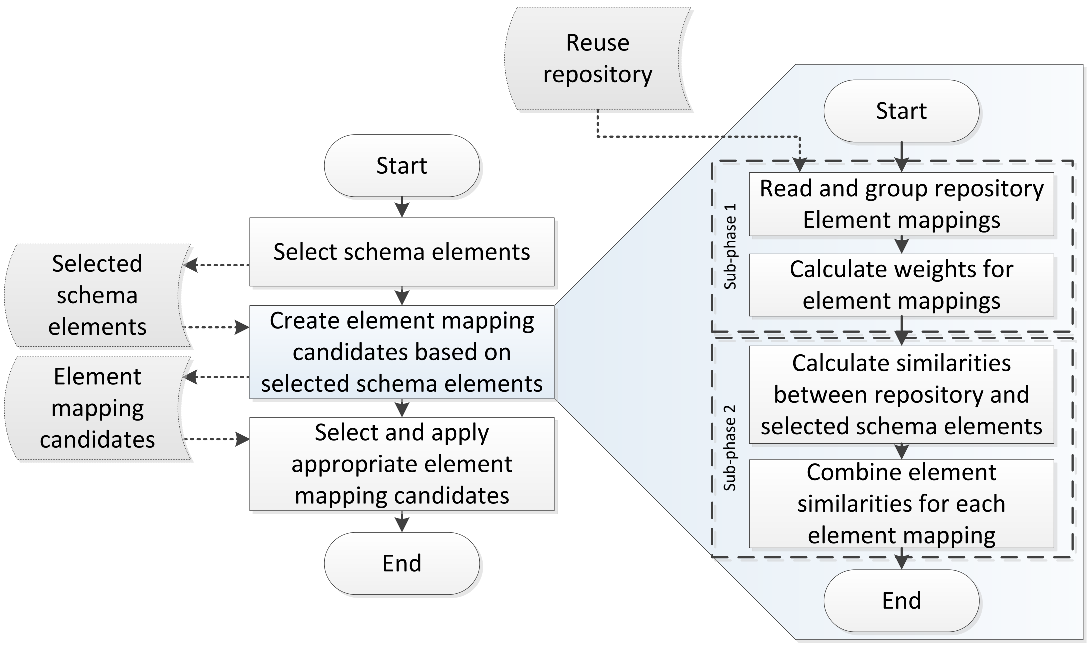

Reuse repository
Start
Select schema elements
Selected schema elements
Create element mapping candidates based on selected schema elements
Element mapping candidates
Select and apply appropriate element mapping candidates
End
Start
Sub-phase 1
Read and group repository Element mappings
Calculate weights for element mappings
Sub-phase 2
Calculate similarities between repository and selected schema elements
Combine element similarities for each element mapping
End

Figure 22: The automation process (left) and the reuse algorithm (right)

Once the elements are selected, they are passed to the reuse or alignment algorithm which is responsible for the identification of element mapping candidates and probability calculation. This process results in a list of all candidates with the assigned probabilities.

In the third and final phase of the element mapping candidate finding process, after all element mapping probabilities are calculated, they are applied to the current mapping. In the case of semi-automatic element mapping application, all candidate element mappings that have a probability above a defined probability threshold are presented to the user. The user may choose element mappings that fit best to the current mapping. In the case of the fully automatic process, element mappings with the highest probabilities are automatically chosen. In both cases, chosen element mapping candidates are then applied to the current mapping.

5.4.2 *Reuse Algorithm*

Reuse represents the ability to develop new applications with the use of existing solutions [97]. It is one of the core goals of the integration as it aims to develop an integrated information system via the reuse of existing applications without modifying them too much. Based on [136], we may classify matching algorithms as: (i) matching algorithms based on isolated element information, (ii) matching algorithms based on element structure, and (iii) matching algorithms based on element semantic. Our goal is to support all of aforementioned matching algorithm types and therefore automation process presented in the left part of Figure 22 passes all selected schema elements to the reuse algorithm which uses one or more matching algorithms to calculate similarity between schema elements. It is up to the algorithm to use these inputs in the best way and to calculate the fitting probability of the element mappings from the repository. Therefore, the approach allows for easy addition of matching algorithms as long as they can accept passed schema elements and produce a single probability value for each element mapping from the repository.

Our goal is to create a generic reuse algorithm that may be used in the creation of mappings between any two technical spaces. In addition to just considering the output of a matching algorithm, presented reuse algorithm also considers past executions and previous user choices in order to improve the accuracy of the automation process. In the rest of this section we present our reuse algorithm which is implemented as a plug-in for the AnyMap tool which is presented in Section 5.5.

The two sub-phases of the algorithm are presented in the right part of Figure 22. The first sub-phase of the algorithm is the pre-processing of all repository element mappings. During this step, a number of occurrences of each repository element mapping is calculated. Based on the number of occurrences, the probability of a repository element mapping is calculated as:

$$W_{S_r \to T_r} = \frac{N_{S_r \to T_r}}{N_{S_r \to \forall}}$$

$W_{S_r \to T_r}$ represents the probability of the $S_r \to T_r$ element mapping being the appropriate repository element mapping for the reuse algorithm. With $N_{S_r \to T_r}$, we denote the number of occurrences of the $S_r \to T_r$ element mapping in the repository. S_r and T_r are sets of source and target elements of a repository element mapping, respectively. With $S_r \to \forall$ we denote all repository element mappings that have S_r as the set of source elements. For example, let us consider a repository containing two instances of the element mapping: $A \to B$ and one instance of the element mapping $A \to C, D$. In total, there are 3 element mapping with the A set of elements as a source. Therefore, the initial probability that the element set A should be mapped onto B is $W_{A \to B} = \frac{2}{3} \approx 0.67$ and that A should be mapped onto C, D is $W_{A \to C,D} = \frac{1}{3} \approx 0.33$.

In the second sub-phase of the algorithm, user-selected elements are matched against the elements from the repository element mappings. The element comparison is done using an implemented matching algorithm. For example, if a matching algorithm only considers isolated element information, it can be one of the existing string-comparison algorithms such are Levensthein [123] and Jaro-Winkler [198] algorithms. Each pair of elements can be compared with an arbitrary number of matching algorithms. Similarities calculated by different matching algorithms can be combined into a single value by weighted multiplication of produced values. The weights are chosen globally by a user, in the tool settings, and assigned to all matching algorithms. Therefore, the element similarity is calculated as:

$$S_{E,E_r} = \frac{\sum_{i=1}^{n} (S_{E,E_r,C_i} \cdot W_{C_i})}{n}$$

S_{E,E_r} represents the similarity of the selected element (E) and a repository element (E_r). With S_{E,E_r,C_i} we denote the similarity of elements E and E_r calculated by the matching algorithm C_i. Matching algorithms produce a normalized similarity that fits the $[0, 1]$ interval. Additionally,

W_{C_i} is the weight assigned to each matching algorithm by a user and it has a value in the same interval. The sum of all calculated similarities is divided by the number of matching algorithms (n) in order for the final similarity to be also normalized to fit the same interval.

In order to calculate a probability of a repository element mapping being an appropriate candidate for reuse, similarities between all repository elements and user-selected elements must be calculated and combined into a single number specific for the element mapping. This is calculated as follows:

$$P_{S_r \to T_r} = \left(\frac{\sum_{i=1}^{n} S_{E_{s_i}, E_{rs_i}} + \sum_{i=1}^{k} S_{E_{t_i}, E_{rt_i}}}{n+k}\right) \cdot W_{S_r \to T_r}$$

$P_{S_r \to T_r}$ represents the probability of a element mapping $S_r \to T_r$ being a candidate for reuse. With E_{s_i} we represent a selected source element, while with E_{rs_i} we denote a source element of a repository element mapping. $S_{E_{s_i}, E_{rs_i}}$ represents a similarity between aforementioned source elements. Similarly, $S_{E_{t_i}, E_{rt_i}}$ represents the similarity between a selected target element (E_{t_i}) and a target element of a repository element mapping (E_{rt_i}). Both user-selected element collection and repository element collections are ordered in the same way and comprise the same number of source elements (n) and target elements (k). $W_{S_r \to T_r}$ is a weight factor calculated in the first sub-phase of the algorithm.

We should note here that the collection of user-selected elements may contain zero or more source elements and zero or more target elements. If the user initiated the algorithm without selecting any elements, the algorithm will search for the element mapping candidates containing any element from a source or target generic element tree. If a user selects one or more source elements, the reuse algorithm considers only these elements instead of all generic tree elements. In the case when, for example, all selected source elements correspond only to a subset of a repository source elements, other element mapping source elements must be also considered. They are compared to the rest of the unselected generic source tree elements to find a match. Only when a match is found for all of these other rule elements, it can be considered as a candidate. This is due to the fact that we consider a rule to be an atomic semantic unit that is either considered for reuse with all of its elements, or completely ignored. We do not consider rules with just a subset of its elements. The algorithm works in a similar way when the user-selected elements collection comprises zero or more target elements.

In the case where a collection of selected source elements has fewer elements than n or a collection of selected target elements has fewer elements than k, then the following formula may be used to calculate the rule probability for reuse:

$$P_{S_r \to T_r} = \left(\frac{\sum_{i=1}^{n} S_{E_{s_i}, E_{rs_i}} + \sum_{i=n}^{m} S_{E_{gst_i}, E_{rs_i}} + \sum_{i=1}^{k} S_{E_{t_i}, E_{rt_i}} + \sum_{i=k}^{l} S_{E_{gtt_i}, E_{rt_i}}}{n+m+k+l}\right) \cdot W_{S_r \to T_r}$$

Two new segments are added to this formula. The $\sum_{i=n}^{m} S_{E_{gts_i}, E_{rs_i}}$ segment represents the calculation of similarities between repository elements that are not paired with any of user-selected elements (E_{rs_i}) and one of the elements from the generic element source tree (E_{gts_i}). The number of repository elements not paired with the selected source elements is denoted with m. An element from the generic source tree is chosen to have the maximum similarity with the element E_{rs_i}. This maximum similarity must be larger than a user-defined threshold. Analogously, $\sum_{i=k}^{l} S_{E_{gtt_i}, E_{rt_i}}$ segment represents a calculation of similarities of the unmatched target elements of the repository element mapping. The number of unpaired repository target elements is denoted with l.

5.4.3 *Alignment Algorithm*

Alignment algorithm takes selected schema elements as an input, invokes a matching algorithm for each pair of selected source and target elements, and calculates a probability for a mapping between each pair of elements. Just like the reuse algorithm, the alignment algorithm is invoked as a second step of an automation process which is presented in the right side of Figure 23. Same matching algorithms can be used to calculate element similarity. However, unlike the reuse algorithm, alignment algorithm does not take into consideration any past knowledge or repositories with stored element mappings. In the right side of Figure 23 the only step of alignment algorithm is presented.

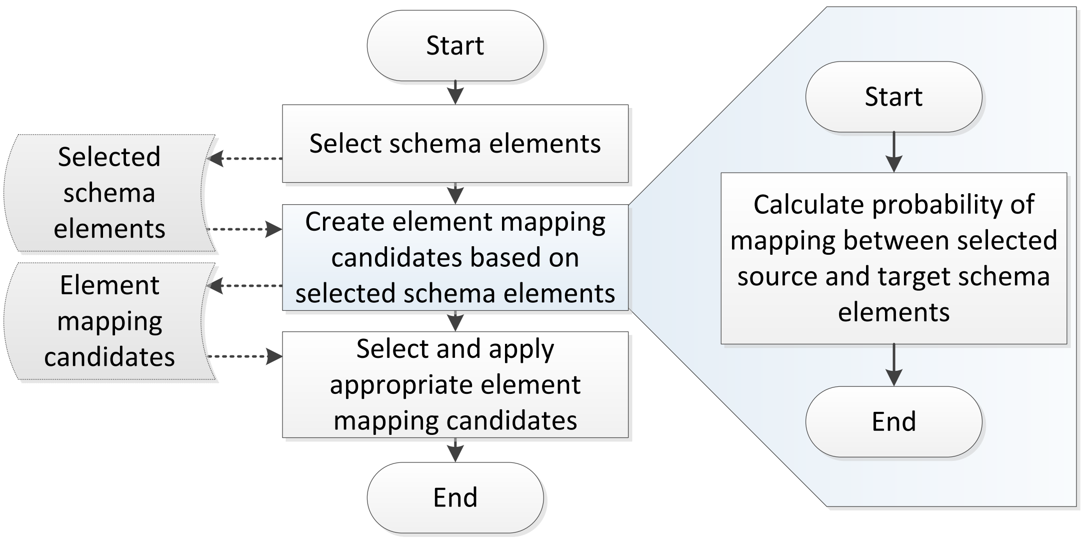

Start
Select schema elements
Selected schema elements
Create element mapping candidates based on selected schema elements
Element mapping candidates
Select and apply appropriate element mapping candidates
End
Start
Calculate probability of mapping between selected source and target schema elements
End

Figure 23: The automation process (left) and the alignment algorithm (right)

The probability of a mapping between a pair of elements being a good fit for the current mapping scenario is calculated according to the following equation:

$$P_{E_s \to E_t} = \frac{\sum_{i=1}^{n} (S_{E_s,E_t,C_i} \cdot W_{C_i})}{n}$$

$P_{E_s \to E_t}$ represents the probability that an element mapping between a selected source element (E_s) and a selected target element (E_t) should be considered as a candidate. With S_{E_s,E_t,C_i} we denote the similarity of elements E_s and E_t calculated by the matching algorithm C_i. Matching algorithms produce a normalized similarity that fits the $[0, 1]$ interval. Additionally, W_{C_i} is the weight assigned to each matching algorithm by a user and it has a value in the same interval. The sum of all calculated similarities is divided by the number of applied matching algorithms (n) in order for the final similarity to be also normalized to fit the same interval.

5.5 ANYMAP TOOL

In order to be used in industry, each integration approach needs to have an appropriate tooling support. Tools need to support all the principles and steps of the approach and provide a suitable set of interaction elements for user. As each tool vendor usually creates a tool that supports their approach, in order to support our approach presented in Section 5.2 we have developed an easily extensible integration tool, named AnyMap. The architecture of the tool is presented in Figure 24.

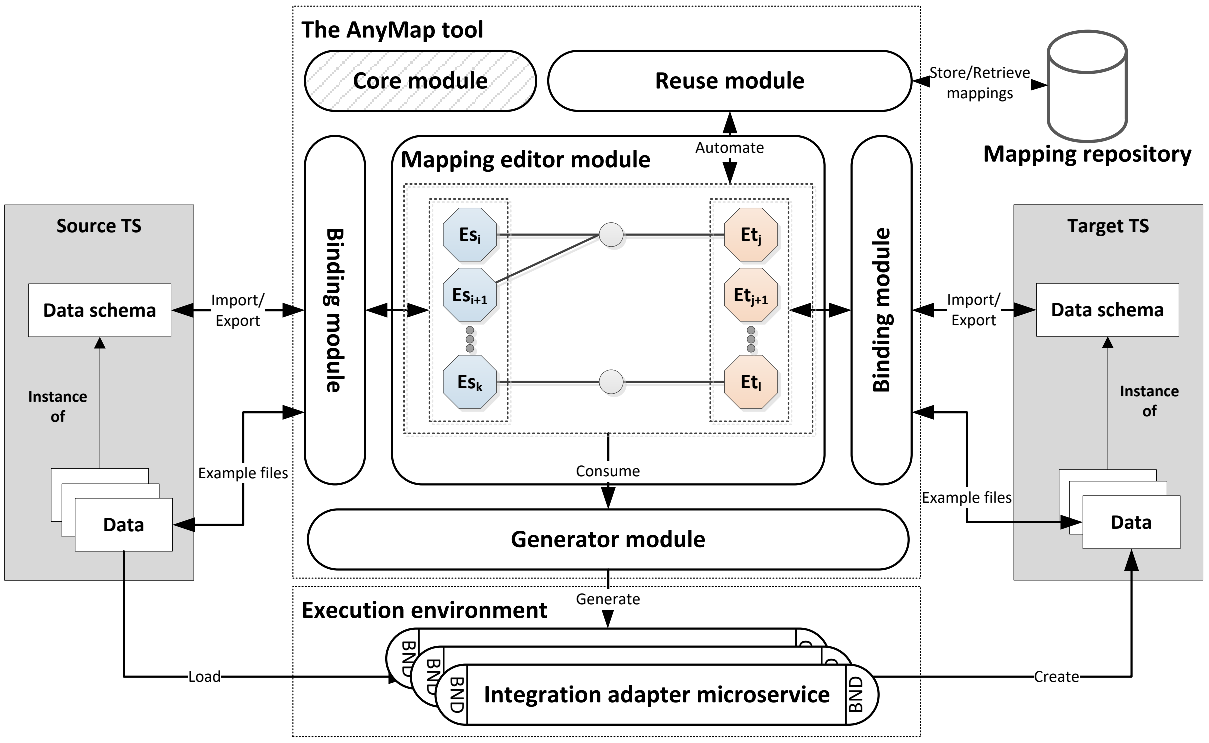


Figure 24: Architecture of the AnyMap tool

There are multiple alternative approaches to building an integration tool, but the most notable ones are: (i) developing a tool from scratch, (ii) adapting an existing open-source integration tool by adding or modifying its internal mechanism, or (iii) developing a tool in a form of plug-ins for an extensible IDE. The first approach is most flexible one as a developer has the largest degree of freedom to choose implementation technologies, make architectural decisions, and make arbitrary customizations of the tool. However, this approach requires more time to build a tool and therefore it is the least appropriate for building tool prototypes. On the other hand, by utilizing an existing integration tool and its internal mechanisms, a development time is significantly shortened and tool prototypes can be developed more easily in shorter iterations. Drawbacks of such an approach include limitations such as the need to follow a predefined tool architecture, design patterns, and programming languages used by the tool developers. One of the examples of such an approach may be found in [190], in which the authors extend the class hierarchy of the Talend Open Studio in order to add an ontology-based integration approach to the tool.

The third approach to building an integration tool can be placed somewhere in between the first two approaches by several criteria. Although a developer extends an existing environment by adding new plug-ins, the environment is used just as a shell that supports basic functionality for interaction with users. Therefore, the developer can focus just on implementing an integration-specific functionality. Developers need to follow design guidelines imposed by the environment vendor and therefore they have less freedom compared to the first approach but are less limited in comparison to the third approach. Also, from the development time standpoint, although third approach stands between the previous two approaches, it is closer to the second one as the most of the repetitive, user-interaction functionality is already provided. In our survey, presented in Chapter 4, we have examined several tools build this way: Open Mapping Software, Vorto, MuleSoft Anypoint Studio, Clover ETL, and Talend Open Studio. All of these tools extended the Eclipse [4] IDE.

Eclipse IDE is a widely-used IDE by Java developers for both web and desktop applications. With its time-tested extensible architecture, it represents a popular choice for the development of tool prototypes as it allows fast and agile development of plug-ins that extend its functionality. For all of the aforementioned reasons, we have developed the AnyMap tool as a set of plug-ins for the Eclipse IDE. All plug-ins are implemented in Java [5] and Xtend [6] programming languages.

The AnyMap tool is implemented as a set of five distinct modules. The AnyMap tool represents the main interaction point with a user and comprises all activities from reading and parsing data schemas, specifying the mappings, to generating the integration adapter. Each of AnyMap modules comprises one or more plug-ins that implement the same interface defined in the core module of the tool. New plug-ins may be easily added to a module by implementing the appropriate interface from the core module and registering their execution with the Eclipse runtime engine. In Section 5.5.1, Section 5.5.2, Section 5.5.3, Section 5.5.4, and Section 5.5.5 we give a short overview of each module together with the main decisions concerning the architecture design and the technologies used to build the modules.

In addition to the AnyMap tool, we have also developed an execution environment to support execution of generated adapters in a scalable and transparent way. Although a user may provide different generators for the AnyMap tool in order to generate adapters for different execution environments, for the purpose of this thesis we have implemented a framework based on the microservice architecture. We chose such an architecture in order to show that the integration adapters can be generated as stateless code components, similar to the AWS Lambda [7], which can be instantiated on-demand depending on the frequency in which the input data is received. The execution environment is described in more detail in Section 5.5.6.

5.5.1 *Core Module*

The Core module comprises essential components which are used throughout the rest of the tool modules. These core components are developed in a way to provide only the most basic functionality and to allow easy extension of other modules. The Core module contains concepts of the mapping language, expression language, and interfaces for the implementation of binders and generators. All interfaces, which need to be implemented in order to extend one of the other modules, are a part of the Core module.

All of the concepts presented in the Figure 20 are implemented in the Java programming language and are part of the Core module. They represent the abstract syntax of the mapping language. For each of these concepts, as a part of the Mapping module, concrete syntax or syntaxes are developed in order to provide adapter developers the most appropriate visual components for their tasks.

4 `https://eclipse.org/ide/`
5 `https://www.java.com/`
6 `http://www.eclipse.org/xtend/`
7 `https://aws.amazon.com/documentation/lambda/`

To support the binder development, the Core module contains two interfaces: *ISchemaBinding* and *IDataBinding*. These interfaces provide the appropriate methods for importing and exporting data schemas and data respectively. Each new binder that is to be developed for a specific TS must implement all the methods from these interfaces. The *IRuntime* interface contains all methods that an adapter must implement in order to read data from original data files and to execute generated transformation rules. In addition to binder-related interfaces, the Core module also contains multiple interfaces for implementing code generators.

5.5.2 *Binding Module*

The Binding module contains plug-ins which represent TS binders used for importing and exporting data and data schemas of a specific TS. For each new TS that needs to be supported by the AnyMap tool, a new binder must be developed in a form of a new Eclipse plug-in. Binder core functionality and algorithms for loading data schemas and reading data files are developed by implementing the *ISchemaBinding* and *IDataBinding* interfaces from the Core module. In addition to the core functionality, each binder plug-in comprises appropriate GUI that provides user interaction with binders. For example, while importing a new file from a TS, a user can go through a GUI wizard and set up the binder parameters. These GUI elements are registered with the Eclipse environment as GUI contributions in order for a user to be able to access the binder by interacting with the main tool interface.

The main task of each binder is to allow importing of data schemas and their transformation to the generic tree structure on top of which the mappings are specified. Each schema concept is transformed to an instance of the *Element* concept (see Section 5.3.2). If imported data has no schema defined, it is up to the binder to extract schema specification and make schema element representation. This schema extraction process can be automatic, i. e., embedded in the binder algorithm, or manual, where a user goes through several pages of the GUI wizard to specify the appropriate schema document. Therefore, each binder is built to support all the process steps presented in Figure 17.

In addition to the transformation of schemas to generic element trees, a binder provides functionality for storing the reverse links from generically represented schema elements to the original schema elements. This is a mandatory process as the generated adapter has to execute transformations of data which conforms to original data schema while the transformations, i. e., mappings, are specified on top of the generically represented schema elements. Therefore, based on these reverse links (see the *binding* attribute of the *Element* concept in Figure 20), a binder is able to read the source data by finding appropriate schema elements to which the data conforms in the TS. In the process of writing data, reverse links are used to write the target data formatted in the appropriate way so as to ensure that a target data file conforms to a target data schema.

5.5.3 *Mapping Editor Module*

The Mapping Editor module is the main interaction point between a user and the AnyMap tool. This module provides the graphical concrete syntax for the mapping language, textual concrete syntax for the expression language, and all necessary GUI classes (event listeners, commands, menu items, etc.) needed to provide interaction between the user and the tool. Furthermore, this module serves as a central registration point of all other plug-ins that contribute to the GUI of the AnyMap tool.

In our survey of the integration tools we have identified several different types of concrete syntaxes for the mapping language: (i) graphical concrete syntax, comprising nodes and lines, (ii) textual concrete syntax, similarly to common programming languages it comprises text literals used for the definition of transformation rules, (iii) tabular concrete syntax, which represents both schema elements and mappings as cell values of a mapping table, and (iv) configuration-based syntax, in which a user specifies a mapping through a series of GUI dialogs and data input fields. Our opinion is that the graphical concrete syntax is the most appropriate one for the integration domain in which we are interested. Such a syntax enables a user to have the best overview of the entire mapping specification and in our opinion it is easiest to learn and to comprehend. A drawback of the graphical concrete syntax is that the diagram may get overcrowded when a lot of mappings is created. However, we feel that the benefits of such a syntax overweight its drawbacks.

Currently, the Mapping Editor module implements a graphical concrete syntax of the mapping language. The graphical concrete syntax of the mapping language is created using the Standard Widget Toolkit (SWT) [8]. We chose SWT as it is the preferred way to contribute to the Eclipse IDE that we are expanding. The modularity of the solution allows for the new concrete syntax to be provided without changing any other modules of the tool. For each concept in the abstract syntax (c.a. Figure 20) a new shape is created. For each shape, GUI handling mechanisms have are also added in order to support the interaction between a user and the tool. As the mapping is serialized in terms of the abstract syntax concepts and their instances, it is possible to have multiple concrete syntaxes on top of a single abstract syntax and to use them simultaneously for the mapping specification.

In regard to the expression language, most of the surveyed tools had either textual or graphical concrete syntax. As we decided to use Java as our expression language, our tool supports the textual concrete syntax for the expression language. We have restricted the possible number of functions that can be used for creation of expressions. Each expression can be only defined on top of the object which is an instance of the *Value<T>* class presented in Figure 20.

5.5.4 *Reuse Module*

The Reuse module comprises multiple plug-ins that constitute the automation engine and reuse and alignment algorithms. The automation engine implements reuse and alignment algorithms. The engine is in charge of reading and writing to a reuse repository and coordinating the calculation of element mapping candidates. As an input, automation engine takes source and target generic tree elements selected by a user. In the current implementation, if the user has not selected any elements, automation engine takes into the consideration all the elements from the appropriate generic element tree.

Following the element selection, the user can choose one or more reuse and alignment algorithms to be applied in the current integration scenario. If a reuse algorithm is selected, the address of an appropriate reuse repository needs to be provided. Reuse and alignment plug-ins are developed to facilitate calculation of similarities between schema elements based on different reuse or alignment algorithms. These algorithms are described in Section 5.4.2 and Section 5.4.3. As the AnyMap tool can be used to create mappings between any two TSs, we can often rely only on isolated mapping information as we do not know mapped TSs in advance. Therefore, the matching algorithms implemented in the AnyMap tool belong to the first category of matching algorithms from the Section 5.4.2, which use only isolated element information to calculate probability of two or more schema elements are a good match.

8 https://www.eclipse.org/swt/

As an output of the automation engine a list of element mapping candidates is provided. A user may review all the candidates with their probabilities and select the appropriate element candidates that are the most appropriate for the current integration scenario. After the selection the AnyMap tool automatically applies selected element mappings to the current mapping specification.

5.5.5 *Generator Module*

The Generator module comprises plug-ins for the generation of executable transformations for a desired execution environment. The task of each generator plug-in is to parse a mapping file, extract necessary mappings, and generate executable transformation code based on these mappings. Each plug-in is built so as to extend the tool capabilities to generate code for a different execution environment. Currently, we have only implemented a generator for our custom execution environment that is presented in Section 5.5.6.

Each generator is implemented in the Xtend programming language which compiles to Java and is then executes on the Java Virtual Machine (JVM). The Xtend is a dialect of Java that makes the development of code generators easy by interleaving template based-language with the dynamically typed expressions for parsing and preparing data used in the templates. All Xtend templates and parsing statements are compiled to readable Java 5 expressions. Therefore, these generators can be easily packaged as Eclipse plug-ins and executed from the Eclipse shell which is the base of the AnyMap tool.

In addition to filling the templates with data parsed from the mapping files, generators need to package necessary libraries and translate expressions in order to generate fully executable code for a target execution environment. In order for an adapter to be able to read the source data file and to write to a target file, binders need to be packaged together with the adapter. Right now, we only support generation of adapters that are executed on the JVM as binders are written in Java and can only be executed on JVM. In order to support generation of adapters in other programming languages, required binders need to be either written in the desired language (e. g., C#) or cross-compiled to that language using one of the existing cross-compilers. Similarly, expressions language statements from the mapping file need to be translated to a programming language of the execution environment. Currently, as expression language statements are written in Java, we can easily copy them to the adapter code as it is also written in Java.

5.5.6 *Execution Environment*

In our integration tool survey, we have noticed that most of the tools either provide built-in execution of the specified mappings or generate adapters that can be run in a single-machine non-scalable environment. It is up to the user to support scaling of such adapters in the distributed environment. Another noticeable characteristic of adapters is that they usually do not have any internal state storage. To be more precise, adapters receive data, transform it, and provide transformed data as the output. As such, adapters do not need any internal data storage to store special information about the process.

In today's world, where data is produced with high frequency and in high volumes, single-machine adapters may represent a performance bottleneck. Two common problems may arise in such execution environments: (i) the problem of machine malfunctioning when the integration adapter is not accessible by devices that send data, and (ii) the problem when there are too many devices sending data at the same time. These problems may lead to creation of message queues which may become too large due to the frequency in which the data is produced. If a message queue becomes too large data can be lost. In industrial settings, where losing data may influence the production process, this is not acceptable. Integration components must be fault tolerant, constantly available, easily scalable, and they must run on limited resources such are the ones provided by industrial computers.

As an adapter does not have to use any data storage to store transformation states, one possible candidate for the execution environment is Amazon Web Services (AWS) Lambda [9]. AWS Lambda enables zero-configuration execution of Java programs (among other programming languages) in a distributed environment where a company pays only for the computing power used when the program is executed. Therefore, AWS Lambda programs are executed when a data file arrives and after the transformation they can be shutdown. Although the AWS Lambda is the perfect fit for the execution environment, not knowing where the data is held and transformed may lead to some serious security issues and considerations for companies. Therefore, we have implemented a microservice-based solution inspired by AWS Lambda which can be deployed on company's premises.

The execution environment comprises many microservices [150] which are autonomous execution units usually created to perform a single task. The advantage of microservice-based application over the monolithic application, where the whole application is tightly coupled, is the possibility to scale and change microservices independently as long as the message flow between them is uninterrupted. However, the main drawback of such a solution is that microservices need an accompanying infrastructure which facilitates exchange of messages and communication with the user.

In Figure 25 we present the architecture of our execution environment. There are four types of microservices, where *Gateway*, *Discovery* and *Load Balancer and Circuit Breaker* microservices are a part of the communication infrastructure, while the *Transformation* microservice is generated from the AnyMap tool based on the mapping specification. All microservices are implemented by using the Spring framework [10], to be more precise the Spring Cloud library [11].

The *Discovery* microservice serves as a central registry of microservices. Each microservice that is run must first register itself with the Discovery microservice. This microservice is implemented using Eureka service from the Spring Cloud library.

The *Transformation* microservice is generated from the mapping specification by the AnyMap tool. It implements Representational State Transfer (REST) API that enables sensors and devices to send data over The Hypertext Transfer Protocol (HTTP). This service can be run once or multiple times, on a single computer or on a cluster, depending of the frequency of incoming data and the speed by which data needs to be transformed. If multiple microservices are run on a single machine, each *Transformation* microservice instance must listen on a different port.

In order to alleviate the problem in which devices must know the fully qualified addresses of *Transformation* microservices, a single point of entry and a load balancer must be implemented. A single point of data entry is the *Gateway* microservice implemented using the Zuul service from the Spring Cloud library. Each device or a sensor sends data to the *Gateway* service and it passes the data to the *Load Balancer and Circuit Breaker* microservice which is implemented by using Ribbon and Hystrix services from the Spring Cloud library. The Ribbon-based load balancing microservice receives the file from Gateway and chooses the appropriate *Transformation* microservice to which the file is then routed. Multiple load balancing algorithms can be implemented but we chose the round-robin algorithm in which the data file is sent to each *Transformation* microservice in equal portions and in circular order. In the case

9 https://aws.amazon.com/lambda/
10 https://spring.io/
11 https://cloud.spring.io/

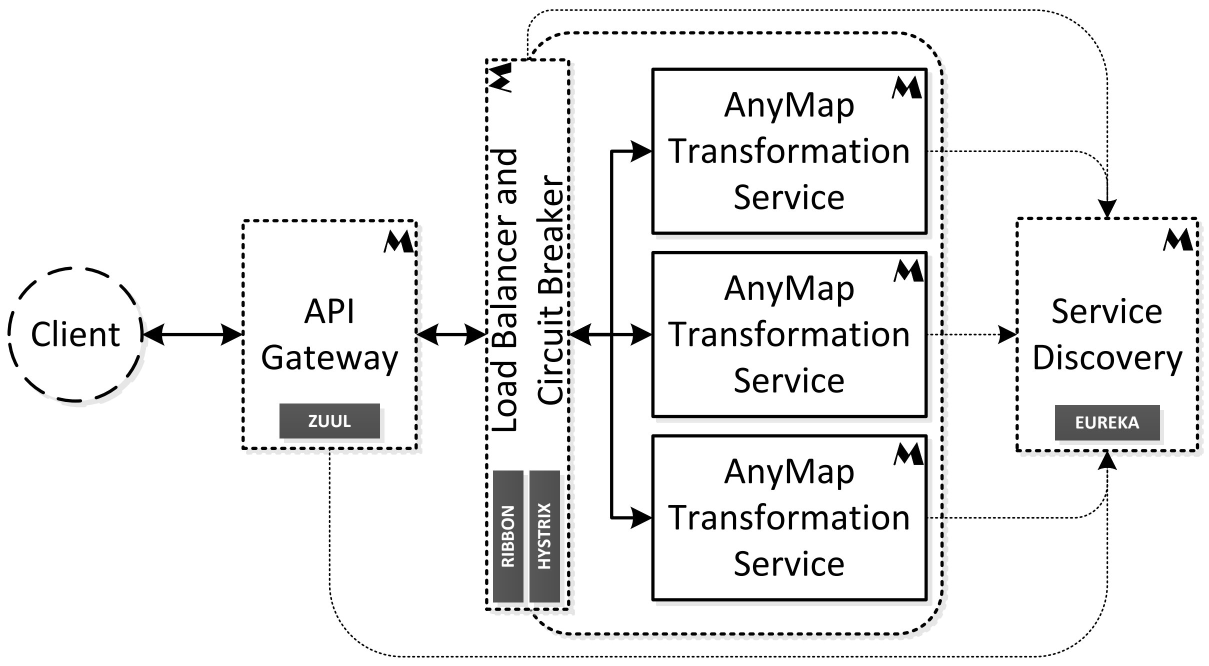

Client
API Gateway
ZUUL
Load Balancer and Circuit Breaker
RIBBON
HYSTRIX
AnyMap Transformation Service
AnyMap Transformation Service
AnyMap Transformation Service
Service Discovery
EUREKA

Figure 25: Architecture of the execution environment

where a data file cannot be passed to any of the *Transformation* microservices, Hystrix-based circuit breaker executes a default error reporting method and resumes the normal operation of the whole system.

Any number of *Transformation* microservice instances can be started or shutdown on-the-fly. This does not require introduction of changes to the whole execution environment. Each time a new *Transformation* microservice instance is started, it is registered with the *Discovery* microservice. Therefore, the *Load Balancer* will recognize that a new instance is registered and will include it in the execution process without the need for manual configuration of other microservices.

5.6 SUMMARY

In this chapter we have presented our vision of the approach to the specification of integration adapters, its place in the generic integration process, and the AnyMap tool that provides the means to utilize the approach.

Our first goal behind creating such an approach is to provide an integration mechanism which would allow users to integrate arbitrary technical spaces (TSs) in a uniform way. This would allow users to learn the approach and become accustomed to the tooling support once and afterward just spend time on performing the integration tasks without learning the implementation and serialization details of each integrated TS. In order to achieve this goal, our approach includes steps for translating the original TS data schemas that are being integrated to a generic schema representation. We decided to flatten a graph-like generic schema representation and use a tree-like structure in order to facilitate easier, more compact, and cleaner adapter specification.

With this approach, our aim is also to provide users with an adapter specification mechanism which is at the appropriate abstraction level. Such an approach would enable users to focus on the integration task and not on a programming language and structure of the adapter code. This is achieved by allowing a user to specify the adapter, i. e., its transformation rules, at the level of data schemas using a custom domain-specific mapping language. As concepts of such a language are tailored to fit the industrial integration domain and non-programmers' needs and skills, by using these concepts a user does not need to be experienced in any of the contemporary programming languages. The domain-specific mapping language allows a user to specify correspondences between appropriate source and target schema elements using a language with a combination of graphical and textual concrete syntaxes. Based on these correspondences, an executable integration adapter can be automatically generated and prepared for the execution on a desired platform.

Maybe the most important goal behind the development of the approach is the automation of adapter development. Although by increasing the abstraction and by providing a universal tool for integrating any TS the speed of adapter development should be increased, arguably the largest increase in speed should be a consequence of the increased development automation. By utilizing reuse and alignment algorithms in our approach, we aim to alleviate a user of a tedious, error-prone, and time-consuming process of creating repetitive mapping (transformation) rules. This is most evident when specifying multiple adapters in the same integration scenario where multiple adapters are specified between same two or more TSs. In such a scenario, the change in device or information system configuration introduces just minor changes to the data schema. Therefore, if these changes are predictive and follow some machine recognizable patterns it should be even possible to fully automate the specification process. However, in most use cases it is possible to achieve semi-automation at most. In this way, the automation algorithms of our approach provide a list of element mapping candidates that need to be reviewed by a user. After the user selects appropriate candidates they are automatically applied to the current integration context. Our approach supports both full and semi-automation of specification processes.

All of the approach steps are supported by the AnyMap tool. It is developed as a set of plug-ins for the Eclipse IDE and it is easily extended to allow further customization . These plug-ins are grouped together in five modules: Core, Binding, Mapping Editor, Reuse, and Generator modules. In regard to the survey characteristics presented in Section 4.1, the AnyMap tool can be characterized by the values presented in Table 4.

One of the possible future research directions is to implement more advanced reuse and alignment algorithms. Currently, automation is based on simple matching algorithms that calculates element mapping candidate probability based on isolated element information. Multiple other algorithms can be implemented including the ones based on element semantic and element structure. Another option that needs to be examined is to implement a recommender system [165] which would create a matrix of elements and previously stored repository rules and recommend best fitting candidates.

Also, new binders and code generators are needed in order to provide more domain coverage for the AnyMap tool. By implementing these elements, we would acquire more use cases on which to evaluate and further validate both the approach and the tool. In addition to binders and generators, new concrete syntaxes could lead to additional increase in the mapping specification speed. By allowing users to simultaneously use multiple concrete syntaxes, they could switch between them using the benefits of each syntax in order to further increase the speed of adapter development. For example, configuration-based syntax can be used to represent a mandatory sequence of steps and it is the only one that can enforce users to do them in the right order. Afterward, tabular representation can be used when a large number of mappings are present as it is the most concise syntax. Graphical syntax gives the best overview in case of small to medium sized mappings, while textual syntax, if done right, could be used by integration experts so as to increase the specification speed even further. Our

Characteristic	**AnyMap**
Distribution	Desktop application, freeware and commercial license
Recentness	2017
Domain: application	Data integration in the industrial context
Mapping approach	Direct approach to mapping specification
Language: mapping	Graphical mapping language
Language: expression	Java-based textual expression language
Code generation and execution	Externally executed adapters are generated
Reuse: concepts	*TS specifications*—as binders can be inherited and reused, *functions*—as functions can be packaged as external Java libraries, *mappings*—through automation mechanism of the tool, i. e., element mapping reuse
Extensibility: technical spaces	Any technical space can be integrated by implementing a new binder for it
Extensibility: functionality	New expression language functions (Java libraries) and code generators (tool plug-ins) can be added without changing the tool code-base

Table 4: AnyMap characteristics

future research will encompass finding the best ways to represent mappings in each of these syntaxes while allowing easy and seamless transition between them.

In the next chapter, we will present two case studies from the integration domain in which we have applied our approach and the AnyMap tool.

6 APPLICATION AND ANALYSIS OF THE MAPPING APPROACH

Our integration approach uses three-level technical spaces (cf. Figure 16) which are encountered in a variety of integration use cases. Therefore, a practical applicability of the presented approach is high. During our previous research and the development of the tool, we have implemented multiple use cases which include technical spaces such as: RDBMS, XML, CSV, SECS/GEM, OPC, and EMF. In this chapter we will present two use cases which are, in our opinion, the most suitable for presenting all concepts introduced in the previous chapter.

The approach presented in this thesis can be used in the industrial context as one of the main enablers of the factory automation. Our focus is on the commonly encountered problem of integration between sensor machines and information systems. Although we choose one specific sensor and a specific IS module, conclusions made in this chapter will be valid for the integration between any sensor machine and any information system in the industrial context. The presented use case comprises integration of a sensor that measures different characteristics of semiconductor wafers and an IS module for data visualization. The integration in this use case is performed between CSV and XML TSs. Sensors gather data and send it formatted as a CSV document. The information system visualizes data using the JSChart library and expects data to be formatted according to a predefined XML schema.

This example is also suitable for presenting the benefits of automation algorithms. The automation algorithm can increase developers' efficiency through decreasing adapter development time as multiple adapters often need to be developed between similar devices. This industrial use case is presented in Section 6.1.

In addition to the industrial application, our approach is applicable to many other, non-industrial software integration domains. One of the notable problems is the interchange of models and meta-models between meta-modeling environments. This is a known issue and it has been discussed by Kern et al. in [104, 107]. Although meta-modeling environments have the export and import mechanisms, they usually focus on just a small number of serialization formats. If two meta-modeling environments do not support the same serialization format or do not exchange semantics together with a model it is impossible to use these environments in a team or to migrate from one tool to another. We will use our approach combined with the M3-Level-Based Bridges (M3B) approach introduced by Kern in [106] as to provide meta-modeling environments with an external model interchange functionality. We use M3B to transform meta-models and models into the EMF space, which in this case serves as a mediatory technical space. The integration is then performed in our AnyMap tool as a mapping between two different structures of the same EMF TS. This example is chosen as it represent an integration scenario which is on a higher abstraction level than the previous one. Furthermore, it shows that our approach and the tool are not isolated or closed systems. They can be used in collaboration with other tools in order to provide better solution for a given integration problem. This use case is presented in Section 6.2.

It is worth mentioning that model-to-model transformations presented in [53] served as the initial use case for the development of the AnyMap tool. In the paper, we presented manually-developed transformations between different data schemas in the EMF technical space. Transformations were developed by using ATL and ETL languages and are a part of the Multi-Paradigm Information System Modeling Tool. The main purpose of the developed transformations was to provide means to transform different data and database models such as Form Type data model, Extended Entity-Relationship data model, and relational data model.

Aforementioned transformations were also specified using an early version of the AnyMap tool in addition to a manual specification. The strict separation of meta-models (schemas) and models (data) found in that example provided us with a perfect environment for testing our ideas and initial implementations of the graphical mapping language. Some of the created components are still a part of the tool such are parts of the graphical language, ETL code generators that are used in Section 6.2, and EMF binders. However, majority of the components suffered a serious redesign and improvements over the time.

The main drawback of transformations presented in [53] is that they can be considered as one-of transformations and as such are quite complex to handle. Therefore, they are not appropriate candidates for a generative approach that heavily relies on a declarative graphical language and reuse. These transformations are considered as one-of transformations because after their specification they are executed in isolation, requiring no additional interaction with other transformations. Furthermore, as these transformations are written at a higher level of abstraction the knowledge they represent is not reusable in other domains and in the same domain there is no need for reuse. Additionally, these transformations cover many fringe use cases that can only be specified in an purely imperative manner. Due to their nature, we have used them only during initial tool development and later switched our focus to other use cases where the reuse of mappings was of more importance.

Finally, in Section 6.3 we provide discussion about the established hypotheses and present the reasons of their confirmation or rejection. Where applicable, we present the limitations of our approach in comparison to hypotheses. We also provide a discussion about the advantages and disadvantages of our approach and the tool in the light of the presented application and previous literature and integration tool surveys.

6.1 DATA INTERCHANGE BETWEEN SENSOR MACHINES AND INFORMATION SYSTEMS

Our approach presented in Chapter 5 could be used in a wide range of machine-to-machine, machine-to-IS, and IS-to-IS integration scenarios. However, we choose a very specific example in order to present all characteristics and implementation details of our approach through an example that is easy to understand and follow. This use case concerns measuring various physical properties of semiconductor wafers during the production process. Such measurements are important to ensure the quality throughout the entire production process. The measurement (sensor) machines offer different measurement methods such as, grid, profile, or spot measurements. Depending on the selected method, the machine produces different output data. In this case, each machine produces one CSV document per operation containing measured values. For data processing and analysis, the CSV document must be imported into an IS module which can only receive XML documents conforming to a predefined schema. This example has been also used as a base example in our survey of integration tools which results are presented in Chapter 4. In Figure 9 an example CSV document and XML schema documents are presented that are also used in this section. A high level overview of the required transformation rules between the two structures is presented in Figure 10.

Beside the inter-space (technical) heterogeneity between the CSV and XML technical spaces, the import mechanism must overcome the intra-space (functional) heterogeneity as well. The existence of different measurement methods leads to a variability in CSV document structure. Therefore, an IS vendor needs a set of different adapters for the integration of the sensors that use different measuring methods. The manual implementation is in most cases insufficient, time-consuming, costly, and error-prone. Hence, we will use our integration approach in order to ease specification of adapters in presence of the two heterogeneity problems. In Figure 26, we give an overview of the approach steps used in this use case as well as the tool modules that were used.

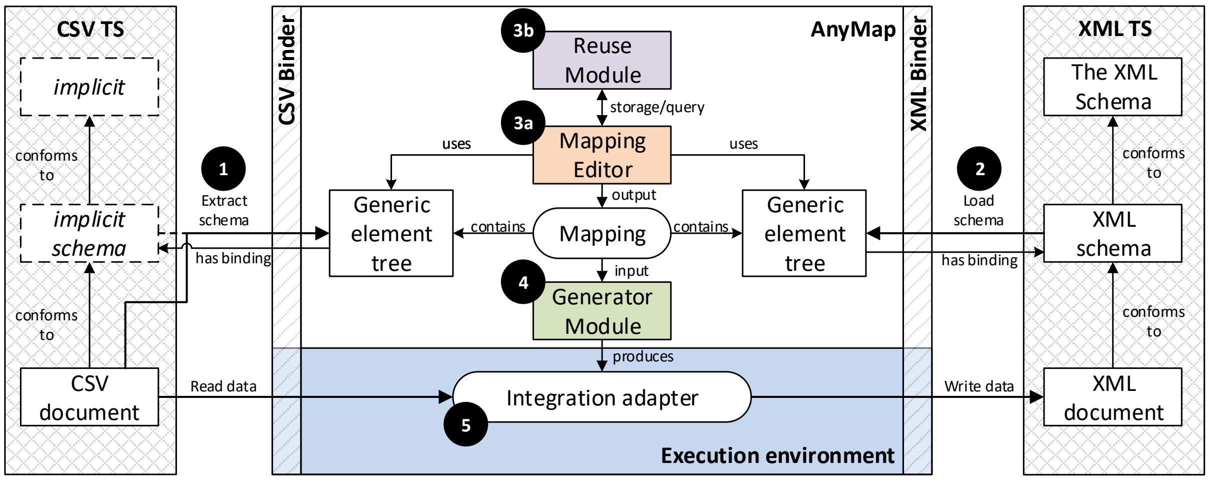


Figure 26: Integrating CSV and XML technical spaces using the AnyMap tool

The five main steps that need to be performed in order to integrate CSV and XML are presented in Figure 26 as black circles containing a number. A detailed explanation of these steps is as follows:

Step 1 *Extract and load the CSV schema*. As the CSV technical space is schema-less i. e., there is no explicit schema definition language. Therefore, schema specification is extracted from an example data file. A developer uses the CSV binder to read the data file and extract schema information from it. Schema is extracted either by reading column names from a header of the file or by manually specifying column names and types.

Step 2 *Load the XML schema*. Constructing a generic element tree from a schema structure in the XML technical space is a straightforward process. Using the XML binder, a developer inputs an existing XML Schema Definition (XSD) document which is then transformed to the generic tree representation.

Step 3 *Create the mapping specification*. Once both schemas are represented in a generic way, the developer specifies a mapping. In Section 6.1.1 the developer specifies the mapping manually by using the *Mapping Editor* module (*Step 3a*). In Section 6.1.2 the developer additionally relies on a repository of previously defined mappings and the *Reuse* module in order to speed up the specification process (*Step 3b*).

Step 4 *Invoke the microservice code generator*. The created mapping specification represents an input to the *Generator* module which is used to generate an executable integration adapter. In this use case, the developer selects the generator for the microservice execution environment (cf. Section 5.5.6).

Step 5 *Execute the integration adapter*. The output of the generation process is an integration adapter that can be immediately executed in the execution environment. During the execution, the adapter reads the input data by using the appropriate methods of the CSV binder. Read data is then transformed according to the generated transformation rules and written to the target TS. The adapter uses XML binder in order to write data formatted according tho the loaded XML schema.

More detailed explanations of these steps are given in the rest of this section.

In CSV documents presented in this section, we have changed the names of columns and the measured values due to a non-disclosure agreement that we have with the companies that have provided these examples. However, we have done our best to ensure that all the structural characteristics of the CSV document remain the same and that these changes do not influence the integration process. Similar column names remained similar even after the anonymization process. All values were altered in a same way so as to retain the ratio between values of the most important columns. All company related meta-information together with sensitive device data and configuration were removed from participating documents. These removed information were not a part of the data payload but were in the CSV document header.

6.1.1 *Single-Layered CSV Data*

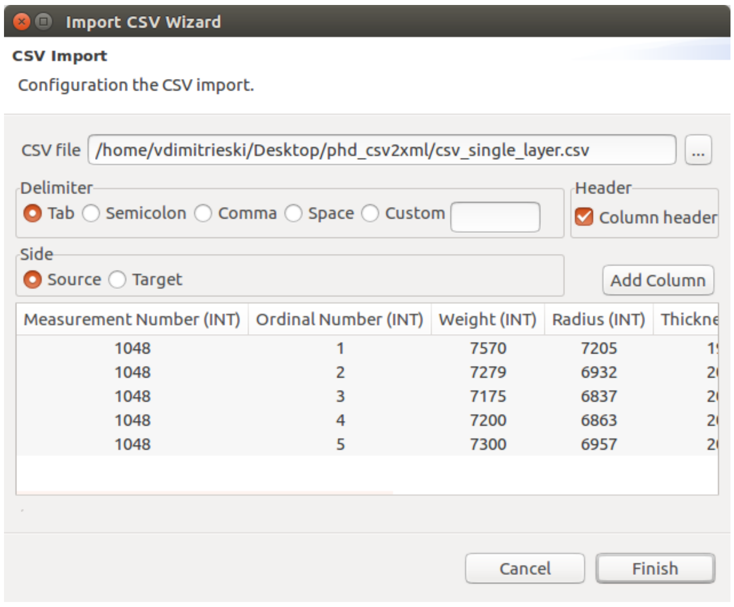
By using one of the measuring methods, a sensor produces a so-called single-layered CSV document. In this file, only one column per measured wafer property is present. For example, in Figure 9 it can be seen that a sensor measured *Weight*, *Radius*, and *Thickness* of a wafer only once per measurement. Each measurement is represented as a single row while each of the measured properties is represented by only one column in the file. In addition to the aforementioned columns, for each measurement a new *Ordinal Number* is assigned. Other columns are not of interest to this use case but they are given so as to keep an entire payload structure intact.

On the other hand, the IS visualizes measurement data in a form of a line graph. The visualization module uses JSChart library to show the data points formatted according to the XSD document presented in Figure 9. Each instance of the *data* element represents a data point to be plotted in the graph with its x-axis value (the *unit* attribute) and the y-axis value (the *value* attribute). Data points are grouped together into data sets (the *dataset* element). The type of the data set (the *type* attribute) represents the way in which it is visualized, while the id (the *id* attribute) provides a unique identification for the data set. In this use case, the *type* of the data set should be set to "line" while the *id* should be equal to the name of the plotted wafer property.

In Figure 10 we present a high-level overview of the transformation rules that need to be implemented in order to transform the single-layered CSV document to the appropriate XML document.

The integration process starts with the import of data schemas into the AnyMap tool. The implemented binders are in charge of reading CSV and XSD files and translating them into the generic element tree which is needed for the creation of mappings. In Figure 27 we present the GUI interface of the CSV binder.

The CSV binder supports schema extraction from CSV data files and manual schema creation if there are no example data files available. A developer needs to provide a path to the example file, select a delimiter, choose whether the file contains a header with column names, and select the mapping side on which the CSV technical space should be included. During this initial process, the effect of each user's input is shown in the dynamic preview of the extracted schema. This dynamic preview can be seen at the bottom of Figure 27. In the figure, the constructed schema is presented together with several data rows so the developer can validate the output. A binder is able to automatically read most of the aforementioned parameters. Additionally, the binder is able to identify column types from the example data files. However, in some cases, column types cannot be inferred automatically or a column name is not present in the CSV document header. In these cases, the binder GUI allows developers to manually specify columns by selecting a column and populating a newly opened dialog. Sometimes, a


Import CSV Wizard

CSV Import

Configuration the CSV import.

CSV file /home/vdimitrieski/Desktop/phd_csv2xml/csv_single_layer.csv ...

Delimiter: [✓ Tab] Semicolon Comma Space Custom

Header: [✓ Column header]

Side: [✓ Source] Target

Add Column

Measurement Number (INT)	Ordinal Number (INT)	Weight (INT)	Radius (INT)	Thickne
1048	1	7570	7205	1
1048	2	7279	6932	2
1048	3	7175	6837	2
1048	4	7200	6863	2
1048	5	7300	6957	2

Cancel Finish

Figure 27: The CSV binder configuration dialog

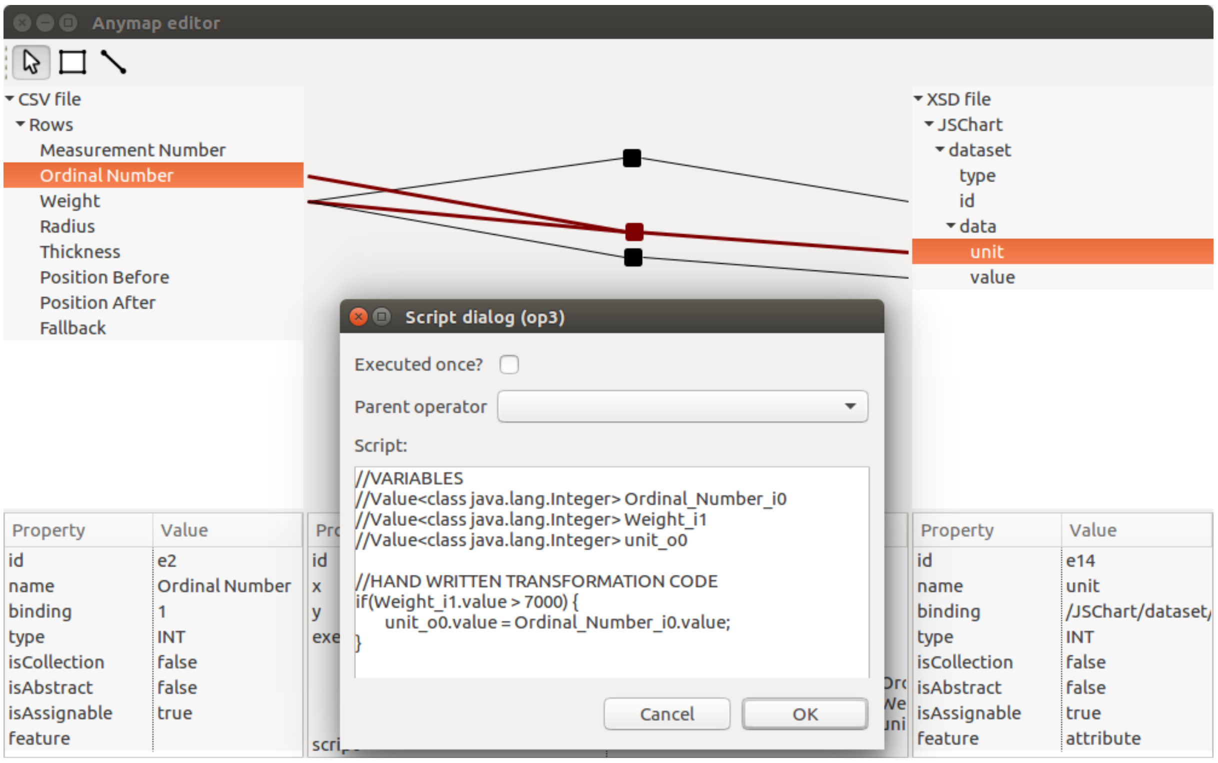
CSV schema must be extended or created from scratch. The binder provides this functionality which is accessed by clicking on the *Add Column* button.

Unlike the CSV binder, the XSD binder is very simple. A developer only provides a path to an XSD document and the mapping side on which this TS participates. In the future we plan to provide binding functionality which is able to infer schema from XML example files. Right now it is possible only to load an existing XSD document into the AnyMap tool.

After importing data schemas from both technical spaces, the tool provides a blank canvas for mapping creation with generic element tree present on both sides of the canvas. In Figure 28 we captured a mapping state from an ongoing mapping specification based on these imported data schemas.

At the left side of Figure 28, an element container named *CSV file* is presented. It contains only one element named *Rows*. The *Rows* element is an abstract element representing data payload of the CSV document. By the notion of payload we denote rows that represent measured values from the SUS. In addition to the payload, a CSV document can have other, top-level meta-attributes that do not represent measured values but information about the protocol, sensor configuration, and manufacturer. Such elements would be created at the same level as the *Rows* element if they are encountered in the document. Each child element of *Rows* represents a single column from the CSV document. These elements are not abstract and they can be used in the mapping specification. Their type is inferred during the binding process and their binding (i. e., reverse link to the original schema element) is in fact an ordinal number of the column they represent. These properties can be seen in Figure 28 in the property view located bellow the generic element tree. Presented property views are displaying properties of the *Ordinal Number* and *unit* elements.

At the right side of Figure 28 we present an element container based on the imported XSD document. The element container representing the XSD document is named *XSD file*. The only child element of the element container is the *JSChart* element which is created from the JSChart root element of the XSD document from Figure 9. All other child elements are created from the XSD sub-elements and their attributes. Binding values are in fact XPath expressions that uniquely identify every element in the XSD document.



Executed once? ☐

Parent operator

Script:

```
//VARIABLES
//Value<class java.lang.Integer> Ordinal_Number_i0
//Value<class java.lang.Integer> Weight_i1
//Value<class java.lang.Integer> unit_o0

//HAND WRITTEN TRANSFORMATION CODE
if(Weight_i1.value > 7000) {
    unit_o0.value = Ordinal_Number_i0.value;
}
```

Property	Value
id	e2
name	Ordinal Number
binding	1
type	INT
isCollection	false
isAbstract	false
isAssignable	true
feature	

Property	Value
id	e14
name	unit
binding	/JSChart/dataset/
type	INT
isCollection	false
isAbstract	false
isAssignable	true
feature	attribute

Figure 28: The mapping specification process

The developer may start to create element mappings only after both generic element representations of both source and target meta-models are created. Each element mapping consists of two components that are specified separately: (i) operators and (ii) links. Operators are created first and are represented as rectangles (the second tool in the AnyMap tool palette). Operators are linked to generic tree elements via links. Links are represented as lines (the third tool in the AnyMap tool palette) and a developer may link tree elements to operators and vice versa. Each link connected to an operator introduces a new variable that can be used in the operator script when writing expressions by means of *Expression language*. The variable name is derived from the element name by adding a single character representing the side of the link in comparison to the operator (i—input, o—output) and an ordinal number of the link at its side of the operator. Examples of link names can be seen in Figure 28 in the opened *Script dialog*.

The *Script dialog* can be opened by double clicking on the operator and it allows specification of the executable script written in our *Expression language*. The first few lines of the script are comments with variable names which provide a good operator overview to developers. For example, the highlighted rule which is marked red on the canvas has two inputs and one output link and therefore has three variables: (i) Ordinal_Number_i0, first input variable corresponding to the *Ordinal Number* element, (ii) Weight_i1, second input variable corresponding to the *Weight* element, and (iii) unit_o0, first output variable corresponding to the *unit* element. Types of these variables are inferred from imported schemas and example data files. The types are then passed instead of the generic type *T* in *Value<T>* (cf. Figure 20). For example, the *unit* element identified in the target XSD schema is of the type INT as it can be seen in the right property view. Based on this information the *unit_o0* variable in the script is created to be of the *Value<class.java.lang.Integer>* type.

By using our Java-based *Expression language* and link variables, a developer may create imperative expressions and further specify transformation rules that could not be expressed just by using the declarative graphical mapping language. An example of the script specification can be seen in the *Script dialog* presented in Figure 28. The rule marked red in the canvas and presented in the script dialog should be executed to set a value of the *unit* attribute in the XML document to the value of the *Ordinal Number* column from the source CSV document. This transformation rule should be executed only when a value of the *Weight* column is greater than 7000 in the same CSV row. This corresponds to the transformation rule presented in Figure 10. All element mappings from this example are presented in the top part of Figure 29.

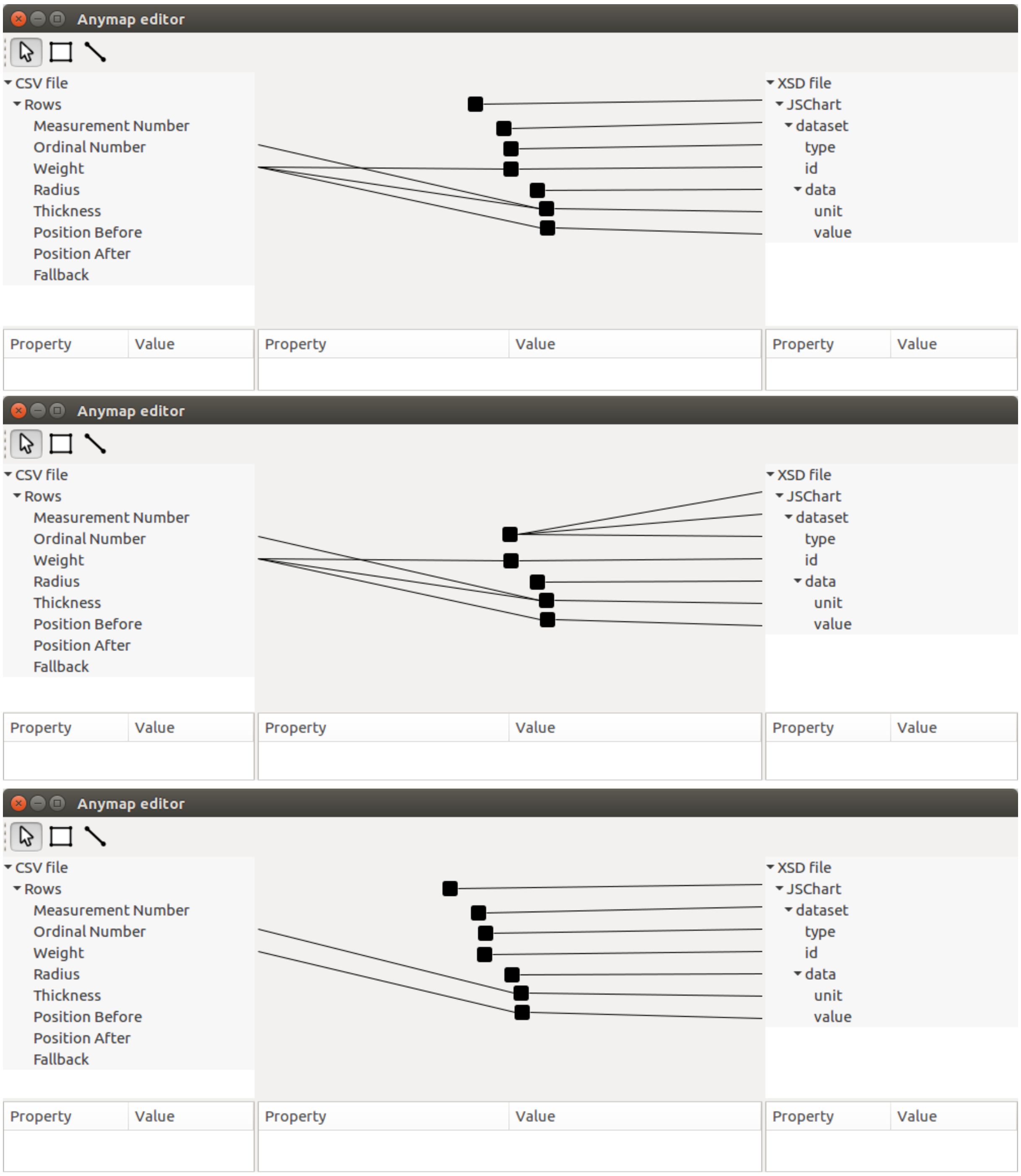


Figure 29: CSV2XML mapping specifications (versions 1–3)

Before presenting all the element mappings, we need to introduce a formula-based representation of element mapping rules to allow easier referencing and providing descriptions in the text. The example mapping from the previous paragraph can also be written in the following way: [$\mathsf{OrdinalNumber} \xrightarrow{Weight>7000} \mathsf{unit}$]. This formula represents an element mapping rule which specifies transformation of element *Ordinal Number* to the element *unit* when the *Weight > 7000* condition has been satisfied. We should note here that the diagrammatic representation differs a bit from the formula-based representation. The diagrammatic representation contains two input links while the formula-based representation has only one input schema element. The reason behind this is that the link to the *Weight* schema element, needed only for the execution condition, is needed in the diagrammatic representation while in the formula-based representation we have a separate place for the condition specification. The previous formula represents a transformation rule that is executed for each piece of data from the source document.

By just writing the name of the source or a target element in a formula we presume that the operation is executed on its value. If the operation requires element meta-data, the name of the appropriate function is written. For example, Weight denotes reading the value of the *Weight* element, while $\mathsf{Weight.elementName}$ denotes reading the name of the element which will return the sting "Weight" in this particular case. For the rules that are executed only once we will use the $\Longrightarrow$ symbol instead of $\longrightarrow$. Constant values are represented inside double quotation marks. Zero-to-Any element mappings are represented with the empty set symbol ($\emptyset$) at the left side. In the case of Many-to-Many element mappings, multiple elements can be found on the source and the target side. The default meaning behind such a formula is that the combination of input element values are combined into a value for the combined target elements. However, in some cases of Many-to-Many element mappings, there is a strict correspondence between some source and target elements. Such complex rules are represented as a list of more simple rules separated by semi-columns. For example: [$\text{"}\mathsf{line}\text{"} \Longrightarrow \mathsf{type}; \emptyset \longrightarrow \mathsf{JSChart}, \mathsf{dataset}$]. Such rules can be separated into independent, atomic rules but in the end it is up to mapping designers to choose the appropriate granularity level for their mappings.

In the *Script dialog*, the developer is also able to specify if an operator is executed only once or multiple times for each input document. This is done by checking or unchecking the *Executed once?* checkbox in the *Script dialog*. Operators can be executed multiple times for each input document in the cases when a single document is divided into smaller data units. By the notion of a data unit we denote an atomic piece of data that is provided as a single input to a transformation system. For example, in the CSV TS each row of the payload can be sent independently and as such transformed to a desired target structure. On the other side, in a large number of use cases, an XML document must be sent as a whole in order to be interpreted and transformed properly.

First three element mappings of the top mapping presented in Figure 29 are the ones that are executed only once for the whole transformation process. In order to specify this, an *Executed once* checkbox should be selected. All other element mappings are executed for each set of values (i. e., a row from a CSV document) that is provided as the adapter input. In order to reference these element mappings in the text more easily, we present them by using the formula-based representation in the lines 1–8 of Listing 1.

Additionally, through the *Parent operator* combobox of the *Script dialog*, the developer is able to select one of the previously defined operators and set it as the parent to the current operator. This is required when the execution of an operator depends on another previously specified operator (parent). By creating operator hierarchies a developer is able to specify groups of operators that are executed in a series.

In addition to the previously described mapping specification, it is possible to create more mappings that will result in the same output. Alternative versions of mappings may be created by grouping simple operators into a more complex one or by splitting a complex operator.

```
1   //Mapping version 1
2   $[\emptyset \Longrightarrow \mathtt{JSChart}]$
3   $[\emptyset \Longrightarrow \mathtt{dataset}]$
4   $[\text{“line”} \Longrightarrow \mathtt{type}]$
5   $[\mathtt{Weight.elementName} \Longrightarrow \mathtt{id}]$
6   $[\emptyset \longrightarrow \mathtt{data}]$
7   $[\mathtt{OrdinalNumber} \xrightarrow{\mathtt{Weight}>7000} \mathtt{unit}]$
8   $[\mathtt{Weight} \xrightarrow{\mathtt{Weight}>7000} \mathtt{value}]$
9
10  //Mapping version 2
11  $[\text{“line”} \Longrightarrow \mathtt{type}; \emptyset \longrightarrow \mathtt{JSChart}, \mathtt{dataset}]$
12  $[\mathtt{Weight.elementName} \Longrightarrow \mathtt{id}]$
13  $[\emptyset \longrightarrow \mathtt{data}]$
14  $[\mathtt{OrdinalNumber} \xrightarrow{\mathtt{Weight}>7000} \mathtt{unit}]$
15  $[\mathtt{Weight} \xrightarrow{\mathtt{Weight}>7000} \mathtt{value}]$
16
17  //Mapping version 3
18  $[\emptyset \Longrightarrow \mathtt{JSChart}]$
19  $[\emptyset \Longrightarrow \mathtt{dataset}]$
20  $[\text{“line”} \Longrightarrow \mathtt{type}]$
21  $[\text{“Weight”} \Longrightarrow \mathtt{id}]$
22  $[\emptyset \longrightarrow \mathtt{data}]$
23  $[\mathtt{OrdinalNumber} \longrightarrow \mathtt{unit}]$
24  $[\mathtt{Weight} \longrightarrow \mathtt{value}]$
```

Listing 1: The formula-based representation of CSV2XML mapping specifications (versions 1–3)

The degree of operator grouping depends on preferences and experience of the developer. A more compact mapping specification than the original one is presented in the middle part of Figure 29. Its formula-based representation is given in Listing 1, lines 10–15.

As it can be seen in the line 11 of Listing 1, previously separate element mappings from lines 2–4 of Listing 1 can be grouped together in a single element mapping. In its expression script the “line” string literal is assigned to the *type* element while *JSChart* and *dataset* elements are created from scratch without a need to handle them specifically in the script. The element mappings can be grouped further to the point where the entire mapping comprises just a single element mapping with a really complex expression script. Although this is a possibility, it resembles a manual specification of the adapter and contradicts the basic motivation of using our approach. The developer of this mapping decided just to group the first three element mappings from the the first mapping version. All of the mapping versions are stored in a mapping repository of the company and this way a variety is introduced to it. Such a variety is useful to provide more data for the reuse algorithm to infer new mappings and to increase the automation of the process. However, some of the introduced variety, especially the one introduced by specifying the same element mapping at different granularity levels, can lead to the less reliable reuse. This will be discussed in more detail in Section 6.1.2.

The third version of the mapping is presented in the bottom part of Figure 29 and lines 17–24 of Listing 1. Unlike the first two versions, the developer decided to introduce two logical changes to the mapping specification one of which may have repercussions to the content of the generated XML document. Instead of assigning the name of the *Weight* element to the value of the *id* element (line 5 in Listing 1), the developer decided to provide the “Weight” string literal as the value of the *id* element (line 21 in Listing 1). Although this is a valid way of specifying the element mapping it can hinder the reuse later in the process. For example, if there is a need to show a JSChart for the *Thickness* measurements, it would be possible to build a generic reuse mechanism that would swap the *Weight* and *Thickness* elements as inputs of this operator. However, it would be hard to create a generic reuse mechanism that would go into every string literal and contextually check if it should remain the same or should it be adapted to a new value.

Furthermore, the developer of the third mapping version omitted the execution condition from the mapping rules in lines 7–8 of Listing 1. Due to this change, the values in the output XML document may be different than the ones generated by executing two previous versions of the mapping. Nevertheless, all generated XML documents have the same structure as they conform to the same XML schema.

The part of the output XML document, with the values generated after executing the first two versions of mappings, is shown in Listing 2. Due to repetitiveness, we have omitted multiple lines from this file. Omitted lines represent data points that have exactly the same structure as data points presented in lines 4–8. The values are generated from the values of the CSV document columns presented in the top part of Figure 9.

```
1  <?xml version="1.0" encoding="UTF-8" standalone="no"?>
2  <JSChart>
3          <dataset id="Weight" type="line">
4                  <data unit="1" value="7570"/>
5                  <data unit="2" value="7279"/>
6                  <data unit="3" value="7175"/>
7                  <data unit="4" value="7200"/>
8                  <data unit="5" value="7300"/>
9                  <!-- omitted due to length -->
10         </dataset>
11 </JSChart>
```

Listing 2: The output XML file

The generated XML document can be used by the information system to render a line chart using the JSChart library. This line chart is presented in Figure 30.

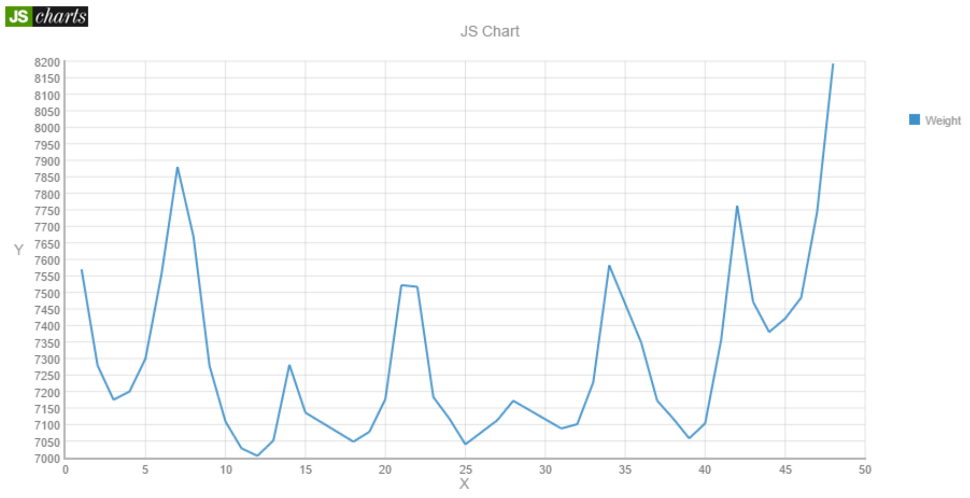
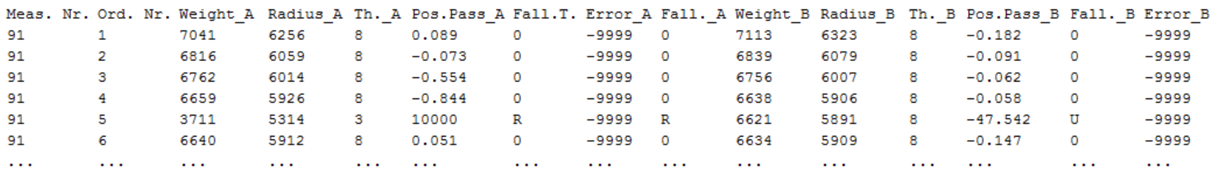


Figure 30: The line chart of the single-layered measurement data

6.1.2 *Double-Layered CSV Data*

By using a second measurement method, a sensor machine measures each wafer physical property twice in the same measurement. As a consequence, the CSV document contains two columns for each measured property. The example of such a document is presented in Figure 31 and we call it a double-layered CSV document. For example, the weight of the wafer is represented with columns titled *Weight_A* and *Weight_B*. It should be noted that, due to space reasons, names of some columns have been abbreviated in Figure 31. The schema element names in the AnyMap tool screenshots are given in full.



Meas. Nr.	Ord. Nr.	Weight_A	Radius_A	Th._A	Pos.Pass_A	Fall.T.	Error_A	Fall._A	Weight_B	Radius_B	Th._B	Pos.Pass_B	Fall._B	Error_B
91	1	7041	6256	8	0.089	0	-9999	0	7113	6323	8	-0.182	0	-9999
91	2	6816	6059	8	-0.073	0	-9999	0	6839	6079	8	-0.091	0	-9999
91	3	6762	6014	8	-0.554	0	-9999	0	6756	6007	8	-0.062	0	-9999
91	4	6659	5926	8	-0.844	0	-9999	0	6638	5906	8	-0.058	0	-9999
91	5	3711	5314	3	10000	R	-9999	R	6621	5891	8	-47.542	U	-9999
91	6	6640	5912	8	0.051	0	-9999	0	6634	5909	8	-0.147	0	-9999
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

Figure 31: Double-layered CSV document example

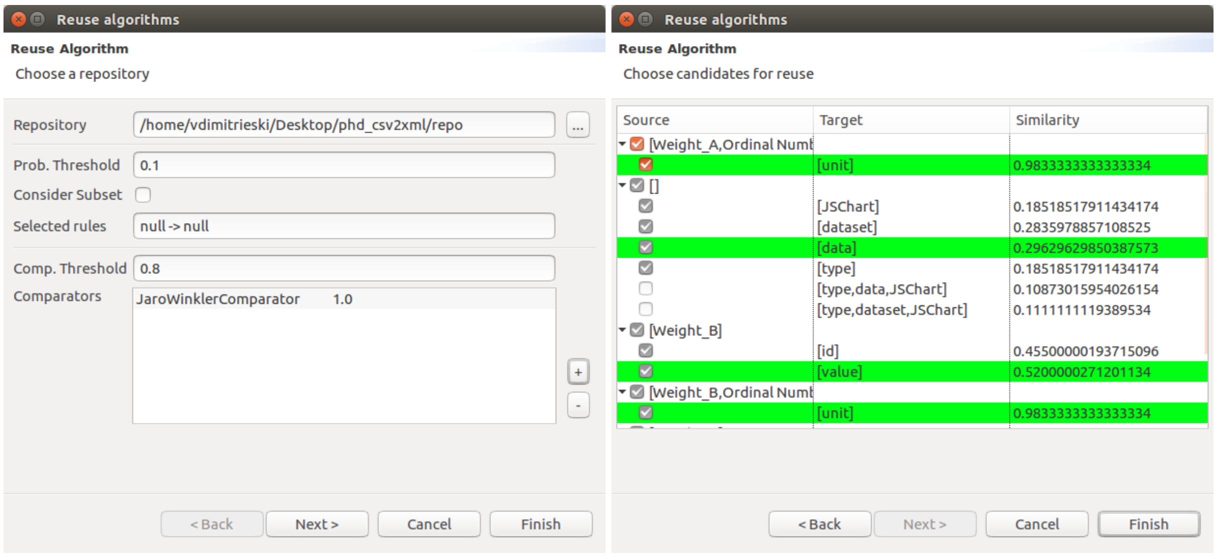
In some use cases the second measuring method is preferred as it potentially ensures better production quality at the expense of network utilization as it requires more data to be sent. Devices using this measurement method must be configured differently than devices using the first measurement method. This change in the configuration causes the creation or adaptation of an integration adapter that would be able to recognize the new data structure and then transform it to a desired target technical space. In our use case, the double-layered CSV documents also need to be mapped onto the same XML schema from Section 6.1.1. Transformation rules presented in Figure 10 are also applicable to double-layered CSVs but with minor adaptations. For example, instead of mapping just the *Weight* schema element, both *Weight_A* and *Weight_B* schema elements should be mapped onto the *value* element.

There are multiple ways in which this mapping can be specified. First, a manual specification is possible in which a developer can specify the entire mapping from scratch. This process is similar to the one presented in Section 6.1.1. Second, a semi-automatic specification is possible in which the developer can rely on a mapping repository and automation algorithms. They can provide a set of element mapping candidates from which the developer can select the ones that are the most suitable for the current integration task. Although this approach requires some manual intervention from the developer, the degree of the involvement is far lower than it is the case with the manual specification. It is only required that a developer sets up the reuse algorithm and its parameters, choose appropriate element mapping candidates from a provided list, and if necessary adapt the resulting mapping.

Finally, the mapping can be specified in an automatic way. This requires the least amount of user involvement. All tasks are performed automatically by the AnyMap tool based on the predefined heuristics. In order for the automatic specification to yield satisfactory results, a mapping repository must be populated with enough mappings from that domain that would steer the process in the right direction. This is often applicable for the integration domains that have a lot of previously specified mappings. As we want to present the whole automation process on our double-layered example, in this section we will present the mapping specification that follows the semi-automatic approach and uses the reuse automation algorithm.

We need a repository that contains some previously created mappings from the same integration domain in order to use the semi-automatic approach. We take into consideration a repository that comprises single-layered CSV to XML mappings created in Section 6.1.1. Several developers have created three versions of the same mapping which are shown in Figure 29 and Listing 1. First two mapping versions (mapping v1 and mapping v2) represent a common integration scenario in this integration domain. Both perform the same logic but are specified in a slightly different way, i. e., they are specified at different granularity levels. As they are common, in the repository we can find four instances of the mapping v1 and three instances of the mapping v2. The third version of the mapping (mapping v3) is created by a developer that did not need a value filter ($Weight > 7000$) and does not prefer to use meta-data information about the elements (e. g., $Weight.elementName$). Although this mapping is not so common in this integration scenario, it is sometimes required to visualize unfiltered measurement data. Therefore, only a single instance of the the mapping v3 can be found in the repository.

The mapping specification process starts by loading double-layered CSV schema and the XML schema into the AnyMap tool. After both TS schemas are loaded, a developer chooses the reuse automation mechanism and configures it. The configuration is performed through a dialog presented in the left side of Figure 32. A path to the reuse repository is defined in the first section of the dialog. In this example we use a local repository which content is described in the previous chapter.



Repository	/home/vdimitrieski/Desktop/phd_csv2xml/repo
Prob. Threshold	0.1
Consider Subset	☐
Selected rules	null -> null
Comp. Threshold	0.8
Comparators	JaroWinklerComparator 1.0

Source	Target	Similarity
☑ [Weight_A,Ordinal Numb		
☑	[unit]	0.9833333333333334
☑ []		
☑	[JSChart]	0.18518517911434174
☑	[dataset]	0.2835978857108525
☑	[data]	0.29629629850387573
☑	[type]	0.18518517911434174
☐	[type,data,JSChart]	0.10873015954026154
☐	[type,dataset,JSChart]	0.1111111119389534
☑ [Weight_B]		
☑	[id]	0.45500000193715096
☑	[value]	0.5200000271201134
☑ [Weight_B,Ordinal Numb		
☑	[unit]	0.9833333333333334

Figure 32: The reuse configuration dialog (left) and the element mapping candidates dialog (right)

In the second section of the dialog, the developer sets the probability threshold that defines a minimal probability after which an element mapping candidate is presented to the developer. If the developer checks the *Consider Subset* checkbox, the algorithm will provide element mapping candidates which comprise a subset of schema elements that are selected by the developer. Otherwise, only a superset or an equal set to the one provided as an input is considered to be a candidate. Selected schema elements that are provided as an input to the algorithm are presented in the *Selected rules* textbox.

The third section of the dialog allows selection and configuration of schema matchers (comparators). As we specified in detail in Section 5.4 schema matchers are used to detect similarities between current schema elements and repository schema elements. These similarities are latter used by the reuse algorithm and combined into a probability of a specific element mapping being the appropriate one for the current mapping context. Schema matchers can detect element similarity based on the structure or semantics of the elements. In our particular use case, the data sent by the sensors follows strict naming rules. If a property is measured multiple times by a single sensor in a single measurement pass, for each measurement a new letter is appended to the property name. Therefore, similarity between CSV columns (source schema elements) in this use case can be measured by using a string comparison mechanism.

In Figure 32 the Jaro-Winkler [198] string comparison algorithm is selected as it is the most suitable for short names and strings.

The AnyMap tool allows for multiple schema matchers to be selected and used simultaneously. Each of the selected matchers is assigned with a weight value which is later used to modify the result of each comparator. As we have only one matcher selected its weight is set to 1.0. Additionally, by entering a value in the *Comp. threshold* textfield, the developer sets the minimal element similarity that needs to be reported by a matcher in order to consider two elements as similar. In this example we consider elements to be similar if they have the similarity of 80% (0.8) or above.

After running the configured reuse algorithm the calculated element mapping candidates are shown in a dialog which is presented at the right side of Figure 32. In this dialog, element mapping candidates are grouped by the source schema elements. In each group target schema elements of the element mapping candidates are presented together with the probability of them being a good fit for the current integration scenario. Elements highlighted with the green color are candidates with the highest probability.

In order to shed some light on the probability values in Figure 32, we will present two element mapping candidates in more details. The first candidate in the dialog is the following element mapping $[OrdinalNumber, Weight\_A \longrightarrow unit]$ with the ~0.98 probability. This element mapping was created based on the $[OrdinalNumber, Weight \longrightarrow unit]$ found in all v1 and v2 repository mappings. Therefore, it is highly probable that this rule should be applied to the current mapping. The probability is not equal to 1 as the string similarity between *Weight* and *Weight_A* is not equal to 1 (0.95 according to Jaro-Winkler algorithm). Another notable case is the case of $[Weight\_B.elementName \longrightarrow id]$ and $[Weight\_B \longrightarrow value]$ element mappings. The second mapping is present in all repository mappings while the first one is present in all v1 and v2 mappings. A single v3 mapping causes the difference of probabilities between the two element mappings. The developer decided not to use meta-data information when connecting the *Weight* schema element to the *id* element, but to use a string literal "Weight".

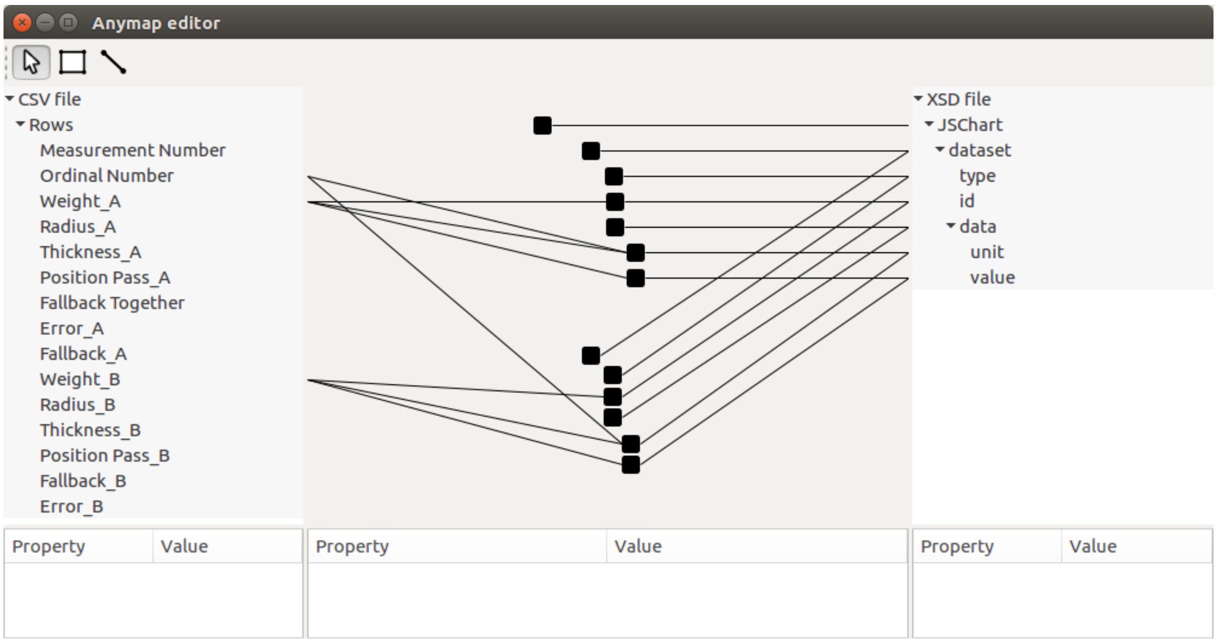


Figure 33: The double-layered CSV2XML mapping specification

The developer selects element mapping candidates that are the most appropriate to the current integration scenario. If this would be a fully automatic algorithm, candidates highlighted in green would be automatically selected. In either case, selected candidates are automatically applied (created) in the mapping canvas. The completed mapping specification is presented in Figure 33 while the corresponding formula-based representation can be seen in Listing 3. The generic element tree on the left side is created from the double-layered CSV document presented in Figure 31 while the generic element tree on the right side is created from the XSD document presented in the bottom part of Figure 9.

1 $[\emptyset \Longrightarrow \mathtt{JSChart}]$
2
3 //hierarchy #1
4 $[\emptyset \Longrightarrow \mathtt{dataset}]$
5 $[\text{“line”} \Longrightarrow \mathtt{type}]$
6 $[\mathtt{Weight\_A.elementName} \Longrightarrow \mathtt{id}]$
7 $[\emptyset \longrightarrow \mathtt{data}]$
8 $[\mathtt{OrdinalNumber} \overset{\mathtt{Weight\_A}>7000}{\longrightarrow} \mathtt{unit}]$
9 $[\mathtt{Weight\_A} \overset{\mathtt{Weight\_A}>7000}{\longrightarrow} \mathtt{value}]$
10
11 //hierarchy #2
12 $[\emptyset \Longrightarrow \mathtt{dataset}]$
13 $[\text{“line”} \Longrightarrow \mathtt{type}]$
14 $[\mathtt{Weight\_B.elementName} \Longrightarrow \mathtt{id}]$
15 $[\emptyset \longrightarrow \mathtt{data}]$
16 $[\mathtt{OrdinalNumber} \overset{\mathtt{Weight\_B}>7000}{\longrightarrow} \mathtt{unit}]$
17 $[\mathtt{Weight\_B} \overset{\mathtt{Weight\_B}>7000}{\longrightarrow} \mathtt{value}]$

Listing 3: The formula-based representation of the double-layered CSV2XML element mapping specification

In Figure 33 we visually separated two sets of element mappings that need to be executed separately. One set of element mappings needs to result in a dataset based on the *Weight_A* element values while another dataset is based on the *Weight_B* values. Therefore, we need to create sub-hierarchies in the mapping so that they can be executed separately. Each of the two operator sub-hierarchies (lines 2–8 and lines 9–15 of Listing 3) has a separate $[\emptyset \Longrightarrow \mathtt{dataset}]$ rule as its root element mapping. The developer had to manually create $[\emptyset \Longrightarrow \mathtt{dataset}]$ and $[\text{“line”} \Longrightarrow \mathtt{type}]$ element mappings as the reuse algorithm created only one. The reason is that on the left hand side of the element mapping rule is a constant which is the same for all the rules in the repository. Therefore, the reuse algorithm suggested the creation of a single element mapping in this integration scenario. Additionally, the developer sets the parent–child relationships for each operator to create execution hierarchies. This causes operators to be executed as a group and in appropriate order.

Generated integration adapter takes the CSV document from Figure 31 as an input and produces the XML document from Listing 4 as an output. Just like in Listing 2, we have omitted repetitive lines. When the information system receives the XML document it plots its content in a form of the line graph which presented in Figure 34.

6.2 MODEL INTERCHANGE BETWEEN META-MODELING ENVIRONMENTS

In this section we present a slightly different use case in which we have applied our approach. Our aim is to provide an external import and export mechanisms for the MetaEdit+ and Microsoft Visio (meta-)modeling tools by using our approach together with M3-Level-Based

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<JSChart>
        <dataset id="Weight_A" type="line">
                <data unit="1" value="7041"/>
                <data unit="9" value="7049"/>
                <data unit="10" value="7398"/>
                <data unit="11" value="7814"/>
                <data unit="12" value="7508"/>
                <!-- omitted due to length -->
        </dataset>
        <dataset id="Weight_B" type="line">
                <data unit="1"  value="7113"/>
                <data unit="10" value="7292"/>
                <data unit="11" value="7752"/>
                <data unit="12" value="7506"/>
                <data unit="13" value="7017"/>
                <!-- omitted due to length -->
        </dataset>
</JSChart>
```

Listing 4: XML output file example

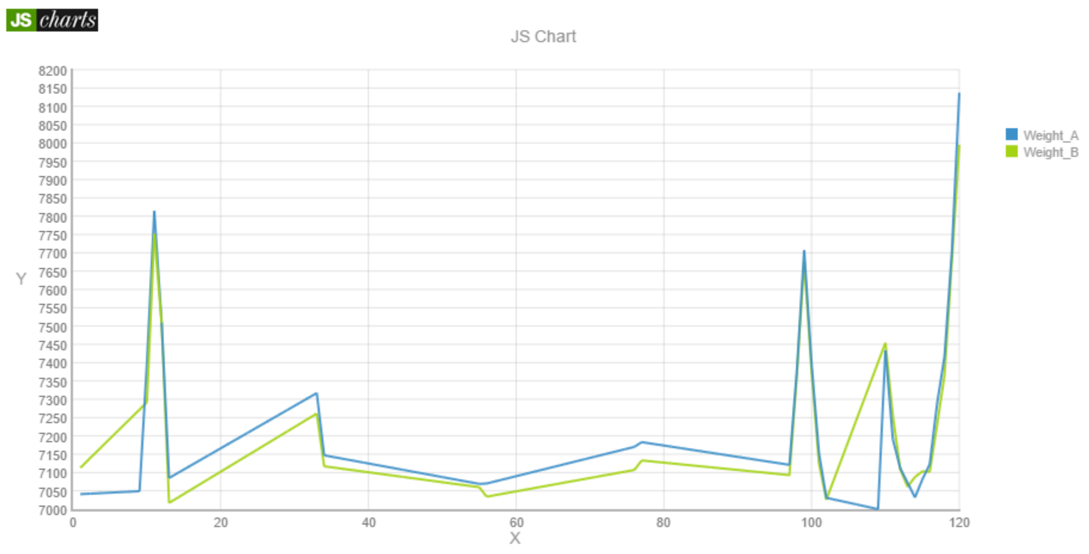


Figure 34: The line chart of the double-layered measurement data

Bridgess (M3Bs). In the rest of this section, the use case presented in our paper [108] is given in more details.

First, in order to understand our motivation behind this example, we must introduce all the key concepts. Models play an important role in Domain-Specific Modeling [103] and other related development disciplines. Generally, models represent a system in an abstract way, improve the understanding of a system, and facilitate the communication between different stakeholders. The creation of models is the result of a modeling process which is supported by a modeling tool. A special class of these modeling tools are meta-modeling tools. In addition to providing a user with a set of predefined modeling languages, meta-modeling tools provide a mechanism for the specification of new modeling languages. Examples of meta-modeling tools are: MetaEdit+ [103], Eclipse Modeling Framework [179], and Microsoft Visio [86].

An important requirement for modeling tools, including meta-modeling tools, is the interoperability with other tools. In the context of this use case, interoperability is defined as the ability of two or more tools to exchange models or meta-models. Additionally, these exchanged models and meta-models must be usable in the tools they are imported in. This means that if two meta-modeling environments have their own meta-model versions of the same modeling language, models that conform to the meta-model in the first environment, once imported, must also conform to the meta-model of the same language in the second environment.

Often, tools support a specific task in the development process. Therefore, a successful application of the whole development process depends heavily on the degree of interoperability between the tools used in the process. Besides the coexistence of tools in the same development process, the evolution of a tool landscape is an important aspect. As the software industry constantly evolves, modeling tools also evolve and the old ones are being replaced by new tools that better fit customer's needs. In order to avoid the vendor lock-in effect, interoperability between tools is necessary and enables the reuse of existing models between tools from different vendors.

Currently, the interoperability between meta-modeling tools is not widely supported [104, 107]. There is no suitable model exchange approach that takes meta-models into consideration. We will address this lack of interoperability between meta-modeling tools and use the approach and the tool proposed in Chapter 5 to provide the exchange of models while retaining their conformance to their respective meta-models. This way we plan to allow an efficient and user-oriented import and export of models in tools currently used in the industry. In this section, we will focus on the model interchange between the two environments: MetaEdit+ [103] and Microsoft Visio [86]. We use the following versions of the aforementioned environments: MetaEdit+ Workbench 5.5 Single User Evaluation Version and Microsoft Visio Professional 2013.

MetaEdit+ is a meta-modeling environment which provides means to create and use graphical DSLs. An abstract syntax of a new language is created by using meta-modeling concepts defined in the Graph-Object-Property-Port-Role-Relationship (GOPPRR) [141, 51] meta-meta-model. A graphical concrete syntax is created by using the built-in graphical editor and by assigning drawn graphical symbols to the corresponding meta-model elements. For the purpose of this example, we have created a modeling language for specifying Event-Driven Process Chain (EPC) models.

According to [185], Event-Driven Process Chains are "*an intuitive graphical business process description language*" which is used "*to describe processes on the level of their business logic, not necessarily on the formal specification level, and to be easy to understand and use by business people*". Although there are many concepts that can be used for the specification of EPC models, only three are considered as core EPC concepts. According to [185] the core concepts can be described as follows:

- *Functions*—A function corresponds to an action (task , process step) which needs to be executed. Functions are usually represented as rectangles with the function name displayed in the center of the graphical symbol.

- *Events*—Events describe the situation before or after a function is executed. They can also be considered as preconditions or postconditions of a function execution. Events are graphically represented as hexagons with the event name displayed in the center of the hexagon symbol.

- *Logical connectors*—Connectors are main concepts for the specification of the control flow as they are used to connect functions and events. There are three types of connectors:

(i) logical AND (∧), (ii) logical OR (∨), and (iii) logical XOR (×). Logical connectors are graphically represented as circles with the corresponding type symbol displayed in the center of the circle.

We have created a MetaEdit+ meta-model of the EPC language that comprises all of the aforementioned core concepts. The meta-model also contains the concept of a directed arc as instances of the core concepts need to be connected via directed lines. We have also specified a concrete syntax that comprises commonly used symbols for the core EPC concepts. After providing the concrete syntax it is possible to use the EPC language to create EPC models in MetaEdit+. An example model specified in MetaEdit+ by using the specified modeling language is presented in Figure 35. Our goal is to migrate this model to the Visio environment without losing any domain-related important information.

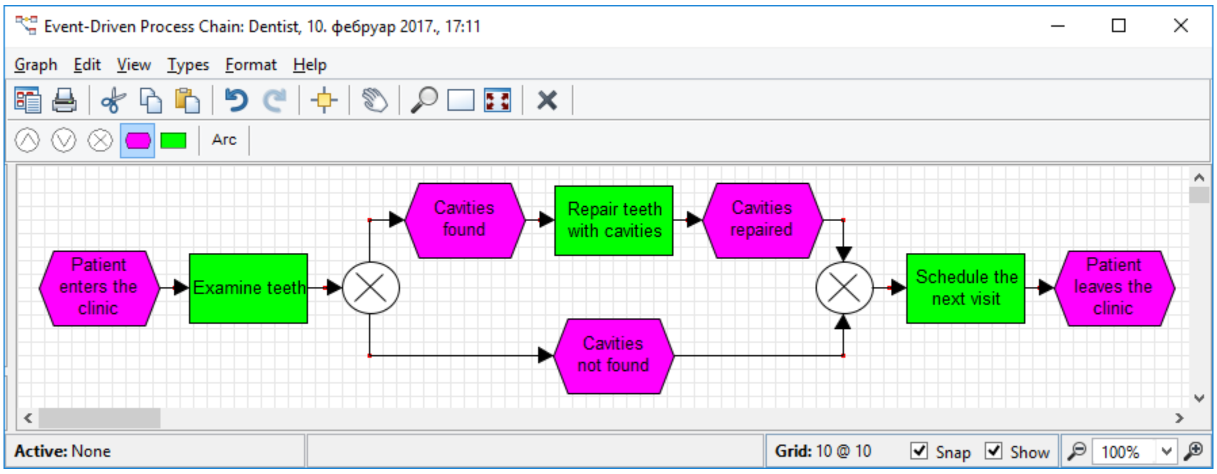

Event-Driven Process Chain: Dentist, 10. фебруар 2017., 17:11
Graph Edit View Types Format Help
Arc
Patient enters the clinic
Examine teeth
Cavities found
Repair teeth with cavities
Cavities repaired
Cavities not found
Schedule the next visit
Patient leaves the clinic
Active: None
Grid: 10 @ 10
Snap
Show
100%

Figure 35: The input EPC model specified in MetaEdit+

We choose to model a simplified version of a well-known process of patient visit to a dental clinic. When a patient comes to the dental clinic (*Patient enters the clinic* event) a dentist examines their teeth for cavities (*Examine teeth* function). There are two possible outcomes of the examination: (i) cavities are found (*Cavities found* event) or (ii) cavities are not found (*Cavities not found* event). As only one of these events could be an outcome of the examination, the two process paths are separated by using the *XOR (×)* logical connector. If the cavities are found, the dentist needs to repair teeth with cavities (*Repair teeth with cavities* function). After all teeth with cavities are repaired the *Cavities repaired* event is triggered. No matter which event has been triggered, *Cavities not found* or *Cavities repaired*, the dentist schedules the next regular visit for the patient (*Schedule the next visit* function). The patient then leaves the dental clinic (*Patient leaves the clinic* event).

An EPC language meta-model can also be created in Microsoft Visio. Such a meta-model is called a *stencil* and is specified by the means of implicitly defined Visio meta-meta-model. The process of creating the stencil is a bit more complex than it is the case with the MetaEdit+. For the purpose of this example we have obtained a stencil containing all of EPC core concepts defined in the MetaEdit+ meta-model.

There are two possible ways to provide model interchange between the environments by using our approach and the AnyMap tool. The first way is to create low-level mappings between technical spaces in which the models of both environments are serialized. This can be burdensome and time-consuming task for meta-modelers as they are usually just aware of correspondences between high-level meta-modeling concepts and are not aware of model serialization details used by an environment. Even in the case of the latest versions of the

MetaEdit+ and Visio tools which can both use the XML TS to store models, such a low-level integration would not be easy for a meta-modeler.

The second way is to use the M3-Level-Based Bridges (M3B) component in order to transform all models and meta-models from their respective environments into the EMF technical space. Correspondences (i. e., mappings) are then specified between high-level meta-modeling concepts represented in the EMF technical space. These mappings are used for generation of Epsilon Transformation Language (ETL) or ATL Transformation Language (ATL) specifications. These transformation specifications are at the same abstraction level as mappings. However, as they are specified textually, it is arguably easier for a meta-modeler to use a graphical syntax to draw correspondences between the meta-modeling concepts then to learn a new transformation language. Both ETL and ATL transformations are executed in an execution environment which is a part of the EMF technical space.

According to Kern in [106], "*the basic idea of an M3-level-based bridge is the conversion of models and meta-models from one model hierarchy into another model hierarchy*". This means that the M3B-based approach is not only suitable for transforming model hierarchies to EMF but also to other technical spaces as well. However, for our use case M3Bs are necessary in order to transform all concepts from different meta-modeling environments into the EMF technical space. An M3B comprises two types of transformations: (i) M2 transformation and (ii) M1 transformation. M2 transformation is specified against meta-meta-models of technical spaces that are being bridged (M3 level). This transformation is specified manually and it represents a recipe on how to transform concepts at the level of meta-models (M2 level), hence the name M2 transformation. Based on the meta-modeling concepts that are transformed during the execution of M2 transformations, the specification of M1 transformation is derived. In another words, "*the mapping of the M1 transformation depends on the transformation instance created by the previously executed M2-transformation*" [105, 106]. The M1 transformation reads source models and creates target models during its execution (M1 level).

For this use case we chose the second way of integrating meta-modeling environments. The main steps of the integration process and the used AnyMap components are presented in Figure 36. There are nine main steps that need to be performed in order to provide model interchange between the MetaEdit+ and Microsoft Visio. Ordinal number of each step is given next to the step name and denoted in the figure as a black circle. In this example we do not rely on automation algorithms to specify mappings, hence the lack of the *Reuse* module in this figure.

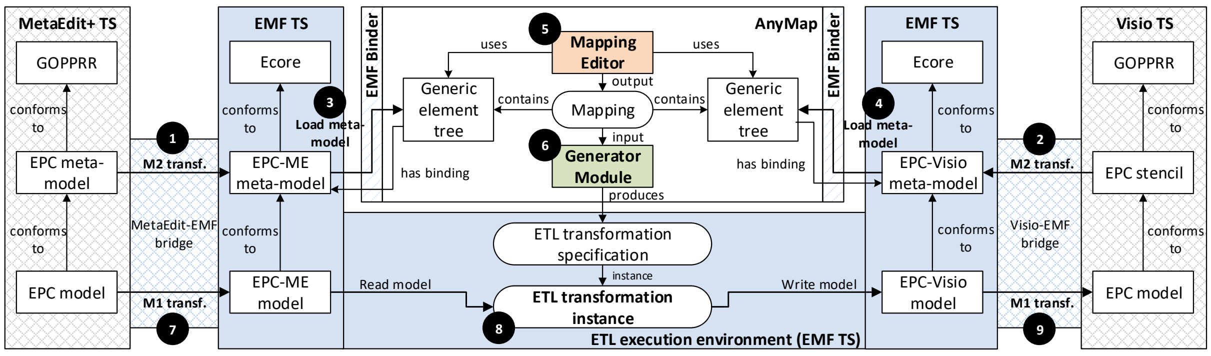


Figure 36: Steps of the model interchange between meta-modeling environments

Here we give a short overview of all nine process steps:

Step 1 *Import EPC meta-model from MetaEdit+ to the EMF technical space*. The EPC meta-model from the MetaEdit environment is created in the EMF technical space by using the M2 transformation of the MetaEdit-EMF bridge. This bridge is in fact a M3B but for the easier understanding in the rest of the text we will denote all M3Bs just as bridges. The created meta-model conforms to the Ecore meta-meta-model and we name it EPC-ME meta-model. This kind of M3B transformation is called M2 transformation as it operates at the level of meta-models.

Step 2 *Import EPC meta-model from Visio to the EMF technical space*. The EPC-Visio meta-model has been created in the EMF technical space based on the EPC stencil from Visio. The EPC-Visio meta-model conforms to the Ecore meta-meta-model. This step is performed by executing the M2 transformation from the Visio-EMF bridge.

Step 3 *Load the EPC-ME meta-model in the AnyMap tool*. The meta-modeler uses the EMF binder to import the EPC-ME meta-model created in Step 1. This creates a source generic element tree in the AnyMap tool.

Step 4 *Load the EPC-Visio meta-model in the AnyMap tool*. The target generic element tree is created from the EPC-Visio meta-model. This is also done by using the EMF binder.

Step 5 *Create the mapping specification*. Once both meta-models are represented as generic element trees, the meta-modeler uses the *Mapping Editor* module to specify a mapping. In this use case the meta-modeler specifies mappings manually without using any of the automation mechanisms. The output of this step is the mapping between the EPC-ME and EPC-Visio meta-model elements.

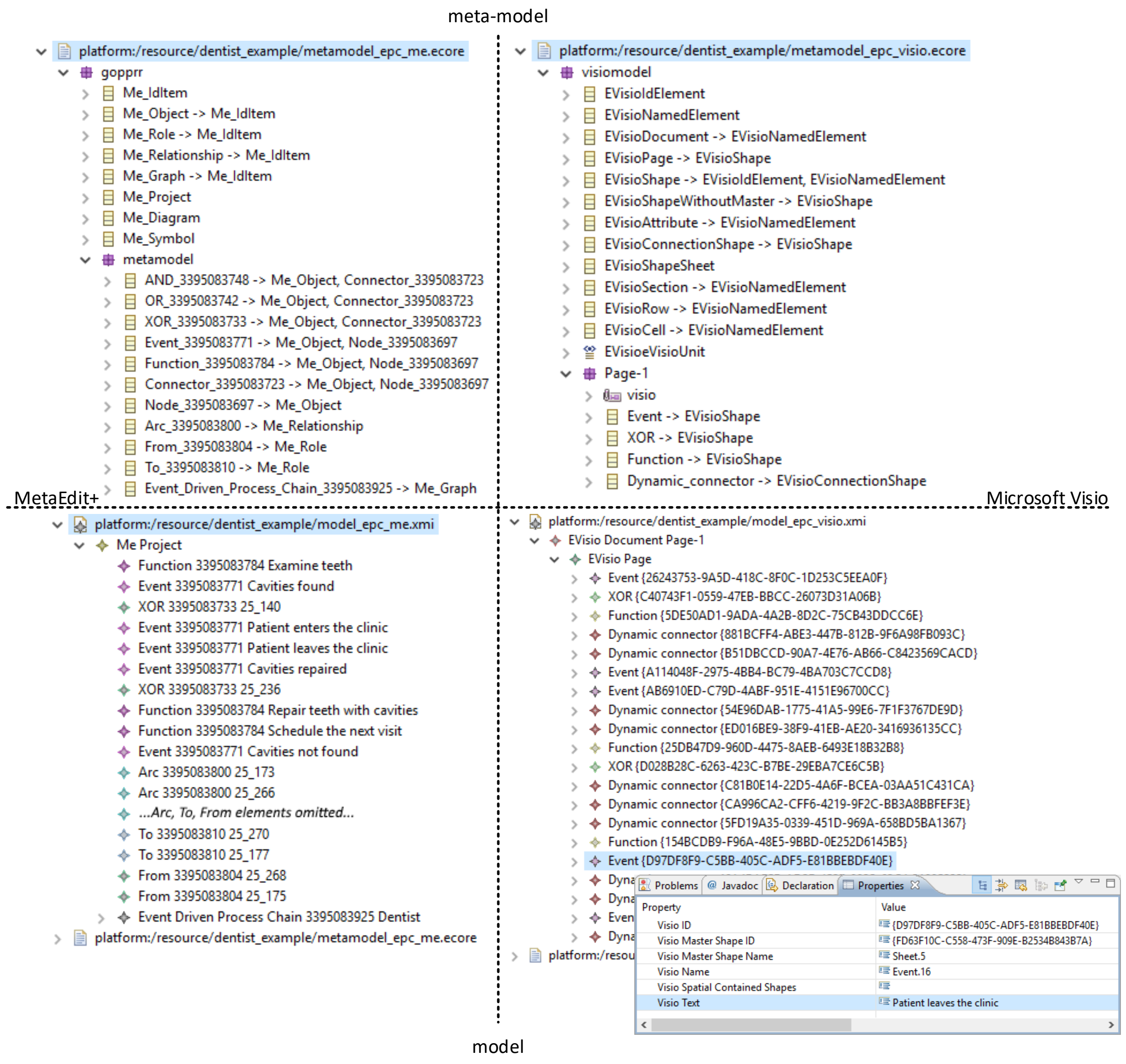
Step 6 *Invoke the ETL code generator*. The meta-modeler invokes the ETL code generator that generates an ETL specification of a model-to-model transformation. The generator takes the previously created mapping as an input and produces a corresponding textual ETL specification.

Step 7 *Import EPC model from MetaEdit+ to the EMF technical space*. The ETL code generator generates specifications which are using meta-model concepts found at the M2 level of abstraction. However, for their execution it is necessary to have an EMF representation of the source model from the MetaEdit+ environment. Such a model, named EPC-ME model, is created by invoking the M1 transformation of the MetaEdit-EMF bridge. The M1 transformation is executed at the M1 abstraction level and it ensures that the generated model conforms to the meta-model generated by the M2 transformation of the MetaEdit-EMF bridge.

Step 8 *Execute the generated ETL transformation*. By executing the instance of the generated ETL transformation, a source EPC-ME model is transformed to a corresponding EPC-Visio model that conforms to the EPC-Visio meta-model.

Step 9 *Export the EPC-Visio model from the EMF technical space to Visio*. By invoking the M1 transformation of the Visio-EMF bridge, the EPC-Visio model that is the result of the previous step is transformed the model that can be opened in Microsoft Visio and that conforms to the Visio EPC stencil.

Existing bridges (MetaEdit-EMF and Visio-EMF) are built as plug-ins for the Eclipse IDE. Therefore, a user can perform all the process steps from Figure 36 inside the Eclipse environment. During the bridge configuration phase a user just needs to provide paths to the meta-models and models that are being imported into the EMF technical space. It is also worth noting that the Steps 1–6 are performed just once for a modeling language (meta-model) that needs to be integrated between the two meta-modeling environments. Afterward, for each model specified in this language, only Steps 7–9 are performed. The result of transforming the meta-models from the MetaEdit+ and Microsoft Visio TSs are presented in the top part of Figure 37. In the same figure we present the EPC models in the EMF TS. The models will be described in more details later in this section.



```
platform:/resource/dentist_example/metamodel_epc_me.ecore
  gopprr
    Me_IdItem
    Me_Object -> Me_IdItem
    Me_Role -> Me_IdItem
    Me_Relationship -> Me_IdItem
    Me_Graph -> Me_IdItem
    Me_Project
    Me_Diagram
    Me_Symbol
    metamodel
      AND_3395083748 -> Me_Object, Connector_3395083723
      OR_3395083742 -> Me_Object, Connector_3395083723
      XOR_3395083733 -> Me_Object, Connector_3395083723
      Event_3395083771 -> Me_Object, Node_3395083697
      Function_3395083784 -> Me_Object, Node_3395083697
      Connector_3395083723 -> Me_Object, Node_3395083697
      Node_3395083697 -> Me_Object
      Arc_3395083800 -> Me_Relationship
      From_3395083804 -> Me_Role
      To_3395083810 -> Me_Role
      Event_Driven_Process_Chain_3395083925 -> Me_Graph
```

```
platform:/resource/dentist_example/metamodel_epc_visio.ecore
  visiomodel
    EVisioIdElement
    EVisioNamedElement
    EVisioDocument -> EVisioNamedElement
    EVisioPage -> EVisioShape
    EVisioShape -> EVisioIdElement, EVisioNamedElement
    EVisioShapeWithoutMaster -> EVisioShape
    EVisioAttribute -> EVisioNamedElement
    EVisioConnectionShape -> EVisioShape
    EVisioShapeSheet
    EVisioSection -> EVisioNamedElement
    EVisioRow -> EVisioNamedElement
    EVisioCell -> EVisioNamedElement
    EVisioeVisioUnit
    Page-1
      visio
      Event -> EVisioShape
      XOR -> EVisioShape
      Function -> EVisioShape
      Dynamic_connector -> EVisioConnectionShape
```

```
platform:/resource/dentist_example/model_epc_me.xmi
  Me Project
    Function 3395083784 Examine teeth
    Event 3395083771 Cavities found
    XOR 3395083733 25_140
    Event 3395083771 Patient enters the clinic
    Event 3395083771 Patient leaves the clinic
    Event 3395083771 Cavities repaired
    XOR 3395083733 25_236
    Function 3395083784 Repair teeth with cavities
    Function 3395083784 Schedule the next visit
    Event 3395083771 Cavities not found
    Arc 3395083800 25_173
    Arc 3395083800 25_266
    ...Arc, To, From elements omitted...
    To 3395083810 25_270
    To 3395083810 25_177
    From 3395083804 25_268
    From 3395083804 25_175
    Event Driven Process Chain 3395083925 Dentist
platform:/resource/dentist_example/metamodel_epc_me.ecore
```

```
platform:/resource/dentist_example/model_epc_visio.xmi
  EVisio Document Page-1
    EVisio Page
      Event {26243753-9A5D-418C-8F0C-1D253C5EEA0F}
      XOR {C40743F1-0559-47EB-BBCC-26073D31A06B}
      Function {5DE50AD1-9ADA-4A2B-8D2C-75CB43DDCC6E}
      Dynamic connector {881BCFF4-ABE3-447B-812B-9F6A98FB093C}
      Dynamic connector {B51DBCCD-90A7-4E76-AB66-C8423569CACD}
      Event {A114048F-2975-4BB4-BC79-4BA703C7CCD8}
      Event {AB6910ED-C79D-4ABF-951E-4151E96700CC}
      Dynamic connector {54E96DAB-1775-41A5-99E6-7F1F3767DE9D}
      Dynamic connector {ED016BE9-38F9-41EB-AE20-3416936135CC}
      Function {25DB47D9-960D-4475-8AEB-6493E18B32B8}
      XOR {D028B28C-6263-423C-B7BE-29EBA7CE6C5B}
      Dynamic connector {C81B0E14-22D5-4A6F-BCEA-03AA51C431CA}
      Dynamic connector {CA996CA2-CFF6-4219-9F2C-BB3A8BBFEF3E}
      Dynamic connector {5FD19A35-0339-451D-969A-658BD5BA1367}
      Function {154BCDB9-F96A-48E5-9BBD-0E252D6145B5}
      Event {D97DF8F9-C5BB-405C-ADF5-E81BBEBDF40E}
      Dyna
      Dyna
      Even
      Dyna
platform:/resou
```

Problems | Javadoc | Declaration | Properties

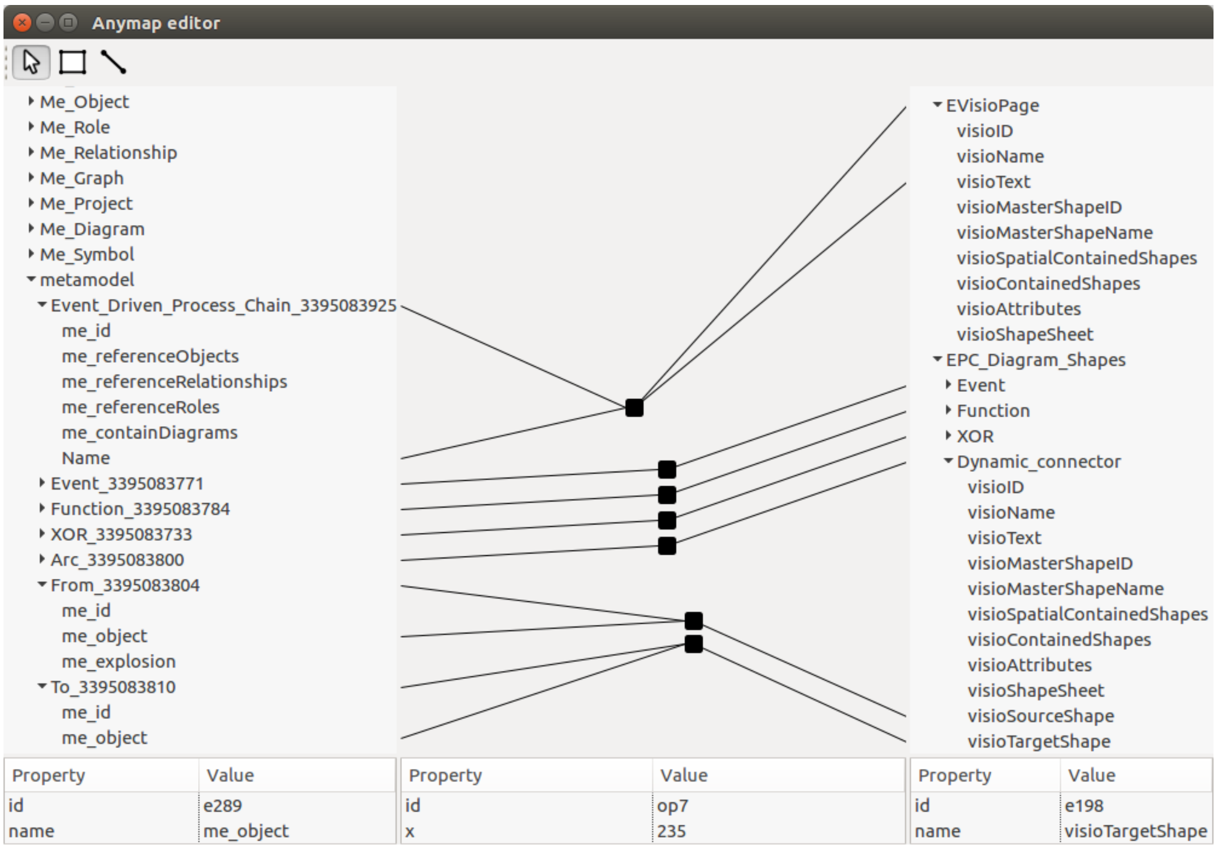
Property	Value
Visio ID	{D97DF8F9-C5BB-405C-ADF5-E81BBEBDF40E}
Visio Master Shape ID	{FD63F10C-C558-473F-909E-B2534B843B7A}
Visio Master Shape Name	Sheet.5
Visio Name	Event.16
Visio Spatial Contained Shapes	
Visio Text	Patient leaves the clinic

Figure 37: MetaEdit+ and Visio meta-models and models in the EMF technical space

Each meta-model that is created by using M3Bs, comprises an abstract meta-model part and the transformed meta-model part. If we take a look at the Visio meta-model (top-right part of Figure 37) we can see that abstract concepts are created directly in the root *visiomodel* package. The EPC meta-model concepts are created in the *Page-1* sub-package. Abstract concepts are concepts that represent concepts from a meta-meta-model, storage concepts as well as visualization concepts of a meta-modeling environment. These concepts are present in every EMF meta-model created from the same environment regardless of the meta-model being transformed. They are necessary as they are often inherited by the concepts of meta-models and they provide some properties and relationships necessary for the complete transformation. For example, in EPC-Visio meta-model *EVisioShape* is inherited by all other meta-model concepts that model visual elements which instances are shown on a Visio page. All child concepts will inherit attributes such is *visioText* which is essential for the mapping as it holds a value that represents a name of a model element.

The EPC-ME meta-model is created in a similar way and is presented the top-left part of Figure 37. All GOPPRR meta-meta-model concepts are created in the *gopprr* package. In the same package, some MetaEdit+ concepts aimed at the diagrammatic representation and grouping of models are created. These concepts are Diagram, Project and Symbol. All concepts created from the MetaEdit+ EPC meta-model are created in the *metamodel* sub-package. Therefore, in the created Ecore meta-models of the EPC language, we have concepts from M2 and M3 levels from their respective environments. This allows a creation of complete mappings between the meta-models as all properties of meta-concepts are present regardless of their explicit or implicit specification.

Although the meta-models imported in the EMF technical space represent the same modeling language, there is still some heterogeneity between their elements. For instance, elements with the same meaning may have different names or have different relationships. In order to overcome this heterogeneity, a user defines a mapping containing correspondences between the two meta-models. This is performed as a part of Step 5 of the integration process presented in Figure 36. The only requirement to start the mapping process is that both Ecore meta-models are presented as generic element trees in the AnyMap tool (Step 3 and Step 4). Generic representations of meta-models and the complete mapping specification may be seen in Figure 38.



- Me_Object
- Me_Role
- Me_Relationship
- Me_Graph
- Me_Project
- Me_Diagram
- Me_Symbol
- metamodel
  - Event_Driven_Process_Chain_3395083925
    - me_id
    - me_referenceObjects
    - me_referenceRelationships
    - me_referenceRoles
    - me_containDiagrams
    - Name
  - Event_3395083771
  - Function_3395083784
  - XOR_3395083733
  - Arc_3395083800
  - From_3395083804
    - me_id
    - me_object
    - me_explosion
  - To_3395083810
    - me_id
    - me_object

- EVisioPage
  - visioID
  - visioName
  - visioText
  - visioMasterShapeID
  - visioMasterShapeName
  - visioSpatialContainedShapes
  - visioContainedShapes
  - visioAttributes
  - visioShapeSheet
- EPC_Diagram_Shapes
  - Event
  - Function
  - XOR
  - Dynamic_connector
    - visioID
    - visioName
    - visioText
    - visioMasterShapeID
    - visioMasterShapeName
    - visioSpatialContainedShapes
    - visioContainedShapes
    - visioAttributes
    - visioShapeSheet
    - visioSourceShape
    - visioTargetShape

Property	Value
id	e289
name	me_object

Property	Value
id	op7
x	235

Property	Value
id	e198
name	visioTargetShape

Figure 38: MetaEdit+ to Visio mapping specification created in the AnyMap tool

Each *Event_Driven_Process_Chain_3395083925* instance created in MetaEdit+ is mapped onto a new *EVisioPage*. The name (*Name*) of the instance is set as the *visioText* of the created *EVisioPage*. This pattern is repeated for almost all other mappings. First, corresponding meta-concepts are mapped and then the *Name* attribute is mapped onto *visioText* in order to be displayed on the Visio diagram. However, due to the clarity and understandability of the figure, other mappings that follow this pattern are presented in the collapsed form. A code snippet of the generated transformation is presented in Listing 5. The entire code of ETL transformations can be found in [106]. These mappings are: [Event_3395083771 $\longrightarrow$ Event; Name $\longrightarrow$ visioText] (lines 6–10 of Listing 5) and [Function_3395083784 $\longrightarrow$ Function; Name $\longrightarrow$ visioText] (lines 11–15 of Listing 5). As the *XOR_3395083742* element has a fixed name, only a top-level element mapping is created between it and the *XOR* element in the EPC-Visio meta-model. Finally, each arc (*Arc_3395083800*) that connects symbols in the MetaEdit+ EPC model must be translated into a dynamic connector (*Dynamic_connector*) between corresponding elements in the Visio diagram. Elements connected by an arc have *from* and *to* roles in that relationships (*From_3395083804* and *To_3395083810*). EPC-Visio model elements that are created EPC-ME model elements with a *from* role are assigned as source elements (*visioSourceShape*) of the appropriate dynamic connector. Similarly, target elements of dynamic connectors are assigned as target shapes (*visioTargetShape*).

The ETL code generator is executed once a user is satisfied with the created mapping. Generator execution results in an ETL specification of a model-to-model transformation between EPC-ME and EPC-Visio meta-models. In Listing 5 we can observe the same rules as the ones specified in the previous paragraph just in the ETL textual syntax. Some of the rules are simplified by creating an abstract parent rule. For example *graph2page* rule inherits a lot of its functionality from the abstract *Graph2Page* rule.

```
1  rule graph2page
2     transform event_driven_process_chain_3395083925 : INMM!
          Event_Driven_Process_Chain_3395083925
3     to evisiopage : OUTMM!EVisioPage extends Graph2Page {
4        evisiopage.visioText := event_driven_process_chain_3395083925.Name;
5     }
6  rule event2Event
7     transform event_3395083771 : INMM!Event_3395083771
8     to event : OUTMM!Event {
9        event.visioText := event_3395083771.Name;
10    }
11 rule function2Function
12    transform function_3395083784 : INMM!Function_3395083784
13    to function : OUTMM!Function {
14       function.visioText := function_3395083784.Name;
15    }
16 rule xor2xor
17    transform xor_3395083742 : INMM!XOR_3395083742
18    to _xor : OUTMM!XOR {
19    }
20 rule arc2dynamicConnector
21    transform arc_3395083800 : INMM!Arc_3395083800
22    to dynamic_connector : OUTMM!Dynamic_connector {
23       dynamic_connector.visioTargetShape := arc_3395083800.me_role.equivalent();
24       dynamic_connector.visioSourceShape := arc_3395083800.me_role.equivalent();
25    }
```

Listing 5: Code snippet from the generated ETL specification

This ETL specification is at the M2 level as it uses concepts of the source and target Ecore meta-models. An instance of this transformation is executed at M1 level against the provided source model. As a part of the Step 7, the EPC model from MetaEdit+ presented in Figure 35 is transformed into an EPC model in the EMF technical space which is shown in the bottom left corner of Figure 37. This transformation is performed by using the M1 transformation of the MetaEdit-EMF bridge. It can be seen in Figure 37 that the EPC-ME model contains all the events, functions, and logical connections as the original model specified in MetaEdit+. Due to the space reasons some of the arcs and *from* and *to* roles have been omitted from the figure.

The EPC-Visio model is created as a result of executing the ETL transformation from Listing 5 (Step 8). The model is presented in the bottom-right part of Figure 37. As the names of the model elements are not directly visible in the tree representation, we present a property view of the selected event. It can be seen that the selected event is created from the *Event_3395083771 Patient leaves the clinic* as they share the same name. As the final step of the model interchange process (Step 9) this generated EPC-Visio model is exported from the EMF technical space to the Visio environment by using M1 transformation from the Visio-EMF bridge. The generated EPC model in Visio is presented in Figure 39. If a new EPC model needs to be exchanged between MetaEdit+ and Visio environments, only the Steps 7–9 need to be executed again.

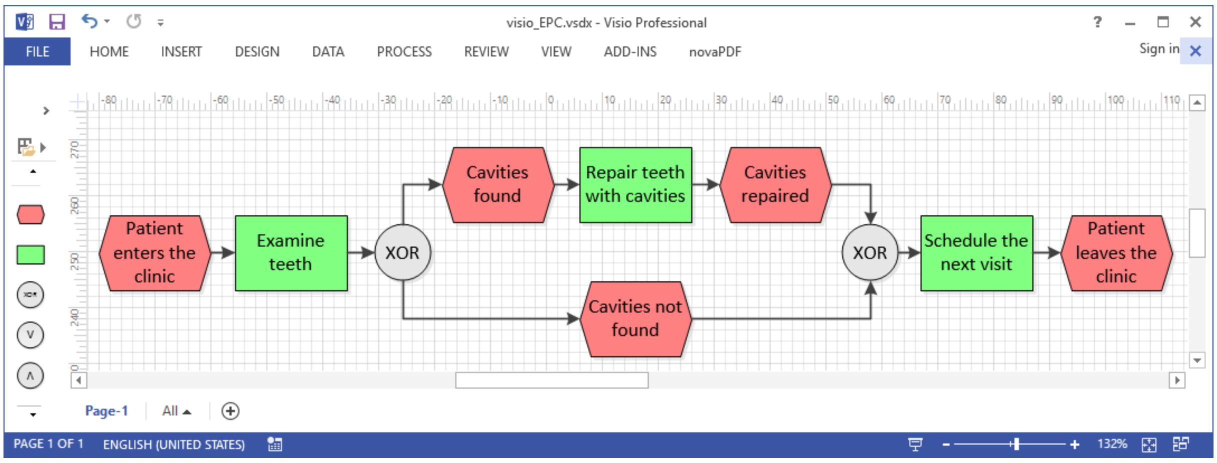


Figure 39: The output EPC model in Microsoft Visio

All elements of the MetaEdit+ model presented in Figure 35 are also present in the model depicted in Figure 39. Therefore, we may conclude that the model has been successfully migrated from the MetaEdit+ to Microsoft Visio without loosing any of the domain information. We should note that only information loss was related to the graphical positioning of the model elements in Visio. After model exchange we had to manually rearrange graphical symbols in order to get a readable diagram.

6.3 ANALYSIS OF THE MAPPING APPROACH AND DISCUSSION OF HYPOTHESES

We start with the discussion about the hypotheses introduced in Chapter 1. Further, we discuss benefits and drawbacks of our mapping approach and the AnyMap tool. All discussions are given in the light of the application scenarios presented in this chapter and surveyed tools presented in Chapter 4. Although the use cases presented in this chapter are just representative examples of what the approach and the tool are capable of, they provide a solid foundation for a meaningful discussion about hypotheses, proposed approach and the AnyMap tool characteristics.

Before discussing the derived hypotheses, we will first discuss the main hypothesis of the thesis, Hypothesis 0, that states: *It is possible to solve heterogeneity problems in TS integration* *by creating appropriate DSMLs and following the principles of MDSD approach*. As our approach is successfully applied in multiple TS integration scenarios, some of which are presented in this chapter, we can conclude that **it is possible to solve the heterogeneity problems in TS integration by using our approach**. This confirms the first part of the hypothesis. In order to fully confirm the hypothesis, we need to discuss whether our approach is an MDSD approach, and that the used mapping language is in fact a DSML.

The entire adapter development process that we promote, is based around models. First, there are models that represent TS schemas. Then, mapping specification represent transformation rules for transforming a source TS model into a target TS model. If we consider definitions of *model* and *model transformation* notions given in Chapter 2, mapping specification can be seen as a model of a M2M transformation. In this thesis, as main examples of model transformations, we are considering the following processes: code generation and the reuse of created element mapping specifications. In the rest of the chapter, we will call them code generation and reuse. A code generator takes a mapping model as an input and produces program code as an output. Therefore, code generators are considered as model-to-text transformations. Reuse on the other hand is an operation that takes an existing mapping specification or a collection of specifications and produces a new mapping specification. Therefore, reuse can be seen as a transformation between mapping specifications, i. e., between models of M2M transformations. Consequently, reuse can be classified as a high-order transformation. Based on these observations, it can be seen that our approach adheres to the main MDSD principle "Everything is a model", even for the specification of transformations between elements of TSs. Therefore, we may conclude that all the building blocks of our approach can be classified as models and model transformations which is in accordance with the main principles of MDSD (cf. Chapter 2).

In order for an approach to be considered as an MDSD approach, having main building blocks in accordance with its main principles is just a requirement but not a sufficient proof. It is also necessary that the goals of our approach are in accordance with the goals of MDSD. In the following paragraphs we discuss our approach in the light of main MDSD goals given in Chapter 2. The five goals are given in italics and enumerated from G1 to G5.

G1: Increase of the development speed through automation and single point of system definition. In contrast to the manual development, our solution offers a single definition point of transformation rules. For example, let us consider an adapter with a microservice architecture shown in Figure 25. If such an adapter is implemented manually, the development requires writing a lot of redundant code as the code for the same functionality is often scattered across multiple microservices such are discovery, transformation, and routing microservices. This introduces extra complexity and slows the development process. Each modification of single functionality must be also performed in multiple places. All of the aforementioned observations are also true for a non-microservice architectures as well. In these cases, other software design patterns and best practices are used in development where multiple classes are often developed in order to implement a single transformation task. By providing a single definition point of adapters in our approach, developers are able to create or modify an entire adapter at one place, in a single graphical view, and as a single logical unit. It is easier to track changes, understand logic, and even debug the transformation algorithm that is written in a single place. After a single specification is made, automatic code generation process follows and results in a fully functional adapter. This increases developers' productivity and reduces time needed to fully create or modify an adapter.

G2: Increase in software quality through formalization. By formally specifying main concepts and relationships between these concepts in the integration domain, a formal domain-specific language can be developed that supports domain experts. Our DSML, although intentionally simple, supports domain experts in building correspondences between source and target TSs via simple end-to-end mappings without introducing unnecessary complexity. Our graphical DSML comprises two visual elements, nodes and links, that allow domain experts to focus just on identifying and denoting correspondences between appropriate elements. By formally defining core concepts that are close to the domain, domain experts are provided means to easily accomplish the required tasks. In this case, accidental complexity is not introduced by inappropriate tooling and languages. Furthermore, specifications implemented using such a formal language (our DSML), can be checked for logical errors that stem from the usage of domain concepts. It should be noted that in our solution, if necessary, domain experts can use Java programming language to specify mapping rules in detail and influence their execution. Only the core part of the Java language needs to be used and there is no need to use specific external libraries or any complex frameworks. Thus, domain experts with just the most basic programming skills such as understanding of flow control and data manipulation language in Java can use it. This also facilitates the automatic validation of models as basic Java statements can be easily executed, evaluated, and simulated in a custom, domain-specific runtime engine. On the other hand, it is hard to perform such validations on adapters written using a GPL.

G3: Increase in component reuse and improved manageability of complexity through abstraction. Similar to some of the previous goals, formal way of specifying models and transformations and a single point of specification lead to the increase in component reuse and complexity reduction. To tackle the problem of component reuse, we have implemented the automation engine, presented in Section 5.4, to support automation process. As the integration between two technical spaces requires specification of element correspondences it is easy to reuse these correspondences if they are at the appropriate level of abstraction. These correspondences can be considered as more or less isolated units as they often do not depend on the execution of other correspondences. On the other hand, manual programming offers limited means to facilitate reuse due to a high degree of code coupling. For example, in a programming language such is Java, we can build libraries in which we can store reusable functions for transforming data. These functions may be independent of a schema being transformed, however, they are often bound to a specific TS to which the schema belongs to. Another drawback of a manually written adapter is that the creation of such a reusable library requires detailed analysis of adapter components and the possibility of their reuse. It also requires more advanced engineering and programming skills. If a transformation code is implemented in multiple functions and classes, it is hard to reuse it. On the other side, if correspondences are created between generic structures as a self-contained, high-level, and formally defined units which implementation does not depend on a particular TS, they are highly reusable. Our approach builds on this idea by considering a mapping DSML which is at the appropriate level of abstraction in order to provide useful and usable reuse. The automation engine that facilitates reuse is discussed later in this section when we discuss Hypothesis 3. As we already stated, accidental complexity is greatly reduced by having a formal domain-specific language and a single point of specification. This increase in abstraction, from Java functions to graphically represented correspondences using lines and nodes, allows domain experts to further mitigate accidental complexity introduced by Java and focus on the task laid before them.

G4: Greater domain expert inclusion in the development process. Nowadays, adapters are manually developed by software developers that are proficient in a GPL chosen to implement the adapter. Experts in the integration domain, machine engineers, process engineers, or data analysts very often just provide some basic information or an informal specification of the integration to be done. It is up to the software developers to do all the “dirty” work. In our approach, integration domain experts take the driving seat. They are responsible for the adapter specification task. They do not need to have an advanced programming skills as all the programming complexity is hidden behind the abstraction provided by the mapping language. It must be noted that domain experts must posses some programming skills, namely basic understanding of flow control and data manipulation in Java, in order to use our expression language. However, compared to the skill required for the manual adapter specification, this represents a major improvement.

G5: Better communication between different stakeholders in the software development process. Low understanding of the complex code among non-programmers is a well-known issue. In order to solve this issue, user documentation and different models need to be created based on the code and system architecture in order for the system overview to be presented to different stakeholders. One of the main issues that arises from this kind of an approach to creating models is that a model is often neglected and it becomes out of date very fast as it is not maintained properly. In our approach, model is developed first, it is in the hart of the development process and from it the adapter is generated. As such, it is necessary to maintain the model as it represents the main point of adapter specification. Furthermore, as the modeling concepts are formally defined in an unambiguous and domain-specific manner, models are concise and can be used to easily specify solutions in the domain.

Based on the previous discussion, we may conclude that our approach is built in full accordance with the main MDSD principles and is adhering to all MDSD goals. Therefore, we can state that *our approach to TS integration is an MDSD approach*. All benefits usually associated with the MDSD methodology such as reduced time to market, decreased number of errors, increased software quality and performance will be valid when applying our approach as well. However, in order to metrically evaluate all the benefits of the approach, a metric framework needs to be established and the approach to be applied on more use cases. It should be noted that our goal here is not to provide a metric-based evaluation as constructing a metric framework is a research problem on its own [22]. It requires constructing a metric framework and conducting a series of experiments. Our plans for future research include creating an evaluation framework that will be used to quantitatively evaluate our approach and DSML and compare them to other surveyed solutions.

Finally, Hypothesis 0 also states that the approach will utilize a DSML as a means to achieve integration between TSs. Although the approach can be considered as a generic and language-agnostic, we provided a simple, yet powerful mapping language. As the language is built so as to provide essential concepts for modeling correspondences in the TS integration domain, by definition *our mapping language is a Domain Specific Modeling Language (DSML)* as it is focused on a single domain, i. e., TS integration domain, and provides concepts easily understandable by domain experts.

We may conclude that Hypothesis 0 is confirmed as our approach is a MDSD approach that uses a DSML which we have successfully applied in several practical use cases to integrate different TSs and thus facilitate data exchange between them.

Several hypotheses were derived from the main hypothesis in order to address the main characteristics of the proposed approach. These characteristics are generic TS representation, graphical DSML, and mapping reuse. We will now discuss the derived hypotheses and the reasons for their confirmation or rejection together with the possible limitations of our approach in the light of these hypotheses.

Hypothesis 1 states: *It is possible to represent data schemas (i. e., meta-models) from the three-level technical spaces in a uniform way by using a graph-like representation, while preserving links to original elements*. In order to discuss if it is possible to represent meta-models as graphs, we must consider the definition of the meta-model notion and meta-meta-models as tools for their specification. As we have already said in Chapter 2, a meta-model can be considered as a "*specification of entity types, properties, and relationships*". Therefore, from the definition of the meta-model, we may see that it is in fact a graph where entity types and attributes are nodes and relationships between them are edges (links). Meta-meta-models, as means for specifying meta-models, must provide appropriate concepts that allow creation of nodes and links. For example, the most popular meta-meta-models are GOPPRR [141] and Ecore [179]. GOPPRR is assigned a graphical concrete syntax, and meta-models are in fact built in the form of graph diagrams. The main diagrammatic concept is called graph and it contains nodes such are objects, properties, ports, and roles. Each of these nodes is connected to other nodes via different kinds of relationships, i. e., links. Ecore-based meta-models are built in a similar manner and the example can be seen in Figure 20. Although Ecore has a condensed representation, if we extract attributes and relationship cardinalities as separate nodes, the meta-model can be seen as a regular graph. On the other hand, each textual EBNF grammar can also be represented in the form of a graph-like railroad diagram [33]. Each terminal symbol, non-terminal symbol, and connection type can be represented as a node, while a production rule is created by connecting them via links in a graph structure. Therefore, *in a three-level TS, it is possible to represent meta-models, i. e., schemas, in a graph-like manner due to their inherent graph nature*.

The limitation of our tool is of representational nature. Due to the visualization tools at our disposal, which are influenced by the choice of development environment, we have represented schemas as trees just like many other tools surveyed in Chapter 4. The root of the tree represents the schema document, while other schema elements that are organized as graphs need to be flattened into a tree structure by following the rules and the approach given in [134]. While this limits the usability of our tool to the schemas that can be flattened, i. e., schemas without infinite recursions of their elements, in use cases in which we have applied our approach we have rarely encountered infinite recursion. Moreover, in the cases where we have encountered infinite recursion at the schema level, flattening a structure to a desired depth and breaking recursion proved to be enough to solve the issue. However, we must conclude that *our approach is limited only to the subset of meta-models that can be flattened to a degree in which we do not lose any important semantic properties but can represent the meta-model in the form of a tree*. Our future research plans include identification and application of other visualization components that allow representing schemas as true graphs while preserving all the good characteristics of tree-based components such are easy handling and easy finding of elements.

The second part of the hypothesis states that it is possible to preserve links to the original TS. This was confirmed by applying the approach in our use cases. Binders were able to extract schemas, represent them as generic element trees, and introduce the back-links to original schema elements. Each binder uses a TS-specific notation to link the element of the generic tree to the original element. Binders also provide means for the interpretation of this notation during the execution of generated adapters. This is necessary as the mappings are created on the generic tree elements while the execution is performed over data files in the source TS by consulting original schema elements. Therefore, *our approach supports preserving links to the original elements by using a TS-specific notation for the identification of schema elements*.

For the reasons stated, we may conclude that **the Hypothesis 1 is confirmed with the limitation that our approach can be applied to the three-level meta-models with the graph structure that can be flattened into a tree without loosing any significant semantic properties**.

Hypothesis 2 states that *It is possible to create a relationship-based mapping language that allows the creation of high-level mappings between the uniform data schema representations, from which the data integration adapters can be generated*. While discussing Hypothesis 1 we have concluded that it is possible to represent data schemas in an uniform way, i. e., via a generic element tree, under some limitations. Our DSML allows creation of mappings between elements of source and target generic element trees. By creating a declarative mapping between two elements, a domain expert is specifying that there is correspondence between the elements. In the terminology of M2M transformations, such declarative transformations are called relationship-based transformations and thus *our DSML may be denoted as a relationship-based mapping language*.

The mappings that are created by using our tool are at a higher abstraction level than it is the case with the manual adapter code. Instead of working with variables, functions, classes, and other code constructs, a domain expert can focus on connecting source and target schema elements by using graphical links between elements called element mappings. Element mappings are created by drawing lines and, if necessary, by providing assignment expressions in Java. Unlike manual development, domain expert is using only a simple Java expressions to provide connection or value transformation from source to target elements. Therefore, we may conclude that *this relationship-based mapping language provides means for creation of high-level mappings* that are later used in the adapter generation process. Element mappings and their assignment expressions represent a sufficient input to a generator which is able to generate a fully functional adapter in a target platform. This is shown in use cases presented in this chapter.

Therefore, we may conclude that Hypothesis 2 may be fully confirmed as **we have created a relationship-based mapping language that allows the creation of high-level mappings between the uniform data schema representations, and used such specifications to generate fully functional integration adapters**.

Finally, Hypothesis 3 states: *It is possible to create an extensible reuse framework based on the created domain-specific integration language that will allow reuse of previously defined integration adapters in the presence of intra-space heterogeneity*. Most of the tools that we have surveyed in Chapter 4, did not provide the reuse at the level of mapping specifications. Just less then a third of the tools (28% (5/18)) provided this kind of the reuse. Only Karma (6% (1/18)) provided the reuse at the element mapping level while other tools provided reuse of entire mappings. As we have already argued, the largest increase in automation comes from the element mapping reuse and thus we have provided this mechanism in the AnyMap tool. As element mappings are created with our DSML, the reuse in our approach *is based on our DSML which can be seen as a domain-specific integration language*.

In addition to the mapping specification reuse, our approach and the tool support TS specification reuse as well. Our binders facilitate automatic creation of TS specifications from data files and as such they can be used in multiple integration scenarios whenever a particular TS needs to be imported into the tool. In this way, by reusing binders we are reusing TS specifications.

The notion of reuse framework from Hypothesis 3, in the context of our tool, relates to the automation engine and the reuse algorithm of the reuse module presented in Section 5.5. As we have presented in Section 5.4.2, the implemented reuse algorithm is based on a set of element matchers or matching algorithms as they are called. These matching algorithms calculate similarity between current generic tree elements and the generic tree elements used in previous mappings. Based on these similarities, probability is calculated that an existing element mapping can be applied to a current integration scenario. Matching algorithms, can be implemented in multiple ways: (i) matching algorithms based on isolated element information, (ii) matching algorithms based on element structure, and (iii) matching algorithms based on element semantic. We have currently provided several implementations of matching algorithms based on isolated element information. They rely on string comparison algorithms such as Levensthein and Jaro-Winkler algorithms. User can choose to run the reuse algorithm using one or multiple matching algorithms. As all matching algorithms implement the same interface, it is possible to add new algorithms to be used with the reuse algorithm without having to change the tool at all. The only limitation in the creation of matching algorithms is that the similarity between elements is calculated in the interval $[0, 1]$. These new matching algorithms will be just added to the list of all possible algorithms a user can choose from.

In addition to extending the reuse algorithm, automation engine is also extensible. It is possible to replace the current reuse algorithm with an alignment algorithm and still the output to be a set of element mapping candidates. This kind of flexibility allows developers of the tool to create a domain-specific reuse or alignment algorithms that are best fit for their respective domain. We can conclude that *the research framework comprising of automation engine and reuse and alignment algorithms is extensible*.

Finally, the created reuse framework is successfully applied in multiple use cases to achieve reuse of previously created mapping specifications, i. e., integration adapters. One of the use cases, in which intra-space heterogeneity is present, is presented in Section 6.1. By successfully applying the reuse framework, we can confirm that *it is possible to apply a reuse framework to reuse of previously defined integration adapters in the presence of intra-space heterogeneity*. Therefore, **we can fully confirm Hypothesis 3 as we have shown that our reuse framework is based on our DSML, can be extended with new algorithms, and can be successfully applied to create new integration adapters based on previous implementations**.

Besides discussing hypotheses, it is important to discuss the characteristics of our approach in the light of the main characteristics of DSMLs and MDSD. We will take into consideration the benefits and drawbacks of our approach based on the DSML benefits given in Section 2.2.2.

Benefits of using a DSML, such as *Domain Expert Involvement* and *Productivity and Quality*, are already addressed in the previous paragraphs while discussing the main goals of MDSD. In our approach, domain experts are not only included in the adapter development process, but they become developers themselves. Such inclusion of experts in the process shortens the development time by mitigating unnecessary communication and knowledge transfer between software developers and domain experts. This improves the productivity and quality of the solution in comparison to the manual adapter development. Both productivity and quality are also improved by reduction of accidental complexity. As the DSML provides concepts close to the domain that are easily understood by end users, it is easier for a domain expert to use such a language instead of a GPL. As the language is formally defined, domain-specific errors can be identified at the specification level instead of identifying them at runtime. Automatic generation of executable code reduces errors by lowering the impact of the human factor. As generators are developed by software development experts, generated adapters are optimized for an executable environment regardless of the integration expert who is specifying the mapping.

Productive Tooling is another benefit often associated with the usage of DSMLs. Providing a domain-specific programming language for solving domain-specific problems is just one element in achieving the productivity increase. Providing a useful and powerful tool or IDE even for a GPL can greatly increase the productivity of a developer. IDEs such as Eclipse, NetBeans [1] or InteliJ Idea [2], can greatly increase productivity of Java developers by providing context-aware assistance, help and on-demand documentation, debugging tools, intelligent error reporting, and automatic code completion. Although our prototype does not include many of these elements, in our future research we plan to develop context-aware error reporting and debugging, as well as simulation of transformations on a randomly generated data that conforms to source TS schemas. This is possible as the used DSML is close to the integration domain and all transformations are clearly identifiable, can be isolated, and errors can be easily connected to the right transformation in one place. Currently, our prototype

1 https://netbeans.org/
2 https://www.jetbrains.com/idea/

does include the implementation of automation engine and provides automatic application of identified mapping candidates to the current mapping. This way, the results of running automation algorithms can be applied without user intervention. We expect that this automation engine and automatic mapping creation will greatly increase the domain expert productivity as the algorithm becomes more precise and reliable with the growth of the reuse repository.

Platform Independence (Isolation) is supported in our approach by the means of code generators. Mapping specifications are created using our DSML while the executable adapters for desired platforms are generated from mapping specifications via code generators. We rely on the framework-based generation strategy. This strategy involves writing platform-specific and problem-agnostic code in the form of a framework, while the rest of the code, which is specific to the problem being solved, is entirely generated from the mapping specification. The core part of our framework is the binder implementation as it is in charge of reading and writing data. As we have developed the AnyMap tool, including binders, in the Java programming language, it was natural to reuse the same binders in the execution environment presented in Chapter 5. Although we have used the Spring framework as a basis of our microservice-based adapter code, generators for other Java frameworks can be easily added without having to change other modules of the tool. Furthermore, we are not limited to only using Java programming language with the currently implemented binders. We can use any other JVM-based language such as Groovy [3], Kotlin [4] and Scala [5]. These platforms are able to use our binders without any changes as they are all executed in the same environment. The next step in the adapter portability, to other execution platforms which are JVM-based, requires a bit more effort but far less than it is required in the manual development of adapters. In order to port a manually developed adapter to a C/C++–based or .NET-based execution environment for example, a developer has to rewrite every adapter in the appropriate programming language. In our approach, only new binders and generators have to be developed for the new platform while all mapping specifications would remain the same. This way, we have a large initial effort to develop new binders and generators. However, in the long run, with every new adapter automatically generated from the mapping specification this approach becomes more and more efficient than the manual rewriting of code.

Reduction of Execution Overhead is also provided through the existence of generators. As all adapters are entirely generated from the mapping specifications, every adapter is generated in a same way so as to satisfy some strict platform-specific requirements. Further, accidental introduction of non-optimized code and errors is prevented as manual interventions in code are minimized. We feel that our DSML comprises all the necessary concepts to provide enough expressiveness to the specified mappings in order for the adapters to be generated without the need for manual interventions. Having in mind that we have tested the approach and the DSML on a limited number of use cases, we plan to further apply it in new use cases and thus further evaluate the expressiveness of the language.

In addition to all the benefits we discussed, we need to discuss our approach in the light of the drawbacks (cf. Chapter 2) of applying MDSD and DSMLs. Some of the most frequent criticism of MDSD includes: *advertized MDSD approach not targeting all goals of the MDSD paradigm*, *insufficient tooling support*, *model rigidity and inflexibility*, and *focus on generation without enough customization* [32, 49, 50, 178].

The issue of an MDSD approach not targeting all goals of the MDSD paradigm cannot be considered as major disadvantage of our solution. As we have already shown, our approach is in full accordance with all the main goals of MDSD. Insufficient tooling support, as a frequent

3 http://groovy-lang.org/
4 https://kotlinlang.org/
5 https://www.scala-lang.org/

disadvantage of MDSD solution is evident in our case as we have provided only a prototype of the tool. Although the tool could use additional modules, our focus was on providing a tool that would support all the steps of the approach while focusing on productivity increase via reuse of existing mapping specifications. Simulation, debugging, and model-level error reporting modules will be developed as a part of our future research efforts.

By model rigidity and inflexibility, critics often state that models are not as flexible as the manually written code and thus they limit programmers' abilities. The main effect of such rigidness is that the more commonalities are hard-coded in the framework which is used in the generated code. We are aware of this drawback and of the fact that if a high-abstraction language is provided, such is our DSML, many details are implemented in the framework and user is limited in the amount of customization that can be done. This is the reason why we have opted to use pure Java language for the specification of assignment and filter expressions (i. e., our expression language) instead providing a separate DSML. By allowing a user to specify a part of the mapping in a language at a lower abstraction level, we aim to decrease the rigidness and inflexibility of mapping specifications.

Although we have introduced some flexibility by using Java as a part of the expression language, extensive customization of the generated code is not possible. It can be concluded that our solution is focused more on the code generation than on the customization. Our future plans encompass introduction of extension points in the execution engine. Using these extension points, developers would be able to fine-tune the generated artifacts and integrate manually written code with the generated one when necessary. Another future development task is to separate binders in separate schema and data manipulation parts. As data manipulation is performed at the execution level, by exposing it to developers, domain-specific optimizations could be introduced.

Other drawbacks are related to the drawbacks of the DSMLs in general. *Effort and Cost of Building a DSL* and *Cost of Evolution and Maintenance of a DSL* are present in the development process of any software tool and GPL language as well. From the language developer perspective, it is necessary to improve the language to represent all the changes in the integration domain. Furthermore, it is necessary to improve the appropriate tooling continuously as the language evolves as the tooling has a big impact to the productivity and quality of the solution. However, from the user perspective, these two characteristics have the least impact on his or her work. Of course, it is important to have tools that are maintained and improved. However, it is more important to have an appropriate tool that can be used to solve problems in the required domain at the required level of abstraction.

Language Engineering Skills are required when creating a DSML. For the end user of our DSML, these skills are not essential and therefore we consider this not to be a drawback of our approach.

Tool and Process Lock-in is often encountered in industry. The lack of interoperability between software tools leads to companies having to buy the entire software suite from a single vendor. We acknowledge that our tool may cause a tool lock-in for a company. Currently, there are integrations between our tool and other commonly used integration tools such are the ones presented in Chapter 4. The main reason of this lack of interoperability is the difference in the underlying structures, level of abstractions, and the way in which the mappings are created. All of these things make the integration of the tools hard and the benefits of providing such adapters may prove not to be cost-effective over time. On the other hand, our approach is pretty much flexible in terms of tools, languages, and process steps that are taken in the adapter definition. In its current form, our approach offers multiple ways of defining adapters, manual and semi-automatic, and introduces many byproducts on top of which new process steps can be added and the entire approach to be extended or adapted. Therefore, our opinion is that our approach is less probable to introduce a process lock-in as it provides domain experts with means to develop integration adapters and an option to easily adapt the approach to their specific needs and problems.

Our tool was developed partly based on our previous experience in the integration domain and partly based on the common characteristic of surveyed tools in Chapter 4. We have chosen to build our tool based on these characteristics as we wanted to provide users of these tools a familiar look and feel when switching to our tool. Additionally, these common characteristics and functionalities are the ones most developers are used to. The main advantage our tool has over the surveyed tools, is the implementation of the automation engine and reuse at the level of element mappings. This can greatly improve the specification speed. However, as it is already said, the reuse algorithm will be more precise and reliable if the mapping repository contains more mappings from appropriate integration scenarios.

As the front-facing part of our tool, graphical syntax is the first thing users are presented with. We chose the graphical concrete syntax as it is the most common concrete syntax among surveyed tools. We have intentionally tried to keep it as simple as possible and yet intuitive enough in order to provide a good learning curve. Such a graphical syntax has it strengths in providing the best overview of the mapping specification compared to other concrete syntaxes. This can be observed in our tool as well. However, for mapping specifications with a large number of graphical elements, a diagram may be too crowded and the users' ability to work efficiently may be hindered. For that reason we plan to develop other concrete syntaxes for our DSML. The user will be able to use them simultaneously and take advantage of all positive aspects of each syntax.

6.4 SUMMARY

In this chapter we have presented two use cases in which we applied our integration approach and the AnyMap tool. Our goal was to show that the approach may be used in different integration scenarios where the integration is not performed at the same abstraction level. In the first use case we have integrated systems at the level of exchanged messages, i. e., at the data level. On the other hand, in the second example we have used our approach to facilitate the exchange of high-level business models between incompatible meta-modeling environments. Through these examples we have focused on different aspects of the approach. First example was selected in order to represent usage of different binders and the reuse algorithm. In the second example our focus was on the approach application in an existing tool and framework landscape. We have used AnyMap to create abstractions between schemas of a higher abstraction level. The focus was also on the plug-and-play nature of the tool, which allows for different generators to be used as we have used ETL generator instead of the commonly used Microservice generator from the first example. Due to all aforementioned reasons, overviews of use cases that are presented in Figure 26 and Figure 36 have a different architectural representation of the AnyMap tool.

The first use case involves integration of industrial sensors that send measurement data and ISs that visualize the data. Sensors send measurement data in the form of *CSV* documents while ISs can only read XML documents that are formatted according to a predefined schema. In order to bridge the gap caused by the inter-TS heterogeneity between the two TSs, we have created a mapping between source and target data schemas. These mappings were manually created using the AnyMap tool. The code generators were then applied to generate a microservice that executes transformations from received CSV documents to XML documents according to the transformation rules described in the specified mappings. The produced microservice can be instantiated and run multiple times in order to achieve more scalable transformation solution that allows greater throughput of data. This is especially important for the real-time or near real-time systems that are often encountered in industry.

As a part of the same use case, we also solved an intra-TS heterogeneity issue that is often encountered in the manufacturing industry. Different measurement methods or different sensor configurations often introduce significant changes to the format of a CSV document. Nowadays, these changes would cause manual interventions in the integration adapter code. The number of these code interventions grows exponentially as the more changes are introduced to the CSV file. In this integration scenario, all changes to the sent data are in fact string-based changes. The names of CSV columns are similar for the columns that have a similar semantic. To solve the intra-TS heterogeneity issue we have utilized AnyMap's reuse module in the semi-automatic mode. The AnyMap tool automatically provided element mapping candidates to the user by using the reuse algorithm which was in fact based on string matching algorithms. After selecting the most appropriate candidates and after some small manual interventions in the mapping specification, an altered transformation microservice was generated. No manual interventions at the code level were required.

In the second use case we have shown that our approach can be used together with M3Bs to provide external import and export mechanisms for the MetaEdit+ and Microsoft Visio meta-modeling tools. Unlike the first use case in which the mappings were specified directly between source and target TSs, in this use case we use a mediatory EMF technical space. All meta-models and models are first transformed to the EMF technical space by using appropriate M3Bs. AnyMap is used for manual mapping creation between the elements of source and target meta-models once everything is set up in the EMF technical space. The ETL generator is then used to create model-to-model transformations which are executed also in the EMF technical space.

Based on the successfully transformed data and models in the aforementioned two use cases, we have shown that our approach is applicable in various integration scenarios. Furthermore, our approach does not depend on the abstraction level of the original structures that are being mapped. Based on the successfully and completely generated Visio model in the second use case we have shown that our approach can be combined with other approaches, Java-based libraries, and frameworks in order to achieve easier integration. The graphical and declarative nature of the mapping language also gives a good overview and visualization of the created mappings compared to the textual Java transformations that are necessary in the first use case or textual ETL transformations from the second example.

Our approach could be also applied to many other industrial and non-industrial use cases. For example, ECTL processes in the domain of databases could be specified using our approach. As these processes aim at gathering data from various data sources formatted in different ways, an ECTL process could be seen as the integration of these various TSs from data sources on one side and relational TSs of a relational database on the other. In addition to ECTL processes, a notable application would be in the ontology alignment process in which the alignments can be specified using a mapping language. This is especially needed in the domain of health-related information systems, more specifically for the exchange of Electronic Health Record data. We have identified this need in [55] and will further investigate it in the future.

It is worth noting that it could be beneficial to use a semantic-based matcher in the industrial example as it would allow the AnyMap tool to find similarities between different schema elements that are not so similar in terms of string similarity. This would allow a developer to find a similarity between *Weight* and *Thickness* or *Weight* and *Radius* as they all represent physical dimensions of semiconductor wafers and there is a high probability that they need to be visualized in a similar way. One option is to create a semantic matcher that uses a synonym dictionary like WordNet [6] or a domain-specific synonym dictionary tailored for a specific application domain. One of our planned future results is to provide new and improved matchers that can increase precision and provide more domain-specific reasoning to the reuse algorithm.

In this chapter we have also provided a structured discussion about hypotheses introduced in Chapter 1. We have disseminated evidence that both main hypothesis (Hypothesis 0) and two of three derived hypotheses (Hypothesis 2 and Hypothesis 3) are fully confirmed. The last derived hypothesis (Hypothesis 1) that states: "*It is possible to represent data schemas (i. e., meta-models) from the three-level technical spaces in a uniform way by using a graph-like representation, while preserving links to original elements*" is confirmed but with the observed limitation of our approach. The limitation is that the approach can be applied only to the three-level technical spaces in which meta-models with the graph structure that can be flattened into a tree without loosing any significant semantic properties. This is a limitation of our approach introduced by the choice of technological environment in which the AnyMap tool is built. Our opinion is that this limitation will only affect a small number of technical spaces and that our tool may be applied in a large majority of industrial use cases. In our future research efforts, we plan to improve our tool and the approach in order to avoid this limitation.

We have also seen that our approach is in full accordance with the main goals of MDSD and that the mapping language is considered as a DSML. As such, our approach can have all the benefits and drawbacks associated with these two notions. We argued that all the benefits will be applicable to the approach while the main disadvantages are the model rigidness and lack of customization of the generated solution that would allow synergy between generated and manually written code. We plan to address these drawbacks in the future. Also, one of our future research directions will be to establish metrics and evaluation framework. With this framework we will try to quantify our statements from this chapter and prove that all the benefits are applicable to our approach.

6 https://wordnet.princeton.edu/

7 CONCLUSION AND FUTURE WORK

In this thesis we have presented research aimed at solving issues of inter-space and intra-space heterogeneity. These issues often occur during the integration of different Technical Spaces (TSs) in the context of Industry 4.0 and Industrial Internet of Things (IIoT). They hinder productivity and efficiency of a manufacturing system as manual development of custom integration adapters is required. Such a manual development process is usually time-consuming, error-prone, and costly.

IIoT TSs, which need to be integrated, often have a three-level architecture that comprises data, data schema, and meta-schema levels. The integration adapters are implemented by taking into consideration both data and schema levels and then programming executable data transformation functions. An adapter developer must tackle two kinds of complexity: (i) complexity introduced by the integration scenario as well as (ii) complexity introduced by the chosen GPL which is not specifically tailored for the integration adapter development. First type of complexity is inherent to the integration of TSs and cannot be avoided. On the other hand, the second type of complexity, accidental complexity, can be eliminated by providing appropriate domain-specific languages, tools, and by raising the abstraction level at which the adapter is specified.

At the beginning of this research, we have established one main and three derived hypotheses from which the main goals and expected results of the thesis were formulated (cf. Chapter 1). In order to confirm the main hypothesis, Hypothesis 0: *It is possible to solve heterogeneity problems in TS integration by creating appropriate DSMLs and following the principles of MDSD approach*, we have developed and applied an original methodological approach to the integration of TSs and the accompanying AnyMap tool. These hypotheses led to the formulation of the main goal of our research: to provide a structured, automated, and reusable integration of TSs. Another research goal was to show that such an approach can be both theoretically established as well as practically implemented and applied in real-world use cases. To better disseminate the results of our research, we have divided the contributions of our work into three categories: theoretical, development, and application contributions.

Theoretical contributions. The first step of the approach specification was to survey current integration approaches in order to identify main concepts and methods that are used for the integration of TSs. Although we have not identified any existing MDSD approaches, performing literature and integration tool surveys provided us with some valuable ideas and insights in the contemporary machines, information systems, and TSs integration strategies. Based on these findings and our own experience in the domains of MDSD and integration of TSs, we have defined an original methodological approach to specification of integration adapters.

The established approach comprises three main phases. In the first phase, a data schema from a TS is translated into a generic data schema named the generic element tree. This generic element tree representation was created based on the observations of existing meta-meta-models, schema definition languages, and integration tools. Based on our use cases and several studies, we have concluded that the tree representation is appropriate for the representation of most data schemas from three-level TSs in the IIoT domain. Therefore, we can state that the Hypothesis 1: *It is possible to represent data schemas (i. e., meta-models) from the three-level technical spaces in a uniform way by using a graph-like representation, while preserving links to original elements*, is confirmed but with the observed limitation of our approach. The limitation is that the methodological approach introduced in this thesis can be applied only to the three-level meta-models with the graph structure that can be flattened into a tree without loosing any significant semantic properties. This is a limitation of our approach introduced by the choice of technological environment in which the AnyMap tool is built. Our opinion is that this limitation will only affect a small number of TSs and that our tool may be applied in a large majority of IIoT use cases.

In the second phase of our approach, a DSML is used in order to create mappings between source and target generic tree structures. We have developed the DSML by following the principles of language development defined in the MDSD methodology. The language comprises two parts, i. e., sub-languages: (i) the mapping language used for creation of the mapping specifications, and (ii) the expression language used for influencing the execution of transformations and providing fine grained mapping specification. The concepts that we have identified as the most essential for the mapping specification are represented in a form of a meta-model that conforms to the Ecore meta-meta-model. In this way, we have also specified the abstract syntax of the DSML. In addition to the abstract syntax, we provided the graphical concrete syntax for the DSML as it is the most suitable one for the task of mapping creation. This DSML is created so as to provide confirmation of the Hypothesis 2 which states: *It is possible to create a relationship-based mapping language that allows the creation of high-level mappings between the uniform data schema representations, from which the data integration adapters can be generated*.

Although the DSML can be used for the manual creation of mappings, our approach also allows the automatic creation of mappings. There are two types of algorithms developed to support the automation of the specification process: (i) alignment algorithms, that try to find correspondences between the source and the target TSs directly without taking into consideration previously created mappings, and (ii) reuse algorithms, that provide element mapping candidates based both on the current combination of TSs and the previously created mappings. Both of these algorithms were created with a goal to confirm Hypothesis 3: *It is possible to create an extensible reuse framework based on the created domain-specific integration language that will allow reuse of previously defined integration adapters in the presence of intra-space heterogeneity*.

In the third phase of the approach, an executable adapter is generated based on the created mapping specification. The mapping specification is created at the schema level but the generated adapter transforms data files that conform to the source schema into target data files that conform to the target data schema.

Development contributions. With the approach we have laid down the theoretical foundations for the creation of the AnyMap tool. AnyMap is developed using the programming language Java with the notable exception of generators which are implemented in Xtend. The tool comprises implementation of all aforementioned elements: TS importers (binders), DSML, alignment and reuse algorithms, and code generators. Each of these elements is implemented in a form of a separate module of the tool. The modules are:

- *The Core module* comprises essential components which are used in all other modules. It contains concepts of the mapping language, expression language, and interfaces for the implementation of binders and generators. All interfaces, which need to be implemented in order to extend one of the other modules, are located in the Core module.

- *The Binding module* contains plug-ins which represent TS binders used for importing and exporting data and data schemas of a specific TS. For each new TS that needs to be supported by the AnyMap tool, a new binder must be developed in a form of a new plug-in.

- *The Mapping Editor module* is the main interaction point between a user and the AnyMap tool. This module provides the graphical concrete syntax for the mapping language, textual concrete syntax for the expression language, and all necessary GUI classes (event listeners, commands, and menu items) needed to provide interaction between the user and the tool. Both mapping and the expression languages are implemented as a part of this module.

- *The Reuse module* comprises multiple plug-ins that constitute the automation engine with reuse and alignment algorithms. This module is in charge of reading and writing to the reuse repository and coordinating the calculation of element mapping candidates.
- *The Generator module* comprises plug-ins for the generation of executable transformations for a desired execution environment. The task of each generator plug-in is to parse a mapping file, extract necessary mappings, and generate executable transformation code based on these mappings.

In addition to the AnyMap tool, we have also developed an execution environment in which adapters are executed. Although it is possible to create generators that will generate regular, stand-alone Java or C adapters, our opinion is that we could benefit from implementing adapters in the form of a microservice architecture. In our integration tool survey, we have noticed that most of the tools either provide built-in execution of the specified mappings or generate adapters that can be run in a single-machine non-scalable environment. It is up to the user to support scaling of such adapters in the distributed environment. Another noticeable characteristic of adapters is that they usually do not have any internal state storage. To be more precise, adapters receive data, transform it, and provide transformed data at the output. As such, adapters can be considered as isolated pieces of code that can be independently scaled. The execution environment comprises many microservices which are autonomous execution units created to preform a single task. Therefore, in addition to the infrastructure microservices such as registration and discovery services, each adapter is represented as a single microservice that can be instantiated as many times as it is needed to respond to all requests for data transformation. The entire execution environment is written in Java by using the Spring framework and Spring Cloud library.

Application contributions. In this thesis, we have presented the application of our approach in two representative use cases: one industrial and one non-industrial. In the industrial use case, we have achieved integration between a sensor machine and an information system. The machine operates in the CSV TS while the information system can receive data from the XML TS. In the same use case we have tested our reuse algorithm and provided fully automatic creation of mappings in the case of an intra-space heterogeneity. The second use case provided interoperability between Microsoft Visio and MetaEdit+ meta-modeling environments and showed that the approach can be used in more than industrial cases as long as the three-level TS architecture constraint is satisfied. Application of our approach and the tool resulted in a adapter that allowed exchange of models between the environments which was not possible previously.

With all of the aforementioned contributions taken into account, Hypothesis 0, Hypothesis 2, and Hypothesis 3 are fully confirmed while Hypothesis 2 is confirmed but with the limitation as it is already mentioned. Also, we may state that all introduced goals and expected results of this research are met.

7.1 FUTURE WORK

In this section, we present possible direction of future research. We divide the presented future work into three categories: (i) future research in the domain of model-driven system integration, (ii) further developments of the AnyMap tool, and (iii) new application domains. In the following subsections, we present each of these categories and their elements in detail.

7.1.1 *Future research in the domain of model-driven system integration*

Although the reuse and alignment processes presented in this paper represent a significant step towards increasing the automation of the mapping specification process, more advanced automation algorithms are needed in order for them to be ready for continuous industrial application. Therefore, one of the main research issues that needs to be addressed is how to further increase the degree of the adapter development automation.

One of the possible future research directions is to identify more advanced and more efficient element matching algorithms. Currently, automation is based on simple matching algorithms that calculate element similarity and probability based on information about isolated elements. These algorithms do not take the context or relationships of an element into consideration when calculating the probability of it being the right candidate for a new mapping. One possible improvement on this would be to semantically describe TSs with a common ontology or a set of ontologies and then find semantic correspondences between source and target elements in a more automatic manner by using existing ontology alignment methods. Another possible solution would be to provide a single, unified ontology, designed specifically for the IIoT domain. Using this ontology, all TSs from the IIoT domain could be represented with this ontology and thus all TSs would be assigned a meaning. Therefore, finding the correspondence between elements on the source and target sides can be done by assigning each element with the meaning in the same ontological space.

However, it is really hard to identify a single ontology that would cover all the possible elements found in the very wide IIoT domain. Another solution, probably easier to implement, is to represent each TS with a separate ontology and then, by using existing ontology alignment tools, align different ontological elements. As we have presented in [55] it is hard to have an ontology alignment process when there are too many ontological representations of the same SUS. However, if these issues would be solved, i. e., there is a single referent source of appropriate ontologies, the degree of mapping automation would be greatly increased. Although this method resembles pure element matching we described in this thesis, ontologies would provide semantically rich information about TS elements and thus increase the quality of the mapping too.

Another way of improving automation algorithms is to consider the element structure in addition to isolated element information. Similar patterns in the source and target structures may be indicators of element similarity. This approach, also known as graph matching [138] combined with the approach to identify similarities based on their names using the synonym thesauri such as WordNet, could also lead to an increase in mapping automation. Together with the repository searching (cf. Section 5.4), the synonym thesauri would be used as a initial step of the automation process. In this step, similar source and target elements can be identified. Once these elements are identified, the search can be widen around these elements in order to spot similar patterns and deduce new correspondences. This way, the algorithm would consider both the element itself and the context in which the element exists.

Automation algorithms described in this thesis are just one way of increasing the degree of automation. Another option is to create a full-fledged hybrid recommendation system [165] which would create a matrix of current TS elements and previously stored repository rules and recommend best fitting candidates. Our current solution resembles a collaborative recommendation engine, but lacks considerations of result labeling, diversity, quality, and trust factors. In such a collaboration engine, the collaborative element filtering would be in charge of finding appropriate recommendations based on previous mappings that included similar target and source elements to the current use case. In contrast to this, a hybrid recommendation system would comprise both collaborative and content-based element filtering. The content-based filtering would consider properties of the elements being mapped and find the recommendations purely based on the relevant element properties regardless of the previous mappings. The science behind recommendation systems has a research field on its own [165] and the implementation of such a system would require blending recommendation engine knowledge, data science, and system integration experience as well.

In addition to the improvements of reuse, alignment, and matching algorithms, change in the way in which binders operate can improve the integration process. Currently, binders require existence of data schemas in order to construct a generic tree representation of the technical space. Some of the binders, like the CSV binder, are also equipped with a GUI that allows manual specification of the generic element tree. Another future research direction is to investigate the possibility of schema inference from real-time streams of data and automatic adaptation of adapters based on the internal schema changes. First of all, based on the data samples taken from a stream of data, initial schema can be inferred. This is not a new research and has been popular in the XML and RDBMS domains for some time, e. g., [29, 93]. However, these algorithms often consider static data files, that are complete at the point of schema inference. These algorithms can be further adapted to the domain of IIoT where the data is not so verbose, comes from a real-time data stream, and where the efficiency of messaging protocols is of high importance. Once the initial schema is inferred based on enough received data, the mapping can be created. However, new incoming data can reveal new schema elements that were previously undiscovered on the limited data sample. Thus, schema inference algorithms can be applied again on the real-time stream in production in order to capture schema changes, and if possible, adapt the mapping and the generated adapter. This would ease the developers' job as only the breaking schema changes would require interventions in mapping specifications. Further, this adaptation step would be even easier with the appropriate tooling support which is able to show the breaking schema differences and possibly suggest mapping changes.

Future research efforts can also be directed towards the identification of the most efficient way of representing generic element trees and mapping specification. This research spans both system integration and human-computer interaction domains. Our choice of a tree based representation of generic element trees and a graphical syntax for the mapping language was influenced by our experience with all the surveyed tools and previous experience in the IIoT integration domain. Although the tree-based representation can be sufficient for the most IIoT use cases, with three-level TS architecture, in other non- IIoT use cases a fully manipulative graph representation would provide more flexibility and control. Although the graphical syntax gives a good overview of the entire mapping, in the presence of many operators and links, the diagram can become overcrowded and thus the understandability would be hindered. Multiple options exist in order to alleviate this problem. Developing a combination of graphical and some other syntaxes, like tabular, symbolic, or textual, could provide a solution to the aforementioned problem. Further, improvement of the existing graphical syntax could also be a solution if a smart, context-aware presentation mechanism is applied. Such an mechanism would put the focus only on mappings of interest. In either case, a right balance of granularity, syntax type and control needs to be identified from the human-computer interaction standpoint.

Also, one of our future research directions will include establishing metrics and evaluation framework for qualitative and quantitative evaluation of our approach. Evaluating an MDSD approach and measuring DSML quality are well-known issues in the research community and only a limited number of research work are done to this date. Most notable works in this area include [22, 100] in which authors provide several approaches to evaluating DSLs. These works can be a baseline for the creation of such an qualitative evaluation framework. However, a mandatory extension is to introduce quantitative metrics for measuring generated code performance and resource usage efficiency. With this framework we will try to quantify our statements and the evaluation of our approach from Section 6.3. The framework should be able at least to measure the efficiency of the solution compared to the manual development, domain coverage by DSML concepts, domain expert satisfaction, and performance of generated code artifacts. The result of such an evaluation will be also used to prove or disprove that all the benefits identified in that section are applicable to our approach. As the field of quantitative evaluation of DSMLs is at early stages, such metrics and a framework could be extended so as to provide contribution to the general DSL evaluation.

7.1.2 *Further developments of the AnyMap tool*

In addition to the future development of the AnyMap tool that is caused by the research presented in the previous subsection, several other improvements are necessary in order to improve tool usability, efficiency, and domain coverage.

Development of new TS binders would allow for the tool to be applied in more use cases. This would also increase the number of mappings in the reuse repository and indirectly allow for a better mapping automation by reuse. We plan to develop binders for the most popular protocols in the IIoT domain such are OPC UA, Message Queue Telemetry Transport (MQTT), and Modbus. In addition to the currently supported SECS/GEM, CSV and XML, this will provide enough coverage for many use cases encountered in the industrial setting.

There are some possible improvements in the binding process as well. The generic element tree can be enriched with additional meta-data (e. g., whether an element is mandatory) in order for users to have more information available to them when creating mappings. Next, there is a need for a better separation of the schema binding and data binding. The former is used in the initial process of creating the generic element tree, and the later is used at run-time while reading and writing data during the transformation execution. Once these binder parts are separated into schema binder, data reader and data writer, binder libraries could have smaller memory footprint as it could be possible to use only the most necessary interfaces for a specific use case. Additionally, binders could be optimized better and it would be possible to recognize whether a TS has a binder for writing or it just allows reading from a TS. This way, some TSs could be considered only at a source side and some other could be also present at the target side of the mapping.

There is also a need to provide more generators and execution environments. Although the microservice transformation environment that is already implemented can provide necessary flexibility and scalability, in some real-time industrial settings the speed of transformation execution is of utmost importance. Therefore, we plan to develop a generator of the optimized adapters in C or C++ programming languages, that will be executed directly on the industrial PC's to which devices are connected. Such an adapter can be executed near machines to facilitate real-time or near real-time transformation of data. It is worth mentioning that the optimization of such adapters is an important feature as they often run on limited resources.

Manual customization of the generated code is necessary in order to increase the flexibility of our approach. We plan to introduce extension points in the execution engine. Using these extension points, developers would be able to fine-tune the generated artifacts and integrate manually written code with the generated one when necessary. Also, by introducing extension points in binders' data read and write modules, domain-specific optimizations could be introduced.

In order to increase the tool support and thus further increase the productivity of domain experts, new modules will be added. We plan to develop simulation, debugging, and model-level error reporting modules. Model-level error reporting will notify users of possible errors at the level of mapping specifications. Such errors can include missing assignment expressions, unsatisfied element mapping dependencies, and transformation cycle detection. Debugging module will provide debugging tools to the domain expert while simulation module will enable test runs of the specified mappings. We plan to provide a generation of dummy data files based on source schemas that could be used by the simulation module.

We plan to use the AnyMap tool as a core part of a bigger integration platform that will encompass other components such are data analysis component, alerting component and anomaly detection component. The AnyMap tool would be used as a modeling tool for connecting different parts of the system such as anomaly detection component to an alerting system, etc. Furthermore, generated adapters could be expanded to comprise a small data storage component called a buffer zone. Buffer zones would provide temporary data storage in order to move the analysis closer to the device (edge analysis) instead of the current analysis in the cloud. This would allow for the fast detection of anomalies and near real-time reporting on the device status. Once the buffer is full, a system would move the data to the cloud and free some space in the buffer zone.

7.1.3 *New application domains*

Creating adapters and providing interoperability between meta-modeling environments are not only use cases in which the AnyMap tool is applied. As an important IoT sub-field, *system monitoring* enables observation of the system and recognition of incidents that can occur in it. Instead of monitoring and analyzing data from individual sensors, an incident management system can takes into consideration the underlying processes as well. This way, the proposed system tries to identify possible causes and effects of an incident in order to identify an appropriate strategy to handle it. The AnyMap tool can be used to specify initial training set for machine learning algorithms by creating mappings between identified causes and effects of an incident. The training set would connect a list or a database of known incidents in a system, and a list or database of known consequences. This way, initial knowledge would be provided to the incident management system.

Incident management systems are often integrated with alert systems. The combination of both systems should cover complex incidents and significantly improve the incident management and reporting ability. This increases the quality of IoT solutions, reduces costs, and minimizes risks by facilitating fast reactions to incidents. The AnyMap tool can be used to facilitate connectivity by creating mappings between different incidents and alerts provided by both systems. AnyMap flexibility and reuse would be beneficial as there are many manufacturers of these systems that use different data formats and APIs.

In addition to IIoT and IoT applications, AnyMap could be applied in the domain of ontology alignment. In this domain, alignments can be specified using the mapping language. This is especially needed in the domain of health-related information systems, more specifically for the exchange of Electronic Health Record data. We have identified this need in [55]. The main healthcare ontologies today are defined using a number of different formats. Therefore, writing an automated tool that would work between any two ontologies is a challenging task due to the number of possible combinations. We think that in order to solve this problem a common ontology format should be used. Our stance is in accordance with the

Yosemite initiative [83] that proposes OWL/RDF as a common ontology representation format. The Yosemite initiative also suggests a two-step approach to healthcare ontology integration: (i) transforming any ontology format to OWL/RDF and (ii) creating an integration algorithm for two OWL/RDF ontologies. We think that following such an approach to integration will lead to simplifying the currently complicated field of ontology alignment and that our approach and the AnyMap tool can be used in both of the aforementioned steps. The first step comprises creation of mappings between a desired ontology and its OWL/RDF representation while the second step comprises creation of mappings between two OWL/RDF ontologies. By using our approach and the tool, these mappings can be created once the appropriate TSs are imported. By using the automation engine, the entire process can be sped up.

REFERENCES

[1] Adeptia. Adeptia ETL tool, 2016. URL https://adeptia.com/solutions/ETL-software-for-data-transformation.html.

[2] Santa Agreste, Pasquale De Meo, Emilio Ferrara, and Domenico Ursino. XML matchers: approaches and challenges. *Knowledge-Based Systems*, 66:190–209, 2014.

[3] Henning Agt, Gregor Bauhoff, Mario Cartsburg, Daniel Kumpe, Ralf Kutsche, and Nikola Milanović. Metamodeling foundation for software and data integration. In *Information Systems: Modeling, Development, and Integration*, volume 20 of *Lecture Notes in Business Information Processing*, pages 328–339. Springer, Berlin, 2009. ISBN 978-3-642-01112-2.

[4] Slavica Aleksić. *An SQL Generator of Database Schema Implementation Scripts in IIS*Case Tool*. Master Thesis, University of Novi Sad, Novi Sad, 2006.

[5] Slavica Aleksić. *Methods of Database Schema Transformations in Support of the Information System Reengineering Process*. PhD thesis, University of Novi Sad, Novi Sad, 2013.

[6] Bogdan Alexe. *Interactive and Modular Design of Schema Mappings*. PhD thesis, University of California at Santa Cruz, Santa Cruz, CA, USA, 2011.

[7] Bogdan Alexe and Wang-Chiew Tan. A New Framework for Designing Schema Mappings. In Val Tannen, Limsoon Wong, Leonid Libkin, Wenfei Fan, Wang-Chiew Tan, and Michael Fourman, editors, *In Search of Elegance in the Theory and Practice of Computation*, number 8000 in Lecture Notes in Computer Science, pages 56–88. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-41659-0 978-3-642-41660-6. DOI: 10.1007/978-3-642-41660-6_4.

[8] Bogdan Alexe, Laura Chiticariu, Renée J. Miller, and Wang-Chiew Tan. Muse: Mapping understanding and design by example. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 10–19. IEEE, 2008.

[9] Bogdan Alexe, Wang-Chiew Tan, and Yannis Velegrakis. STBenchmark: towards a benchmark for mapping systems. *Proceedings of the VLDB Endowment*, 1(1):230–244, 2008.

[10] Bogdan Alexe, Balder TEN Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Characterizing schema mappings via data examples. *ACM Transactions on Database Systems (TODS)*, 36(4):23, 2011.

[11] Bogdan Alexe, Balder Ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Designing and refining schema mappings via data examples. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, pages 133–144. ACM, 2011.

[12] Bogdan Alexe, Balder Ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. EIRENE: Interactive design and refinement of schema mappings via data examples. *Proceedings of the VLDB Endowment*, 4(12):1414–1417, 2011.

[13] Bogdan Alexe, Mauricio Hernández, Lucian Popa, and Wang-Chiew Tan. MapMerge: correlating independent schema mappings. *The VLDB Journal*, 21(2):191–211, April 2012. ISSN 1066-8888, 0949-877X. doi: 10.1007/s00778-012-0264-z.

[14] Altova MapForce. User & Reference Manual. User Manual, 2013.

[15] Analytix DS. Analytix Mapping Manager, 2016. URL http://analytixds.com/amm/.

[16] Fatima Ardjani, Djelloul Bouchiha, and Mimoun Malki. Ontology-Alignment Techniques: Survey and Analysis. *International Journal of Modern Education & Computer Science*, 7(11), 2015.

[17] Colin Atkinson and Thomas Kühne. Model-driven development: a metamodeling foundation. *Software, IEEE*, 20(5):36–41, 2003.

[18] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, October 2010. ISSN 13891286. doi: 10.1016/j.comnet.2010.05.010.

[19] David Aumueller, Hong-Hai Do, Sabine Massmann, and Erhard Rahm. Schema and ontology matching with COMA++. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 906–908. ACM, 2005.

[20] Radhakisan Baheti and Helen Gill. Cyber-physical systems. *The impact of control technology*, 12:161–166, 2011.

[21] Jelena Banović. *An Approach to Generating Executable Information System Specifications*. PhD thesis, University of Novi Sad, Novi Sad, 2010.

[22] Ankica Barišić, Vasco Amaral, and Miguel Goulão. Usability Driven DSL development with USE-ME. *Computer Languages, Systems & Structures*, July 2017. ISSN 1477-8424. doi: 10.1016/j.cl.2017.06.005. URL http://www.sciencedirect.com/science/article/pii/S1477842417300477.

[23] Zohra Bellahsene, Angela Bonifati, Erhard Rahm, and others. *Schema matching and mapping*, volume 57. Springer, 2011.

[24] Sonia Bergamaschi, Silvana Castano, Maurizio Vincini, and Domenico Beneventano. Semantic Integration of Heterogeneous Information Sources. *Data Knowl. Eng.*, 36(3):215–249, March 2001. ISSN 0169-023X. doi: 10.1016/S0169-023X(00)00047-1.

[25] Jacob Berlin and Amihai Motro. Autoplex: Automated Discovery of Content for Virtual Databases. In Carlo Batini, Fausto Giunchiglia, Paolo Giorgini, and Massimo Mecella, editors, *Cooperative Information Systems*, number 2172 in Lecture Notes in Computer Science, pages 108–122. Springer Berlin Heidelberg, September 2001. ISBN 978-3-540-42524-3 978-3-540-44751-1. DOI: 10.1007/3-540-44751-2_10.

[26] Philip A. Bernstein and Laura M. Haas. Information integration in the enterprise. *Communications of the ACM*, 51(9):72–79, 2008.

[27] Philip A. Bernstein, Sergey Melnik, Michalis Petropoulos, and Christoph Quix. Industrial-strength schema matching. *ACM SIGMOD Record*, 33(4):38–43, 2004.

[28] Philip A. Bernstein, Jayant Madhavan, and Erhard Rahm. Generic schema matching, ten years later. *Proceedings of the VLDB Endowment*, 4(11):695–701, 2011.

[29] Geert Jan Bex, Frank Neven, and Stijn Vansummeren. Inferring XML schema definitions from XML data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 998–1009. VLDB Endowment, 2007.

[30] Kris Bledowski. The Internet of Things: Industrie 4.0 vs. the Industrial Internet, July 2015. URL https://www.mapi.net/forecasts-data/internet-things-industrie-40-vs-industrial-internet.

[31] Jonathan Bowen. *Getting Started with Talend Open Studio for Data Integration*. Packt Publishing Ltd, 2012.

[32] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, San Rafael, 2012. ISBN 978-1-60845-883-7 1-60845-883-0.

[33] Lisa M. Braz. Visual Syntax Diagrams for Programming Language Statements. In *Proceedings of the 8th Annual International Conference on Systems Documentation*, SIGDOC '90, pages 23–27, New York, NY, USA, 1990. ACM. ISBN 978-0-89791-414-7. doi: 10.1145/97426.97987.

[34] Frederick P. Brooks. *The mythical man-month*. Addison-Wesley Reading, MA, anniversary ed. edition, 1995. ISBN 0-201-83595-9.

[35] Fabian Büttner, Ullrich Bartels, Lars Hamann, Oliver Hofrichter, Mirco Kuhlmann, Martin Gogolla, Lutz Rabe, Frank Steimke, Yorck Rabenstein, and Alina Stosiek. Model-driven standardization of public authority data interchange. *Science of Computer Programming*, 89:162–175, September 2014. ISSN 01676423. doi: 10.1016/j.scico.2013.03.009.

[36] Peter Buxmann, Thomas Hess, and Rainer Ruggaber. Internet of Services. *Business & Information Systems Engineering*, 1(5):341–342, September 2009. ISSN 1867-0202. doi: 10.1007/s12599-009-0066-z.

[37] Jean Bézivin. In search of a basic principle for model driven engineering. *Novatica Journal, Special Issue*, 5(2):21–24, 2004.

[38] Jean Bézivin. Model driven engineering: An emerging technical space. In *Generative and transformational techniques in software engineering*, volume 4143 of *Lecture Notes in Computer Science*, pages 36–64. Springer, 2006. ISBN 978-3-540-46235-4.

[39] Jean Bézivin and Olivier Gerbé. Towards a precise definition of the OMG/MDA framework. In *Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on*, pages 273–280. IEEE, 2001.

[40] Jean Bézivin and Ivan Kurtev. Model-based technology integration with the technical space concept. In *Proceedings of the Metainformatics Symposium*. Springer-Verlag, 2005.

[41] S. Castano and V. De Antonellis. Global viewing of heterogeneous data sources. *IEEE Transactions on Knowledge and Data Engineering*, 13(2):277–297, March 2001. ISSN 1041-4347. doi: 10.1109/69.917566.

[42] James Clark. XSL Transformations (XSLT). Recommendation, World Wide Web Consortium (W3C), 1999. URL https://www.w3.org/TR/xslt.

[43] Chris Clifton, Ed Housman, and Arnon Rosenthal. Experience with a combined approach to attribute-matching across heterogeneous databases. In *Data Mining and Reverse Engineering*, pages 428–451. Springer, 1998.

[44] Isabel F. Cruz, Flavio Palandri Antonelli, and Cosmin Stroe. AgreementMaker: efficient matching for large real-world schemas and ontologies. *Proceedings of the VLDB Endowment*, 2(2):1586–1589, 2009.

[45] Krzysztof Czarnecki and Simon Helsen. Classification of model transformation approaches. In *Proceedings of the 2nd OOPSLA Workshop on Generative Techniques in the Context of the Model Driven Architecture*, volume 45, pages 1–17. USA, 2003.

[46] Krzysztof Czarnecki, Ulrich Eisenecker, Robert Glück, David Vandevoorde, and Todd Veldhuizen. Generative Programming and Active Libraries. In *Generic Programming*, pages 25–39. Springer, Berlin, Heidelberg, 2000. DOI: 10.1007/3-540-39953-4_3.

[47] Anish Das Sarma, Aditya Parameswaran, Hector Garcia-Molina, and Jennifer Widom. Synthesizing view definitions from data. In *Proceedings of the 13th International Conference on Database Theory*, pages 89–103. ACM, 2010.

[48] Igor Dejanovic, Maja Tumbas, Gordana Milosavljevic, and Branko Perisic. Comparison of Textual and Visual Notations of DOMMLite Domain-Specific Language. In *ADBIS (Local Proceedings)*, pages 131–136, 2010.

[49] Johan den Haan. 8 Reasons Why Model-Driven Approaches (will) Fail, July 2008. URL https://www.infoq.com/articles/8-reasons-why-MDE-fails.

[50] Johan den Haan. 8 Reasons Why Model-Driven Development is Dangerous, June 2009. URL http://www.theenterprisearchitect.eu/blog/2009/06/25/8-reasons-why-model-driven-development-is-dangerous/.

[51] Vladimir Dimitrieski, Milan Čeliković, Vladimir Ivančević, and Ivan Luković. A Comparison of Ecore and GOPPRR through an Information System Meta Modeling Approach. In *Joint Proceedings of co-located Events at the 8th European Conference on Modelling Foundations and Applications*, pages 217–228, Technical University of Denmark, Kongens Lyngby, Denmark, 2012. Technical University of Denmark. ISBN 978-87-643-1014-6.

[52] Vladimir Dimitrieski, Ivan Luković, Slavica Aleksić, Milan Čeliković, and Gordana Milosavljevic. An Overview of Selected Visual M2m Transformation Languages. pages 450–455, Kopaonik, Serbia, March 2014. Society for Information Systems and Computer Networks. ISBN 978-86-85525-14-8.

[53] Vladimir Dimitrieski, Milan Čeliković, Slavica Aleksić, Sonja Ristić, Abdalla Alargt, and Ivan Luković. Concepts and evaluation of the extended entity-relationship approach to database design in a multi-paradigm information system modeling tool. *Computer Languages, Systems & Structures*, 44, Part C:299–318, 2015. ISSN 1477-8424. doi: 10.1016/j.cl.2015.08.011.

[54] Vladimir Dimitrieski, Milan Čeliković, Nemanja Igić, Heiko Kern, and Fred Stefan. Reuse of Rules in a Mapping-Based Integration Tool. In *Intelligent Software Methodologies, Tools and Techniques*, volume 532 of *Communications in Computer and Information Science*, pages 269–280, Naples, Italy, September 2015. Springer. ISBN 978-3-319-22688-0. doi: 10.1007/978-3-319-22689-7.

[55] Vladimir Dimitrieski, Gajo Petrović, Aleksandar Kovačević, Ivan Luković, and Hamido Fujita. A Survey on Ontologies and Ontology Alignment Approaches in Healthcare. In *Trends in Applied Knowledge-Based Systems and Data Science*, Lecture Notes in Artificial

Intelligence, pages 373–385, Morioka, Japan, August 2016. Springer. ISBN 1611-3349. doi: 10.1007/978-3-319-42007-3.

[56] Hong-Hai Do and Erhard Rahm. COMA: a system for flexible combination of schema matching approaches. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 610–621. VLDB Endowment, 2002.

[57] Hong-Hai Do and Erhard Rahm. Matching large schemas: Approaches and evaluation. *Information Systems*, 32(6):857–885, 2007.

[58] Hong-Hai Do, Sergey Melnik, and Erhard Rahm. Comparison of schema matching evaluations. In *Web, Web-Services, and Database Systems*, pages 221–237. Springer, 2002.

[59] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM, 2001.

[60] AnHai Doan, Jayant Madhavan, Pedro Domingos, and Alon Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the 11th international conference on World Wide Web*, pages 662–673. ACM, 2002.

[61] Rainer Drath and Alexander Horch. Industrie 4.0: Hit or Hype? *IEEE Industrial Electronics Magazine*, 8(2):56–58, June 2014. ISSN 1932-4529. doi: 10.1109/MIE.2014.2312079.

[62] Rainer Drath, Arndt Lüder, Jörn Peschke, and Lorenz Hundt. AutomationML-the glue for seamless automation engineering. In *Emerging Technologies and Factory Automation, 2008. ETFA 2008. IEEE International Conference on*, pages 616–623. IEEE, 2008.

[63] Fabien Duchateau and Zohra Bellahsene. YAM: A Step Forward for Generating a Dedicated Schema Matcher. In Abdelkader Hameurlain, Josef Küng, and Roland Wagner, editors, *Transactions on Large-Scale Data- and Knowledge-Centered Systems XXV*, number 9620 in Lecture Notes in Computer Science, pages 150–185. Springer Berlin Heidelberg, 2016. ISBN 978-3-662-49533-9 978-3-662-49534-6. DOI: 10.1007/978-3-662-49534-6_5.

[64] Fabien Duchateau, Zohra Bellahsene, and Remi Coletta. A flexible approach for planning schema matching algorithms. In *On the Move to Meaningful Internet Systems: OTM 2008*, pages 249–264. Springer, 2008.

[65] Anne Dujin, Cornelia Geissler, and Dirk Horstkötter. Industry 4.0: The New Industrial Revolution - How Europe Will Succeed. Technical report, Roland Berger Strategy Consultants Gmbh, Munich, 2014. URL https://www.rolandberger.com/media/pdf/Roland_Berger_TAB_Industry_4_0_20140403.pdf.

[66] Eclipse Vorto. Eclipse Vorto - IoT Toolset for standardized device descriptions, 2016. URL https://www.eclipse.org/vorto/index.html.

[67] Marc Ehrig. *Ontology Alignment: Bridging the Semantic Gap*, volume 4 of *Semantic Web And Beyond Computing for Human Experience*. Springer, 2007.

[68] Marc Ehrig and Steffen Staab. QOM–quick ontology mapping. In *The Semantic Web–ISWC 2004*, pages 683–697. Springer, 2004.

[69] Marc Ehrig and York Sure. Ontology mapping–an integrated approach. In *The Semantic Web: Research and Applications*, pages 76–91. Springer, 2004.

[70] Milan Čelikovic, Ivan Luković, Slavica Aleksić, and Vladimir Ivančević. A MOF based meta-model and a concrete DSL syntax of IIS*Case PIM concepts. *Computer Science and Information Systems*, 9(3):1075–1103, 2012. ISSN 1820-0214. doi: 10.2298/CSIS120203034C.

[71] David W. Embley, David Jackman, and Li Xu. Multifaceted Exploitation of Metadata for Attribute Match Discovery in Information Integration. In *Workshop on information integration on the Web*, pages 110–117. Citeseer, 2001.

[72] Jérôme Euzenat, Petko Valtchev, and others. Similarity-based ontology alignment in OWL-lite. In *ECAI*, volume 16, page 333, 2004.

[73] Jérôme Euzenat, Pavel Shvaiko, and others. *Ontology matching*, volume 18. Springer, 2007.

[74] AutomationML e.V. AutomationML Whitepaper. Whitepaper, AutomationML e.V., October 2014. URL https://www.automationml.org/o.red/uploads/dateien/1417686950-AutomationML%20Whitepaper%20Part%201%20-%20AutomationML%20Architecture%20v2_Oct2014.pdf.

[75] Peter C. Evans and Marco Annunziata. Industrial Internet: Pushing the Boundaries of Minds and Machine. Technical report, General Electrics, November 2012.

[76] FME. Fme desktop integration tool, March 2016. URL https://www.safe.com/fme/fme-desktop/l.

[77] OPC Foundation. OPC UA - Overview and Concepts v1.03. Industry Standard Specification, OPC Foundation, Scottsdale, USA, October 2015. URL https://opcfoundation.org/developer-tools/specifications-unified-architecture/part-1-overview-and-concepts/.

[78] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[79] Fausto Giunchiglia, Pavel Shvaiko, and Mikalai Yatskevich. S-Match: an algorithm and an implementation of semantic matching. In *ESWS*, volume 3053, pages 61–75. Springer, 2004.

[80] Beth Gold-Bernstein and William Ruh. *Enterprise Integration: The Essential Guide to Integration Solutions*. Addison-Wesley Professional, Boston, July 2004. ISBN 978-0-321-22390-6.

[81] Georg Gottlob and Pierre Senellart. Schema mapping discovery from data instances. *Journal of the ACM (JACM)*, 57(2):6, 2010.

[82] Mikell P. Groover. *Automation, Production Systems, and Computer-Integrated Manufacturing*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2007. ISBN 9780132393218.

[83] Yosemite Project Group. Yosemite Project, October 2017. URL http://yosemiteproject.org/.

[84] Shubham Gupta, Pedro Szekely, Craig A. Knoblock, Aman Goel, Mohsen Taheriyan, and Maria Muslea. Karma: A system for mapping structured sources into the Semantic Web. In *The Semantic Web: ESWC 2012 Satellite Events*, pages 430–434. Springer, 2012.

[85] Iiro Harjunkoski, Rasmus Nyström, and Alexander Horch. Integration of scheduling and control—Theory or practice? *Computers & Chemical Engineering*, 33(12):1909–1918, December 2009. ISSN 00981354. doi: 10.1016/j.compchemeng.2009.06.016.

[86] Scott A. Helmers. *Microsoft Visio 2013 Step By Step*. Microsoft Press, Sebastopol, CA, 1 edition edition, May 2013. ISBN 978-0-7356-6946-8.

[87] Mario Hermann, Tobias Pentek, and Boris Otto. Design principles for Industrie 4.0 scenarios: a literature review. Working Paper No. 01-2015, Technische Universität Dortmund, Dortmund, 2015.

[88] Robert La Verne Hobbs. Mapping for mapping source and target objects, March 2009. URL http://www.google.com/patents/US7499943.

[89] Wei Hu, Yuzhong Qu, and Gong Cheng. Matching large ontologies: A divide-and-conquer approach. *Data & Knowledge Engineering*, 67(1):140–160, 2008.

[90] Jun Huh, John Grundy, John Hosking, Karen Liu, and Robert Amor. Integrated data mapping for a software meta-tool. In *Software Engineering Conference, 2009. ASWEC'09. Australian*, pages 111–120. IEEE, 2009.

[91] Informatica. Data Integration Tools and Software Solutions, 2016. URL https://www.informatica.com/products/data-integration.html#fbid=RNZoAPRdLqq.

[92] inray Industriensoftware GmbH. OPC Router, 2016. URL https://www.inray.de/en/products/opcrouter.html.

[93] Javier Luis Cánovas Izquierdo and Jordi Cabot. Discovering implicit schemas in JSON data. In *International Conference on Web Engineering*, pages 68–83. Springer, 2013.

[94] Yves R. Jean-Mary, E. Patrick Shironoshita, and Mansur R. Kabuka. Ontology matching with semantic verification. *Web Semantics: Science, Services and Agents on the World Wide Web*, 7(3):235–251, September 2009. ISSN 1570-8268. doi: 10.1016/j.websem.2009.04.001.

[95] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, and Ivan Kurtev. ATL: A model transformation tool. *Science of Computer Programming*, 72(1-2):31–39, June 2008. ISSN 01676423. doi: 10.1016/j.scico.2007.08.002.

[96] Jumpeye Components. JSChart, 2016. URL http://www.jscharts.com/.

[97] Matjaz B. Juric, editor. *SOA approach to integration: XML, Web services, ESB, and BPEL in real-world SOA projects*. From technologies to solutions. Packt Publ, Birmingham, 2007. ISBN 978-1-904811-17-6. OCLC: 254083785.

[98] Henning Kagermann, Wolf-Dieter Lukas, and Wolfgang Wahlster. Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution. *VDI nachrichten*, 13:2, 2011.

[99] Henning Kagermann, Johannes Helbig, Ariane Hellinger, and Wolfgang Wahlster. Recommendations for Implementing the Strategic Initiative INDUSTRIE 4.0. Technical report, Forschungsunion, 2013.

[100] Gökhan Kahraman and Semih Bilgen. A framework for qualitative assessment of domain-specific languages. *Software & Systems Modeling*, pages 1–22, 2013.

[101] Serope Kalpakjian and Stephen Schmid. *Manufacturing, Engineering and Technology*. Digital Designs, 7 edition, 2006.

[102] Gabor Karsai, Janos Sztipanovits, Akos Ledeczi, and Ted Bapty. Model-integrated development of embedded software. *Proceedings of the IEEE*, 91(1):145–164, 2003.

[103] Steven Kelly and Juha-Pekka Tolvanen. *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008. ISBN 978-0-470-03666-2.

[104] H. Kern. Study of Interoperability between meta-modeling tools. In *2014 Federated Conference on Computer Science and Information Systems (FedCSIS)*, pages 1629–1637, September 2014. doi: 10.15439/2014F255.

[105] Heiko Kern. The interchange of (meta) models between metaedit+ and eclipse emf using m3-level-based bridges. In *8th OOPSLA Workshop on Domain-Specific Modeling at OOPSLA*, volume 2008, 2008.

[106] Heiko Kern. *Model Interoperability between Meta-Modeling Environments by using M3-Level-Based Bridges*. PhD thesis, University of Leipzig, Leipzig, August 2016.

[107] Heiko Kern, Axel Hummel, and Stefan Kühne. Towards a comparative analysis of meta-metamodels. In *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, &#38; VMIL'11*, SPLASH '11 Workshops, pages 7–12, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-1183-0. doi: 10.1145/2095050.2095053.

[108] Heiko Kern, Fred Stefan, Vladimir Dimitrieski, and Milan Čeliković. Mapping-Based Exchange of Models Between Meta-Modeling Tools. In *Proceedings of the 14th Workshop on Domain-Specific Modeling*, DSM '14, pages 29–34, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2156-3. doi: 10.1145/2688447.2688453. URL http://doi.acm.org/10.1145/2688447.2688453.

[109] Heiko Kern, Fred Stefan, and Vladimir Dimitrieski. Intelligent And Self-Adapting Integration Between Machines And Information Systems. *Iadis International Journal on Computer Science and Information Systems*, 10(1):47–63, 2015. ISSN 1646-3692. URL http://www.iadisportal.org/ijcsis/papers/2015180104.pdf.

[110] Heiko Kern, Fred Stefan, Klaus-Peter Fähnrich, and Vladimir Dimitrieski. A Mapping-Based Framework for the Integration of Machine Data and Information Systems. In *Proceedings of 8th IADIS International Conference on Information Systems 2015*, pages 113–120, Madeira, Portugal, March 2015. International Association for Development of the Information Society. ISBN 978-989-8533-33-3.

[111] Thomas Kühne. Matters of (meta-) modeling. *Software and Systems Modeling*, 5(4):369–385, 2006.

[112] Anneke G. Kleppe. *MDA explained: the model driven architecture: practice and promise*. The Addison-Wesley object technology series. Addison-Wesley, Boston, 2003. ISBN 0-321-19442-X.

[113] Dimitrios S. Kolovos, Richard F. Paige, and Fiona AC Polack. The Epsilon Transformation Language. In *Theory and Practice of Model Transformations*, pages 46–60. Springer, 2008.

[114] Tomaž Kosar, Marjan Mernik, and Jeffrey C. Carver. Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. *Empirical Software Engineering*, 17(3):276–304, August 2011. ISSN 1382-3256, 1573-7616. doi: 10.1007/s10664-011-9172-x.

[115] Vitomir Kovanovic and Dragan Djuric. Highway: a domain specific language for enterprise application integration. In *Proceedings of the 5th India Software Engineering Conference*, pages 33–36. ACM, 2012.

[116] Ivan Kurtev, Jean Bézivin, and Mehmet Akşit. Technological spaces: An initial appraisal. pages 1–6, Irvine, USA, November 2002.

[117] Ralf Kutsche and Nikola Milanović. (Meta-)Models, Tools and Infrastructures for Business Application Integration. In Roland Kaschek, Christian Kop, Claudia Steinberger, and Günther Fliedl, editors, *Information Systems and e-Business Technologies*, number 5 in Lecture Notes in Business Information Processing, pages 579–584. Springer Berlin Heidelberg, April 2008. ISBN 978-3-540-78941-3 978-3-540-78942-0.

[118] Ralf Kutsche, Nikola Milanović, Gregor Bauhoff, Timo Baum, Mario Cartsburg, Daniel Kumpe, and Jürgen Widiker. BIZYCLE: Model-based Interoperability Platform for Software and Data Integration. In *Proceedings of the MDTPI at ECMDA*, volume 430, 2008.

[119] Eric Lander and John P. Holdren. Accelerating U.S. Advanced Manufacturing. Report to the president, President's Council of Advisors on Science and Technology, Washington, USA, October 2014.

[120] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, August 2014. ISSN 1867-0202. doi: 10.1007/s12599-014-0334-4.

[121] Yoonkyong Lee, Mayssam Sayyadian, AnHai Doan, and Arnon S. Rosenthal. eTuner: tuning schema matching software using synthetic scenarios. *The VLDB Journal—The International Journal on Very Large Data Bases*, 16(1):97–122, 2007.

[122] Barbara Staudt Lerner. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems (TODS)*, 25(1):89–127, 2000.

[123] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[124] J. Li, J. Tang, Y. Li, and Q. Luo. RiMOM: A Dynamic Multistrategy Ontology Alignment Framework. *IEEE Transactions on Knowledge and Data Engineering*, 21(8):1218–1232, August 2009. ISSN 1041-4347. doi: 10.1109/TKDE.2008.202.

[125] David S. Linthicum. *Enterprise application integration*. Addison-Wesley Professional, 2000.

[126] Liquid Technologies. Liquid XML Studio, 2016. URL https://www.liquid-technologies.com/xml-studio.

[127] Dominik Lucke, Carmen Constantinescu, and Engelbert Westkämper. Smart factory-a step towards the next generation of manufacturing. In *Manufacturing systems and technologies for the new frontier*, pages 115–118. Springer, 2008.

[128] Ivan Luković. *Integration of Information System Database Module Schemas*. PhD thesis, University of Novi Sad, Novi Sad, 1995.

[129] Ivan Luković, Sonja Ristić, and Pavle Mogin. A methodology of a database schema design using the subschemas. In *Proceedings of IEEE International Conference on Computational Cybernetics*, Budapest, Hungary, 2003.

[130] Ivan Luković, Sonja Ristić, Pavle Mogin, and Jelena Pavićević. Database Schema Integration Process–A Methodology and Aspects of Its Applying. *Novi Sad Journal of Mathematics*, 36(1):115–140, 2006.

[131] Ivan Luković, Pavle Mogin, Jelena Pavićević, and Sonja Ristić. An approach to developing complex database schemas using form types. *Software: Practice and Experience*, 37 (15):1621–1656, December 2007. ISSN 00380644, 1097024X. doi: 10.1002/spe.820.

[132] Ivan Luković, Slavica Aleksić, Vladimir Ivančević, and Milan Čeliković. DSLs in Action with Model Based Approaches to Information System Development. In *Formal and Practical Aspects of Domain-Specific Languages*. IGI Global, September 2012. ISBN 9781466620926, 9781466620933.

[133] Ivan Luković, Sonja Ristić, Aleksandar Popović, Jelena Pavićević, Slavica Kordić, Nikola Obrenović, Pavle Mogin, Jovo Mostić, and Miro Govedarica. IIS*Case, 2012.

[134] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *VLDB*, volume 1, pages 49–58, 2001.

[135] Wolfgang Mahnke, Stefan-Helmut Leitner, and Matthias Damm. *OPC unified architecture*. Springer Science & Business Media, 2009.

[136] Dimitris Manakanatas and Dimitris Plexousakis. A Tool for Semi-Automated Semantic Schema Mapping: Design and Implementation. In *DISWEB*, Luxembourg, 2006.

[137] Carolyn Mathas. Industry 4.0 is closer than you think, December 2013. URL http://www.edn.com/design/wireless-networking/4425363/Industry-4-0-is-closer-than-you-think.

[138] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings. 18th International Conference on Data Engineering*, pages 117–128. IEEE, 2002.

[139] Tom Mens and Pieter Van Gorp. A Taxonomy of Model Transformation. *Electronic Notes in Theoretical Computer Science*, 152:125–142, March 2006. ISSN 15710661. doi: 10.1016/j.entcs.2005.10.021.

[140] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4):316–344, 2005.

[141] MetaCase. GOPPRR: MetaEdit+ Workbench User's Guide, Version 4.5, February 2017. URL https://www.metacase.com/support/45/manuals/mwb/Mw-1_1.html.

[142] Microsoft. MicrosoftSQL Server Integration Services (SSIS), 2016. URL https://msdn.microsoft.com/en-us/library/ms169917.aspx.

[143] Nikola Milanović, Ralf Kutsche, Timo Baum, Mario Cartsburg, Hatice Elmasgünes, Marco Pohl, and Jürgen Widiker. Model&Metamodel, Metadata and Document Repository for Software and Data Integration. In Krzysztof Czarnecki, Ileana Ober, Jean-Michel Bruel, Axel Uhl, and Markus Völter, editors, *Model Driven Engineering Languages and Systems*, number 5301 in Lecture Notes in Computer Science, pages 416–430. Springer Berlin Heidelberg, September 2008. ISBN 978-3-540-87874-2 978-3-540-87875-9.

[144] Nikola Milanović, Mario Cartsburg, Ralf Kutsche, Jürgen Widiker, and Frank Kschonsak. Model-Based Interoperability of Heterogeneous Information Systems: An Industrial Case Study. In Richard F. Paige, Alan Hartman, and Arend Rensink, editors, *Model Driven Architecture - Foundations and Applications*, number 5562 in Lecture Notes in Computer Science, pages 325–336. Springer Berlin Heidelberg, June 2009. ISBN 978-3-642-02673-7 978-3-642-02674-4.

[145] Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. The Clio project: managing heterogeneity. *SIgMOD Record*, 30(1):78–83, 2001.

[146] Tova Milo and Sagit Zohar. Using schema matching to simplify heterogeneous data translation. In *VLDB*, volume 98, pages 24–27. Citeseer, 1998.

[147] Prasenjit Mitra, Gio Wiederhold, and Jan Jannink. Semi-automatic integration of knowledge sources. 1999.

[148] MuleSoft. Anypoint Studio, 2016. URL `https://www.mulesoft.com/platform/studio`.

[149] Judith M. Myerson. *Enterprise Systems Integration, Second Edition*. Auerbach Publications, Boca Raton, 2 edition edition, September 2001. ISBN 978-0-8493-1149-9.

[150] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9), 2014.

[151] Natalya F. Noy and Mark A. Musen. Anchor-PROMPT: Using non-local context for semantic matching. In *Proceedings of the workshop on ontologies and information sharing at the international joint conference on artificial intelligence (IJCAI)*, pages 63–70, 2001.

[152] Nikola Obrenović. *An Approach to Design, Consolidation and Transformations of Database Schema Check Constraints Based on Platform Independent Models*. PhD thesis, University of Novi Sad, Novi Sad, 2015.

[153] Oracle. Oracle Data Integrator, June 2016. URL `http://www.oracle.com/technetwork/middleware/data-integrator/overview/index.html`.

[154] Luigi Palopoli, Giorgio Terracina, and Domenico Ursino. The System DIKE: Towards the Semi-Automatic Synthesis of Cooperative Information Systems and Data Warehouses. In *ADBIS-DASFAA Symposium*, pages 108–117, 2000.

[155] Panos Vassiliadis, Anastasios Karagiannis, Vasiliki Tziovara, Alkis Simitsis, and Ioannina Hellas. Towards a Benchmark for ETL Workflows. In *Proceedings of the 5th International Workshop on Quality in Databases (QDB 2007)*, Vienna, Austria, September 2007.

[156] David Pavlis. CloverETL Rapid Data Integration, 2016. URL `http://www.cloveretl.com/`.

[157] A. Popović, I. Luković, and S. Ristić. A Specification of the Structures of Business Applications in the IIS* Case Tool. *Info M–Journal of Information Technology and Multimedia Systems*, 25:17–24, 2008. ISSN 1451-4397.

[158] Aleksandar Popović. *An Approach to Specification of Application System Executable Models*. PhD thesis, University of Montenegro, Podgorica, 2013.

[159] Aleksandar Popović, Ivan Luković, Vladimir Dimitrieski, and Verislav Djukic. A DSL for modeling application-specific functionalities of business applications. *Computer Languages, Systems & Structures*, 43:69–95, 2015. ISSN 1477-8424. doi: 10.1016/j.cl.2015.03.003.

[160] Stefan Poslad. *Ubiquitous Computing: Smart Devices, Environments and Interactions*. Wiley, Chichester, U.K, 1 edition edition, April 2009. ISBN 978-0-470-03560-3.

[161] Eric Pulier and Hugh Taylor. *Understanding enterprise SOA*. Manning Greenwich, Conn, 2006.

[162] Ananth Raghavan, Divya Rangarajan, Rao Shen, Marcos André Gonçalves, Naga Srinivas Vemuri, Weiguo Fan, and Edward A. Fox. Schema mapper: a visualization tool for DL integration. In *Digital Libraries, 2005. JCDL'05. Proceedings of the 5th ACM/IEEE-CS Joint Conference on*, pages 414–414. IEEE, 2005.

[163] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[164] Kavitha Rathinasamy. *Comparison of Schema and Data Integration tools for the Asset Management Domain*. PhD thesis, MS Thesis, University of South Australia, 2011.

[165] Paul Resnick and Hal R. Varian. Recommender systems. *Communications of the ACM*, 40 (3):56–58, 1997. URL http://dl.acm.org/citation.cfm?id=245121.

[166] Sonja Ristić. *Research on the Problem of Database Subschema Consolidation*. PhD thesis, University of Novi Sad, Novi Sad, 2002.

[167] Sonja Ristić, Pavle Mogin, and Ivan Luković. Specifying database updates using a subschema. In *Proceedings of the 7th IEEE International Conference on Intelligent Engineering Systems*, pages 203–212, Assiut–Luxor, Egypt, 2003.

[168] Sonja Ristić, Ivan Luković, Jelena Pavičević, and Pavle Mogin. Resolving Database Constraint Collisions Using IIS*Case Tool. *Journal of information and organizational sciences*, 31 (1):187–206, 2007.

[169] George G. Robertson, John E. Churchill, Mary P. Czerwinski, Prasad Sripathi Panditharadhya, and Uday Bhaskara. Schema mapper, October 2012. URL http://www.google.com/patents/US8280923.

[170] Jonathan Robie, Michael Dyck, and Josh Spiegel. XML Path Language (XPath), 2015. URL https://www.w3.org/TR/xpath/.

[171] Richard Rozsa. MetaDapper, 2016. URL http://www.metadapper.com/.

[172] James Rumbaugh, Ivar Jacobson, and Grady Booch. *The Unified Modeling Language Reference Manual*. Pearson Higher Education, 2004.

[173] Len Seligman, Peter Mork, Alon Halevy, Ken Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. OpenII: an open source information integration toolkit. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 1057–1060. ACM, 2010.

[174] Leonard J. Seligman, Peter D. S. Mork, Joel G. Korb, Kenneth B. Samuel, and Christopher S. Wolf. Tools and methods for semi-automatic schema matching, January 2008. URL http://www.google.com/patents/US20080021912.

[175] Pavel Shvaiko and Jérôme Euzenat. A survey of schema-based matching approaches. In *Journal on data semantics IV*, pages 146–171. Springer, 2005.

[176] Hassan A. Sleiman, Abdul W. Sultán, Rafael Z. Frantz, Rafael Corchuelo, and others. Towards Automatic Code Generation for EAI Solutions using DSL Tools. In *JISBD*, pages 134–145, 2009.

[177] Herbert Stachowiak. Allgemeine modelltheorie. 1973.

[178] Thomas Stahl, Markus Voelter, and Krzysztof Czarnecki. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, Chichester, England ; Hoboken, NJ, 1 edition edition, May 2006. ISBN 978-0-470-02570-3.

[179] Dave Steinberg, Frank Budinsky, Ed Merks, and Marcelo Paternostro. *EMF: eclipse modeling framework*. Addison-Wesley, Boston, USA, 2 edition, 2008. ISBN 0-321-33188-5.

[180] Stylus Studio XML. XML Data Integration, XML Tools, Web Services and XQuery, 2016.

[181] Balder Ten Cate, Phokion G. Kolaitis, and Wang-Chiew Tan. Schema mappings and data examples. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 777–780. ACM, 2013.

[182] Susan Marie Thomas. Schema mapping and data transformation on the basis of layout and content, July 2012. URL http://www.google.com/patents/US8234312.

[183] Susan Marie Thomas. Schema mapping and data transformation on the basis of a conceptual model, December 2014. URL http://www.google.com/patents/US8924415.

[184] Dragan Đurić, Dragan Gašević, and Vladan Devedžić. The Tao of Modeling Spaces. *Journal of Object Technology*, 5(8):125–147, 2006.

[185] Wil MP Van der Aalst. Formalization and verification of event-driven process chains. *Information and Software technology*, 41(10):639–650, 1999.

[186] Arie Van Deursen, Paul Klint, and Joost Visser. Domain-Specific Languages: An Annotated Bibliography. *Sigplan Notices*, 35(6):26–36, 2000.

[187] Ruben Verborgh and Max De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.

[188] Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-driven software development: technology, engineering, management*. John Wiley & Sons, 2013.

[189] Markus Voelter, Sebastian Benz, Christian Dietrich, Birgit Engelmann, Mats Helander, Lennart CL Kats, Eelco Visser, and Guido Wachsmuth. *DSL engineering: Designing, implementing and using domain-specific languages*. dslbook. org, 2013.

[190] Željko Vuković, Nikola Milanović, and Gregor Bauhoff. Prototype of a Framework for Ontology-aided semantic conflict resolution in enterprise integration. In *Proceedints of 5th International Conference on Information Society and Technology*, Kopaonik, Serbia, March 2015. Society for Information Systems and Computer Networks.

[191] Željko Vuković, Nikola Milanović, Renata Vaderna, Igor Dejanović, and Gordana Milosavljević. SAIL: A Domain-Specific Language for Semantic-Aided Automation of Interface Mapping in Enterprise Integration. In *On the Move to Meaningful Internet Systems: OTM 2015 Workshops*, pages 97–106. Springer, 2015.

[192] Wolfgang Wahlster. Industry 4.0: From Smart Factories to Smart Products, May 2012.

[193] Jean-Baptiste Waldner. *CIM: principles of computer-integrated manufacturing*. John Wiley & Sons, 1992.

[194] Evan Walsh, Alexander O'Connor, and Vincent Wade. The FUSE domain-aware approach to user model interoperability: A comparative study. In *Information Reuse and Integration (IRI), 2013 IEEE 14th International Conference on*, pages 554–561. IEEE, 2013.

[195] Jason Tsong-Li Wang, Kaizhong Zhang, Karpjoo Jeong, and Dennis Shasha. A system for approximate tree matching. *Knowledge and Data Engineering, IEEE Transactions on*, 6 (4):559–571, 1994.

[196] Mark Weiser. The computer for the 21st century. *Scientific american*, 265(3):94–104, 1991.

[197] Manuel Wimmer. *From mining to mapping and roundtrip transformations–a systematic approach to model-based tool integration*. PhD thesis, Vienna University of Technology, 2008.

[198] William E. Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. 1990.

[199] Niklaus Wirth. *Algorithms + Data Structures = Programs*. Prentice Hall, Englewood Cliffs, N.J., 1st edition edition, February 1976. ISBN 978-0-13-022418-7.

[200] Martin Wischenbart, Stefan Mitsch, Elisabeth Kapsammer, Angelika Kusel, Stephan Lechner, Birgit Pröll, Werner Retschitzegger, Johannes Schönböck, Wieland Schwinger, and Manuel Wimmer. Automatic data transformation: breaching the walled gardens of social network platforms. In *Proceedings of the Ninth Asia-Pacific Conference on Conceptual Modelling-Volume 143*, pages 89–98. Australian Computer Society, Inc., 2013.

[201] Robert Worden. Improving Data Quality with Open Mapping Tools. White paper, Open Mapping Software Ltd, February 2011.

[202] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *ACM SIGMOD Record*, volume 30, pages 485–496. ACM, 2001.

[203] Detlef Zuehlke. SmartFactory—Towards a factory-of-things. *Annual Reviews in Control*, 34(1):129–138, 2010.

[204] ZVEI. The Reference Architectural Model For industrie 4.0 (RAMI 4.0), 2015.