УНИВЕРЗИТЕТ У НОВОМ САДУ

ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА

ДЕПАРТМАН ЗА РАЧУНАРСТВО И АУТОМАТИКУ

Дејан Средојевић

Агентски, домен-оријентисани језик за развој интелигентних агената за дистрибуирано не-аксиоматско резоновање

– ДОКТОРСКА ДИСЕРТАЦИЈА –

Ментор:

Проф. др Милан Видаковић

Нови Сад, 2019.

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**:		Дејан Средојевић
Ментор, **МН**:		др Милан Видаковић, редовни професор
Наслов рада, **НР**:		Агентски, домен-оријентисани језик за развој интелигентних агената за дистрибуирано не-аксиоматско резоновање
Језик публикације, **ЈП**:		Српски
Језик извода, **ЈИ**:		Српски
Земља публиковања, **ЗП**:		Србија
Уже географско подручје, **УГП**:		АП Војводина
Година, **ГО**:		2019.
Издавач, **ИЗ**:		Ауторски репринт
Место и адреса, **МА**:		Нови Сад, Факултет техничких наука, Трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/ страна/ цитата/табела/слика/графика/прилога)		10/121/123/3/20/0/0
Научна област, **НО**:		Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:		Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:		ALAS, агентски језици, домен-оријентисани језик, Siebog, NAL, DNARS, мобилност агената
УДК		
Чува се, **ЧУ**:		Библиотека Факултета техничких наука у Новом Саду
Важна напомена, **ВН**:		
Извод, **ИЗ**:		У дисертацији је представљен прототип агентског, домен-оријентисаног језика ALAS. Основни мотиви развоја ALAS језика су подршка дистрибуираном не-аксиоматском резоновању као и омогућавање интероперабилности и хетерогене мобилности Siebog агената јер је приликом анализе постојећих агентских домен-оријентисаних језика утврђено да ни један језик не подржава ове захтеве. Побољшање у односу на сличне постојеће агентске, домен-оријентисане језике огледа се и у програмским конструктима које нуди ALAS језик а чија је основна сврха писање концизних агената који се извршавају у специфичним доменима.
Датум прихватања теме, **ДП**:		25.10.2018. године
Датум одбране, **ДО**:		
Чланови комисије, **КО**:	Председник:	др Игор Дејановић, ванредни професор
	Члан:	др Бојана Димић Сурла, ванредни професор
	Члан:	др Мирјана Ивановић, редовни професор — Потпис ментора
	Члан:	др Синиша Николић, доцент
	Члан, ментор:	др Милан Видаковић, 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**:		Monographic publication	
Type of record, **TR**:		Textual printed material	
Contents code, **CC**:		PhD thesis	
Author, **AU**:		Dejan Sredojević	
Mentor, **MN**:		Milan Vidaković, Ph.D., Full Professor	
Title, **TI**:		Agent-oriented domain-specific language for the development of intelligent distributed non-axiomatic reasoning agents	
Language of text, **LT**:		Serbian	
Language of abstract, **LA**:		Serbian	
Country of publication, **CP**:		Serbia	
Locality of publication, **LP**:		AP Vojvodina	
Publication year, **PY**:		2019.	
Publisher, **PB**:		Author's reprint	
Publication place, **PP**:		Novi Sad, Faculty of Technical Sciences, Trg Dositeja Obradovića 6	
Physical description, **PD**: (chapters/ pages/ref./tables/pictures/graphs/appendixes)		10/121/123/3/20/0/0	
Scientific field, **SF**:		Electrical and computer engineering	
Scientific discipline, **SD**:		Applied computer science and informatics	
Subject/Key words, **S/KW**:		ALAS, agent-oriented languages, domain-specific languages, Siebog, NAL, DNARS, agent mobility	
UC			
Holding data, **HD**:		Library of Faculty of Technical Sciences, Novi Sad	
Note, **N**:			
Abstract, **AB**:		The dissertation presents the prototype of an agent-oriented, domain-specific language ALAS. The basic motives for the development of the ALAS language are support for distributed non-axiomatic reasoning, as well as enabling the interoperability and heterogeneous mobility of agents, because it is concluded by analysing existing agent-oriented, domain-specific languages, that there is no language that supports these requirements. The improvement compared to similar existing agent-oriented, domain-specific languages are also reflected in program constructs offered by ALAS language, whose the main purpose is to enable writing the concise agents that are executed in specific domains.	
Accepted by the Scientific Board on, **ASB**:		October 25th, 2018	
Defended on, **DE**:			
Defended Board, **DB**:	President:	Igor Dejanović, Ph. D., Associate Professor	
	Member:	Bojana Dimić Surla, Ph. D., Associate Professor	
	Member:	Mirjana Ivanović, Ph.D., Full Professor	Menthor's sign
	Member:	Siniša Nikolić Ph.D., Assistant Professor	
	Member, Mentor:	Milan Vidaković, Ph.D., Full Professor	

За израду докторске дисертације дугујем велику захвалност свом ментору, проф. др Милану Видаковићу на великом стрпљењу, корисним саветима, подршци и експертизи, без којих ово истраживање не би било могуће.

Захваљујем се и проф. др Мирјани Ивановић за све сугестије и савете које ми је пружила при изради дисертације.

Највећу захвалност на бескрајном разумевању
дугујем својој породици, родитељима Рајку и Невенци,
сестри Данијели, супрузи Снежани и мојој
највећој мотивацији – синовима Вуку и Ђорђу којима и
посвећујем ову докторску дисертацију.

Дејан Средојевић

Резиме

Докторска дисертација се бави развојем агентског, домен-оријентисаног језика ALAS. Основни мотив за развој ALAS језика је био да се подржи дистрибуирано не-аксиоматско резоновање и обезбеди развој специфичних врста интелигентних агената. ALAS је дизајниран да подржи Siebog мултиагентски систем и имплементацију Siebog интелигентних агената. Siebog је дистрибуирани мултиагентски систем заснован на модерним веб и пословним стандардима. Siebog нуди подршку за закључивање засновану на дистрибуираном систему за не-аксиоматско резоновање (енг. Distributed non-axiomatic reasoning system - DNARS). DNARS је систем резоновања заснован на не-аксиоматској логици (енг. non-axiomatic logic - NAL). Пре развоја ALAS-а, DNARS агенти су могли да се пишу само у Java програмском језику.

Други мотив за развој ALAS језика је био решавање проблема интероперабилности и хетерогене мобилности агената у оквиру Siebog платформе. Циљ таквог језика је да буде универзалан и омогући извршавање интелигентних агената у различитим, хетерогеним окружењима без обзира у ком језику су она имплементирана. За потребе истраживања ALAS је имплементиран тако да се његови агенти могу извршавати у окружењима имплементираним у Java и JavaScript програмским језицима. Као доказ о успешној покривености проблема, у дисертацији су описани поступци конверзије ALAS кода у Java и JavaScript језике а такође су дати примери агената написаних у ALAS језику који се успешно могу конвертовати и у Java и JavaScript језик. Још један од циљева језика је омогућавање програмерима да лакше развијају интелигентне агенте користећи конструкте специфичне за одговарајуће домене примене.

Кључне речи: ALAS, агентски језици, домен-оријентисани језици, Siebog, NAL, DNARS, мобилност агената

Abstract

Doctoral thesis deals with the development of an agent-oriented, domain-specific ALAS language. The main motive for the development of the ALAS language was to support distributed non-axiomatic reasoning and to provide the development of specific types of intelligent agents. ALAS is designed to support the Siebog multiagent system and implementation of the Siebog intelligent agents. Siebog is a distributed multiagent system based on the modern web and enterprise standards. Siebog offers support to reasoning based on the Distributed Non-Axiomatic Reasoning System (DNARS). DNARS is a reasoning system based on the Non-Axiomatic Logic (NAL). So far, DNARS-enabled agents could be written only in Java programming language.

The second motive for the development of the ALAS language was to solve the problem of interoperability and heterogeneous mobility of agents within the Siebog platform. The goal of such a language is to be universal and to be enabled the execution of intelligent agents in different, heterogeneous environments, regardless of the implementation language of these environments. For the research purposes, ALAS is implemented so that ALAS agents can be executed in environments implemented in Java or JavaScript programming languages. As a proof of successful problem coverage, the thesis describes the conversion process of the ALAS code into Java and JavaScript languages as well as examples of agents written in ALAS language that can be successfully converted to both Java and JavaScript. One more of the ALAS goals is to provide that developers can develop intelligent agents more easily by using specific domain constructs.

Keywords: ALAS, agent-oriented languages, domain-specific languages, Siebog, NAL, DNARS, agent mobility

Садржај

Предговор

Предмет истраживања докторске дисертације је универзални агентски и домен-оријентисани језик (енг. domain-specific language - DSL) намењен развоју интелигентних агената који се могу извршавати на различитим платформама. Идеја је да агенти имају одговарајући степен интелигенције као и да буду уско везани за домене у којима се извршавају. У дисертацији је описан дизајн ALAS агентског, домен-оријентисаног језика.

Како би успешно обављали своје задатке, агентима су неопходна одговарајућа окружења. Ова окружења, позната под називом мултиагентски системи (енг. multiagent systems - MASs) или мултиагентске платформе, укључују ефикасну инфраструктуру за агентску комуникацију, обезбеђују подршку за мобилност агената, омогућују агентима да обављају своје задатке, да претражују могућности других агената, итд. Једна од модерних мултиагентских платформи је Siebog која је развијена применом најновијих технологија и стандарда (Mitrović 2015).

Тренутно постоји велики број ефикасних агентских и домен-оријентисаних језика али ни један постојећи језик није довољно ефикасан како би подржао све захтеве Siebog мултиагентске платформе. Скалабилност (енг. scalability), отпорност на грешке (енг. fault tolerance), дељење кода (енг. code sharing), интероперабилност (енг. interoperability), вишеплатформска размена порука (енг. cross-platform messaging), хетерогена мобилност агената (енг. heterogeneous agent mobility), расподела оптерећења (енг. load balancing), агенти засновани на не-аксиоматској логици, али и друге предности Siebog-а у односу на остале мултиагентске системе, захтевале су развој новог језика који ће подржати све особине Siebog-а и омогућити развој продуктивних агената. У овој дисертацији је описан поступак развоја ALAS-а – агентског, домен-оријентисаног језика који ће подржати али и у великој мери унапредити Siebog што је и основни циљ настанка ALAS језика. Основне особине интелигентног, мултиагентског окружења Siebog описане су у поглављу 4.

Основни недостаци Siebog-а као што су делимична интероперабилност и ограничена хетерогена мобилност агената решени су применом ALAS језика. Под појмом хетерогене мобилности агената подразумева се мобилност агената у хетерогеним окружењима. Хетерогена окружења су рачунарски чворови (енг. computer nodes) који представљају инстанце различитих мултиагентских система који могу бити имплементирани у различитим програмским језицима и који се могу заснивати на различитим виртуелним машинама. Циљ је да агенти буду потпуно интероперабилни и да могу да се крећу и извршавају у хетерогеним окружењима.

ALAS спада у категорију агентских програмских језика који представљају кључне алате програмирања агената (енг. agent-oriented programming - AOP). AOP је парадигма развоја софтвера која има за циљ ефикасан развој софтверских агената и мултиагентских система. Његови главни циљеви су да идентификују, анализирају и понуде решења за најважнија теоријска и практична питања везана за дизајн и конструкцију софтверских агената.

С обзиром да је предмет истраживања дисертације развој агентског, домен-оријентисаног језика, у поглављу 2 описано је тренутно стање у области агентских и домен-оријентисаних језика и дат је преглед више таквих програмских језика који су на разне начине имали утицај на развој ALAS-а.

Поред побољшања везаних за интероперабилност и хетерогену мобилност агената у оквиру Siebog-а, главна мотивација да се развије нови језик је била потреба за развојем агената са когнитивним способностима (Serenko i Detlor 2004). Како би Siebog агенти имали елементе вештачке интелигенције (енг. artificial intelligence), уведен је систем за не-аксиоматско резоновање као један од продуктивнијих система вештачке интелигенције. Основна предност и разлог примене не-аксиоматске логике - NAL за развој интелигентних ALAS агената је могућност доношења закључака на основу неконзистентног или често недовољног, тј. несигурног знања. Применом NAL, агенти постају интелигентни и способни да доносе нове закључке. NAL се састоји од девет различитих слојева при чему сваки виши слој поседује особине свих нижих слојева укључујући неке нове, напредније особине (Wang 2006). ALAS језик подржава прва 4 нивоа NAL који су довољни за практичну примену, при чему ће наредни кораци у развоју ALAS-а подразумевати потпуну подршку NAL у виду имплементације преосталих нивоа. Основне могућности које нуди NAL су описане у поглављу 3.

Све већа заступљеност семантичког веба (енг. semantic web) и потреба за извођењем нових структурираних знања имала је утицај на развој дистрибуираног система за не-аксиоматско резоновање, чије су основне особине продуктивност и брзо извођење нових закључака (у реалном времену) на основу огромних база знања (Mitrović 2015). Увођењем подршке DNARS-у који је уједно заснован на концептима NAL-а, у ALAS агенте су инкорпорирани елементи вештачке интелигенције које нуди DNARS. DNARS на коме су засновани ALAS агенти описан је у поглављу 5.

Преглед одговарајућих технологија које су коришћене за развој ALAS-а дат је у поглављу 6.

Дизајн и имплементација ALAS језика, као и начин на који је реализована подршка Siebog-у и DNARS-у описани су у поглављу 7. У овом поглављу је описана целокупна граматика језика која омогућује подршку Siebog-у као и део граматике која се односи на подршку интелигентној страни Siebog-а тј. дистрибуираном не-аксиоматском резоновању. Да би се у потпуности решио проблем хетерогене агентске мобилности потребно је приликом пребацивања агента на други чвор у кластеру рачунара (енг. computer cluster) конвертовати код агента из ALAS језика у одговарајући имплементациони језик одредишне

платформе. Овај поступак, као и примери конверзије ALAS кода у Java и JavaScript изворне кодове описани такође у овом поглављу.

У поглављу 8 представљена је студија случаја са примерима примене ALAS језика за писање интелигентних агената који користе реално знање на основу којег су способни да дају одговоре на питања из реалног света. У овом поглављу приказан је пример ALAS агента који је способан да донесе одговарајући закључак на основу недовољног знања.

На крају, закључак садржи преглед досадашњих резултата, као и наредне кораке који подразумевају додатно побољшање ALAS језика како би у потпуности подржао све захтеве Siebog-а и DNARS-а у виду ефикасније хетерогене мобилности и додатних елемената NAL.

Нови Сад, 2019. *Дејан Средојевић*

1 Увод

Један од највећих мотива многих научника у протеклих 60 година био је формализација људског размишљања и закључивања. Као резултат тестирања функција људског мозга, средином 20. века појавио се појам вештачке интелигенције. У основи вештачке интелигенције је сложена математичка логика чији је циљ да представи људски начин размишљања, расуђивања и закључивања (Ranković 2018). Експлозија јавно доступних информација „о свему“ и огромних база знања као што су Wikipedia[1], DBpedia[2] итд., али и велики напредак у разним доменима вештачке интелигенције који се десио почетком 21. века (расуђивање, машинско учење, експертни системи, неуронске мреже, медицинска дијагноза, обрада језика, слика итд.) омогућили су развој рачунара који су у стању да подрже такве захтеве (Müller 2013).

1.1 Агентска технологија

Истраживања у области вештачке интелигенције у претходних 25 година изнедрила су разна перспективна решења при чему је једна од најперспективнијих области агентска технологија која подразумева примену интелигентних агената у разним доменима (Serenko i Detlor 2004). Агентска технологија представља један од најконзистентнијих приступа развоју дистрибуираних система (Vidaković i dr. 2013). Агентски системи имплементирани у пољу техничког инжењерства у последње две деценије усвојени су као нови и ефикасни концепти за контролу система (Metzger i Polakow 2011). Интелигентни софтверски агенти имају такве особине и могућности да се сматрају новим помаком у информатичкој науци. Користе се

[1] https://www.wikipedia.org
[2] http://wiki.dbpedia.org

широм света, како у академским заједницама тако и у пословним круговима код доношења битних пословних одлука. Агенти могу понудити различите погодности крајњим корисницима аутоматизацијом разних процеса који се понављају или решавањем њихове комплексности (Serenko i Detlor 2004).

Још у почетној фази развоја, Wooldridge и Jennings (1995) су међу првима предвидели веома брз развој интелигентних агената. У (Jennings 2000) агенти су представљени као модели следеће генерације за инжењеринг комплексних, дистрибуираних система. Агенти се такође користе као свеобухватни оквир за обједињавање под-дисциплина вештачке интелигенције које су неопходне за дизајнирање и развој интелигентних ентитета.

Главна предност интелигентних (когнитивних) агената над једноставнијим агентима је знатно дубље разумевање окружења у којем делују (граде семантичку репрезентацију света у којем делују) и различитих процеса који се одвијају у том окружењу. Још једна важна предност је да когнитивни агенти имају способности резоновања на вишем нивоу. Сходно томе, ови агенти су у стању да много ефикасније делују како би подржали различите процесе који се одвијају у одговарајућим срединама. На пример, градећи софистицирани модел корисника (на основу богатог скупа карактеристика наведених од стране корисника, или изведених особина на основу праћења активности корисника), когнитивни агенти могу понудити висок ниво интеракције и персонализације а самим тим и већу вероватноћу проналаска исправног резултата за корисника. Заправо, они могу да играју улогу повећане људске способности уласком у симбиозу са корисником, уместо да само испуњавају једноставнију улогу аутоматизације неких задатака (Nabeth i dr. 2005).

Многи истраживачи су се бавили интелигентним агентима али установљено је да се због разних параметара као што су домен функционисања, опсег одговорности, врсте задатака које треба да извршавају итд., не може формулисати универзална дефиниција интелигентног агента. Међутим, на основу заједничких особина које агенти поседују, може се формулисати генерална дефиниција агента на следећи начин (Green i dr. 1996, 5):

Агент је рачунарски ентитет који:

- делује у име других ентитета на аутономан начин
- извршава своје активности са одређеним нивоом проактивности и / или реактивности
- показује одређени ниво кључних атрибута учења, сарадње и мобилности

У практичној примени агент се може дефинисати као самостални (аутономни) програмски објекат који поседује сопствена средства и механизме помоћу којих, на основу доступних информација, доноси закључке, решава одговарајуће задатке и проблеме, било на сопствену иницијативу, иницијативу корисника одговарајућих система, других агената итд. (Wooldridge 1997). У наставку су набројане неке основне и често заједничке особине агената које уједно представљају разлику између агената и програма:

- Адаптивност (енг. adaptability): способност учења и унапређивања на темељу искуства
- Аутономија (самосталност) (енг. autonomy): оријентисаност ка циљу, проактивно и самоиницијативно понашање
- Кооперација (енг. cooperation): способност "сарадње" са другим агентима како би остварили заједничке циљеве
- Способност закључивања (резоновања) (енг. reasoning): способност реаговања на апстрактне захтеве
- Способност комуникације (енг. communication) са другим агентима на начин који је сличнији људској комуникацији него стандардном симболичком програм-програм протоколу
- Мобилност (енг. mobility): способност преласка с једне на другу рачунарску платформу
- Реактивност (енг. reactivity): способност селективног опажања и деловања.

На основу претходних особина, типова задатака које решавају, или домена примене, агенти се могу класификовати на следећи начин (Nwana 1996; Qusay 2000):

- Колаборативни (кооперативни) агенти (енг. Collaborative/Cooperative agents) - овај тип софтверског агента одликује се аутономијом и сарадњом са другим агентима у остваривању делегираних задатака од стране својих власника. Опште карактеристике ових агената укључују аутономију, друштвене способности, способност реаговања и проактивност.
- Итерфејс агенти (енг. Interface agents) - овај тип софтверског агента карактерише његова аутономија, способност учења и рад са корисницима у истом радном окружењу. Основни задатак ових агената је учење тј. праћење навика корисника на основу чега у будућности могу сами да предвиде понашање корисника.
- Мобилни агенти (енг. Mobile agents) – као што се и из самог назива може закључити, основна одлика ових агената је мобилност, тј. способност премештања (пребацивања) између међусобно повезаних окружења како би остварили задате циљеве.
- Информациони (Интернет) агенти (енг. Information/Internet agents) су настали због велике потражње за алатима који ће обрађивати огромне количине информација које су се појавиле и које се све више и све брже појављују. Информациони агенти имају улогу управљања, манипулације и разврставања информација које долазе из многих дистрибуираних извора.
- Реактивни агенти (енг. Reactive agents) се могу представити као скупови модула који функционишу самостално да би извршили одређени задатак. Ове агенте карактерише једноставност и основна интеракција са другим

агентима. Такође се карактерише динамичком интеракцијом са својим окружењем, што може довести до нежељене комплексности.

– Хибридни (интелигентни) агенти (енг. Hybrid agents) настају комбиновањем два или више претходно наведених типова агената у јединствени ентитет.

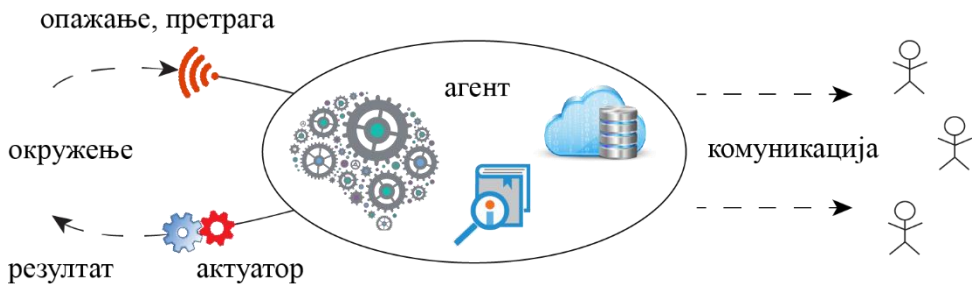
На слици 1.1 је приказан концептуални модел интелигентног агената који поседује следеће особине тј. компоненте:

– Опажање (енг. percepts): претрага, прикупљање информација (извори информација могу бити разни типови сензора, камера, базе података итд.)

– Окружење (енг. environment): домен функционисања

– Комуникација (енг. communication): механизми који омогућују међуагентску комуникацију

– Актуатор (енг. actuator): механизам који обрађује све доступне информације за агента (укључујући и информације које поседују други агенти из истог окружења применом механизма који омогућује међуагентску комуникацију) како би решио одговарајући задатак



Слика 1.1. Концептуални модел интелигентног агента

Постоје разни механизми који се користе за развој интелигентних агената (Rao i Georgeff, 1995). Један од најчешће примењиваних приступа је BDI (енг. belief–desire–intention) архитектура која подразумева моделовање агената који поседују следеће особине:

– *Веровања* (енг. beliefs) – искази (који се односе на домен у ком се агент развија) који могу (али не морају) бити тачни

– *Жеље* (енг. desires) – стање које агент жели да постигне

– *Намере* (енг. intentions) – жеље које агент покушава да оствари

У претходним годинама постојали су бројни формализми који су се користили за развој BDI агената (Guerra-Hernandez, Castro-Manzano i Fallah-Seghrouchni 2009; Hoek i Wooldridge 2013; Rao 1996; Wooldridge 2000). Иако се за развој BDI агената често користе комплексни математички алгоритми, ови формализми генерално не разматрају практичну примену агената тј. не узимају у обзир разне практичне проблеме, па се може десити да агент нпр. нема довољно ресурса, у смислу времена и простора, да реши неки проблем.

Међутим, уместо на BDI архитектури, агенти које подржава ALAS језик описан у овој дисертацији, засновани су на DNARS механизаму за развој интелигентних агената (Sredojević, Vidaković i Ivanović 2018). Главна предност предложене DNARS архитектуре у односу на BDI је у слојевитој и дистрибуираној организацији позадинске базе знања. DNARS је дизајниран са скалабилношћу и толеранцијом грешака, омогућавајући агентима да доносе закључке над веома великим базама знања.

1.1.1 Мултиагентски системи

Технологија интелигентног агента представља нови модел у решавању многих уобичајених проблема који се могу решити структурираним и објектно оријентисаним парадигмама (Mzahm, Sharifuddin i Tang 2014). Међутим, за велике, сложене дистрибуиране системе, постојеће парадигме програмирања постају гломазне, компликоване и споре. Како би решили такве сложене проблеме, истраживачи у области вештачке интелигенције који се баве интелигентним агентима почели су да користе групе агената који заједнички (паралелно) раде, имитирајући понашање људских друштава, како би што брже и ефикасније решили проблем. Основна идеја је да се развију агенти са адекватним знањима и групишу у одговарајуће окружење како би се створио систем у којем је сваком агенту додељен одговарајући задатак. Софтверски системи који садрже скупове логичких ентитета у виду дистрибуираних, аутономних и кооперативних агената који међусобно комуницирају како би решили одговарајуће проблеме називају се мултиагентски системи (Garrido i dr. 2017; Lujak 2017; Shoham i Leyton-Brown 2009; Weiss 2000, 2013; Wooldridge 2009). Мултиагентски систем је софтверски систем са инфраструктурном подршком за своје агенте (Badica i dr. 2011). Његове основне функционалности укључују управљање животним циклусом агента, размену порука, сервисни подсистем који ефикасно подржава агенте, даје им могућност приступа ресурсима, извршавање сложених алгоритама итд. (Vidaković i dr. 2013).

Мултиагентски системи су углавном аутономни и обично раде без надзора својих власника. Сваки агент у систему ради на решавању својих подзадатака, при чему су осталим агентима у систему најчешће додељени различити задаци. Систем функционише тако што разлаже сложени задатак у подзадатке, а затим додељује сваки подзадатак одговарајућем агенту који применом својих могућности треба да реши дати задатак. Ако агент нема довољно знања или ресурса да изврши подзадатак, може "консултовати" друге агенте за решење или им делегирати посао. Агенти у овим системима могу да раде заједно као један ентитет или појединачно, у неколико одвојених ентитета (Laarabi i dr. 2013). Такође, мултиагентски системи у многим ситуацијама нуде безбедност (нпр. агентски код и интегритет података као и енкрипцију порука), систем конекције који омогућава интеракцију агената у физички раздвојеним окружењима,

подршку за мобилност и перзистентност агената. Агенти, програмирање агената и мултиагентски системи уводе нове и неконвенцијалне идеје и концепте. Према (Badica i dr. 2011; Luck, Ashri, i Inverno 2004; Wiebe i Wooldridge. 2013) постоји преко 100 агентских платформи и алата који су развијени или се још увек развијају.

Међутим, функционисање аутономних и колаборативних агената постаје сложено и комплексно кад су у питању различити захтеви различитих агентских окружења (Challenger i dr. 2016). Главни проблем настаје услед платформске зависности што онемогућује несметану међуагентску комуникацију, расподелу послова и ефикасно извршавање задатака. Како би се овај проблем што ефикасније решио, за програмирање агената користе се агентски и домен-оријентисани језици.

1.2 Агентски језици

Агентска технологија је нова парадигма за софтверске системе. Да би се у потпуности искористила способност ове технологије, више агената ради у софтверском окружењу тако што сарађују, координирају или преговарају међусобно. Међутим, ове интеракције захтевају да агенти међусобно комуницирају путем заједничког језика или протокола. Заједно са развојем мултиагентских система, међуагентска комуникација представља подручје интензивних истраживања (Soon, On, Anthony, Hamdan 2018). Улога комуникације у мултиагентском систему је да обезбеди средства за размену информација на основу прихватљивог скупа правила или протокола слања и примања порука. По пријему поруке, агент мора бити у стању да дешифрује значење поруке и да на одговарајући начин реагује како би се постигао заједнички циљ. Такође, агент мора бити способан да комуницира са другим агентима како би могао да извршава задатке за њих.

За ширу примену агентских технологија кључно је постојање агентских програмских језика (енг. Agent oriented programming languages – AOPLs). Под агентским језиком подразумевамо систем који дозвољава програмирање хардверских или софтверских рачунарских система заснованих на теоријским концептима агената. Подразумева се да такви језици укључују структуре које одговарају агентима. Развијени су различити агентски језици и различити семантички модели да би се олакшала комуникација између агената у мултиагентским системима.

Увођење појма *agent-oriented programming languages* приписује се (Shoham 1990) који је представио агентско програмирање као нову парадигму, специјализацију објектно-оријентисаног програмирања која промовише друштвени приступ раду, где више агената ступа у међусобну интеракцију у циљу решавања проблема (Shoham 1990, 1993). Shoham (1993) предлаже да потпуно развијен агентски програмски систем има три компоненте:

– ограничени формални језик са јасном синтаксом и семантиком за описивање менталног стања; ментално стање ће се јединствено дефинисати кроз неколико модалитета, као што су веровање и посвећеност
– интерпретирани програмски језик у којем се дефинишу и програмирају агенти, с примитивним наредбама као што су REQUEST и INFORM; семантика програмског језика мора да одговара семантици менталног стања
– процес *агентификације*, за компајлирање агентских програма у извршне системе ниског нивоа.

Поред агентских језика који се користе за имплементацију агената постоје и агентски комуникациони језици (енг. Agent communication languages - ACLs) (Soon, On, Anthony, Hamdan 2018). ACL представља важну компоненту у мултиагентском систему који омогућује агентима да комуницирају и размењују поруке и знање. Два ACL-а која су међу првима нашли ширу примену у области агентских технологија су KQML и FIPA ACL (Finin i dr. 1994; German i Sheremetov 2007).

1.3 Домен-оријентисани језици

Основни циљ у развоју новог програмског језика је да програмирање учини ефикаснијим. Савршен програмски језик треба да обезбеди прави ниво апстракције, што значи да описује решења природно и сакрије непотребне детаље. Такође, требало би да буде довољно изражајан у домену проблема и да обезбеди гаранције на својства која су критична у том домену. Такође би требало да има прецизну семантику да би се омогућило формално резоновање о програму. Са језицима опште намене (енг. General-Purpose Languages - GPLs), то је тешко постићи, пошто GPL имају тенденцију да буду општи, што резултира лошом подршком за домен оријентисану нотацију. Постизање претходно наведених својстава омогућено је развојем домен-оријентисаних језика – DSL. За специфичне домене примене, DSL-ови су изражајнији и лакши за употребу од GPL-ова, са повећањем продуктивности али и трошкова одржавања (Kosar i dr. 2010). DSL-ови постају све важнији у софтверском инжењерству. Алати за развој ових језика постају бољи и једноставнији тако да се DSL-ови могу развити са релативно мало труда.

DSL-ови су прилагођени да буду веома продуктивни у одређеном домену. Они пружају прикладне апстракције, које се често користе и од стране не-програмера, усклађене са доменом проблема како би се постигла решења на једноставан и кратак начин. Један од најчешће навођених примера DSL-а је SQL. SQL омогућава корисницима да рукују подацима, табелама и базама података без знања програмирања. SQL обезбеђује виши ниво апстракције са специфичним концептима као што су *select* или *update* у домену података који се налазе у редовима и колонама.

Међутим, за развој сложенијих система, један DSL није довољан за постизање продуктивности и побољшање квалитета, а чак ни више DSL-ова који нису добро интегрисани. Само холистички приступ (у којем се неколико DSL-ова беспрекорно интегрисаних може користити за моделовање решења за различите домене проблема), може донети значајна побољшања у развоју.

Поред назива домен-оријентисани језици, у пракси се за ове језике користе и неки други називи као што су: домен-специфични језици, језици специфични за домен, наменски језици, мали језици (енг. little languages) итд.

Постоје разне дефиниције DSL-ова а неколико најчешће коришћених су описане у наставку.

DSL се може представити као језик који је прилагођен и ограничен на специфични домен примене. DSL-ови нуде одговарајуће конструкте и нотације прилагођене домену и доменским експертима (Mernik, Heering i Sloane 2005).

У (Kelly i Tolvanen, 2008) DSL-ови су представљени као програмски језици који подижу ниво апстракције изнад програмирања специфицирањем решења које директно користи концепте и правила из специфичног домена проблема.

Voelter (2013) описује DSL као концизан, обрадив језик за описивање одређеног проблема приликом развоја система у одређеном домену. Апстракције и коришћене нотације су прилагођене интересним групама које специфицирају одређене проблеме.

Доста истраживача се бави DSL-овима међу којима је један од најистакнутијих и најцитиранијих Martin Fowler. У својој књизи (Fowler 2010), он дефинише DSL-ове на следећи начин:

DSL је компјутерски програмски језик ограничене експресивности фокусиран на одређени домен. Постоје четири кључна елемента ове дефиниције.

- Компјутерски програмски језик: DSL се користи од стране људи како би дали инструкције рачунару да уради и како да уради нешто. Као и са било којим модерним програмским језиком, његова структура је дизајнирана тако да олакшава људима да је разумеју, али ипак треба да буде таква да се може извршити помоћу рачунара.
- Природа језика: DSL је програмски језик и као такав треба да има осећај лакоће изражавања при чему експресивност не долази само од индивидуалних израза, већ и од начина на који се могу креирати скупови израза.
- Ограничена експресивност: GPL пружа много могућности: подржава различите податке, контролу и апстрактне структуре. Све ово је корисно, али чини га тежим за учење и употребу. DSL-ови пружају минималан број карактеристика потребних да би се подржао њихов домен. Међутим, цео софтверски систем се не може изградити само применом DSL-а. Пожељно је користити DSL само за одређени аспект система.
- Фокус на домен: Ограничени језик је користан само ако има јасан фокус на мале домене. Фокус на домен је оно што ограничени језик чини вредним.

Даље, Martin Fowler дели DSL-ове на три основне групе: интерни, екстерни и језичке радионице.

Интерни DSL је језик који настаје проширивањем неког постојећег GPL-а, најчешће у виду програмских конструката како би се подржали специфични домени проблема. Један језик опште намене може имати неограничен број интерних DSL-ова који се могу посматрати као додатне библиотеке тог језика. Одређена скрипта написана у интерном DSL-у је важећи код у свом језику опште намене, али користи само подскуп његових карактеристика на одређени начин да би обрадио један мали аспект целокупног система. Међутим, интерни DSL-ови имају велика ограничења. Синтакса интерног DSL-а може бити само оно што допушта синтакса *host* GPL-а. Без тога, код интерног DSL-а се не може анализирати и компајлирати од стране компајлера на *host* језику. Класичан пример интерног DSL-а је Lisp. Ruby је такође развио јаку DSL културу - многе Ruby библиотеке су заправо изграђене на концептима DSL-а.

Екстерни DSL се пише на другачијем језику од главног (*host*) језика у којем се порграмира нека апликација (систем) и трансформише се у тај језик користећи одговарајући компајлер или преводилац. Њихова главна особина је да су изграђени од нуле, а њихова синтакса се пажљиво дизајнира за одговарајући домен. Кључна особина екстерног DSL-а је у томе што не морају да зависе ни од једног другог језика што значи да постоји слобода у коришћењу било каквих облика при њиховом дизајнирању. Постоји доста могућности да се изрази одређени домен у најједноставнијем могућем облику за читање и модификовање. Формат је ограничен само способношћу дизајнера језика да направи преводиоца који може да анализира конфигурациони фајл и произведе неки извршни фајл - обично у основном језику у којем се развија одговарајући систем. Међутим, из овога произилази очигледан недостатак – потребно је изградити преводиоца. За једноставне језике ово није проблем али кад су у питању комплекснији језици то може бити сложен процес који захтева доста труда. Али, развојем нових алата као што је и сам textX који је описан у овој дисертацији, овај процес је у великој мери олакшан. Примери екстерних DSL-ова укључују регуларне изразе, SQL, Awk и XML конфигурационе датотеке за системе као што су Struts и Hibernate.

Језичка радионица је специјализовано окружење (енг. Integrated Development Environment - IDE) за дефинисање и изградњу DSL-ова. Језичка радионица се користи не само да би се одредила структура DSL-а, већ и као прилагођено окружење за уређивање и писање DSL скрипти. Неке од најчешће коришћених језичких радионица су Xtext[3], MPS[4] и Spoofax[5].

[3] https://www.eclipse.org/Xtext
[4] https://www.jetbrains.com/mps
[5] http://strategoxt.org/Spoofax

1.4 Образложење теме докторске дисертације

Развој софтвера применом језика опште намене често је неефикасан када су у питању специфични домени примене. Применом језика опште намене понекад је немогуће имплементирати све захтеве везане за специфичне домене. У тим случајевима знатно су ефикаснији домен-оријентисани језици који представљају ефикасне и концизне језике за описивање специјалних захтева приликом развоја система из специфичних домена.

Све већа заступљеност вештачке интелигенције у информационим системима доводи до наглог повећања примене агентских технологија у развоју великог броја софтверских решења. За што ефикаснију примену агената за решавање сложених проблема користе се мултиагентски системи при чему се за имплементацију агената користе агентски језици.

Докторска дисертација се бави развојем универзалног агентског, домен-оријентисаног језика који је намењен развоју интелигентних агената у оквиру новог мултиагентског система Siebog (Mitrović i dr. 2016a; Mitrović i dr. 2016b;). За разлику од већине мултиагентских система заснованих на класичној BDI (енг. Beliefs-Desires-Intentions) архитектури за резоновање, Siebog користи врло ефикасан дистрибуирани систем за не-аксиоматско резоновање (енг. Distributed Non-Axiomatic Reasoning System (DNARS)) (Mitrović 2015). С обзиром да се ни један постојећи језик не може користити за развој интелигентих DNARS агената, тема докторске дисертације представља развој ALAS језика који ће подржати дистрибуирано не-аксиоматско резоновање и омогућити развој мобилних агената што ће решити проблем платформске независности који је постојао у оквиру Siebog-a.

1.5 Преглед хипотеза истраживања

Хипотеза 1:

Несметане миграције агената између клијент-сервер окружења имплементираних у различитим програмским језицима могу се обезбедити повећањем интероперабилности и обезбеђивањем хетерогене мобилности агената. Ово се може постићи развојем универзалног агентског, домен-оријентисаног језика и механизама који ће омогућити аутоматску конверзију агентског кода у имплементациони језик одредишне платформе.

Хипотеза 2:

Мултиагентско окружење Siebog је пројектовано тако да подржи развој интелигентних агената способних да доносе закључке и самостално решавају одговарајуће задатке. Такви агенти се могу развијати применом језика који ће

моћи да подржи писање агената заснованих на дистрибуираном не-аксиоматском резоновању.

Хипотеза 3:

Интелигентни агенати у случају комплексних области функционисања се могу развијати и без помоћи експерата из одговарајућих домена примене. Ово се може постићи применом конструката специфичних за одговарајуће домене у којима се извршавају интелигентни агенти.

1.6 Предмет, проблеми и циљеви истраживања

Предмет истраживања своди се на развој универзалног агентског и домен-оријентисаног језика ALAS. Циљ је да се реши проблем мобилности Siebog агената, развију специјализоване функције тј. конструкти за одговарајуће, специфичне домене примене и да се омогући развој интелигентих агената заснованих на дистрибуираном не-аксиоматском резоновању.

Идеја је да овај језик омогући програмирање софтверских агената који ће имати одговарајући степен интелигенције како би био подржан дистрибуирани систем за не-аксиоматско резоновање заснован на принципима не-аксиоматске логике. План је да се применом ALAS језика могу написати агенти који ће моћи самостално да доносе одговарајуће закључке на основу релевантних база знања.

Приликом развоја мултиагентске платформе Siebog, остао је нерешен проблем интероперабилности и хетерогене мобилности агената. Циљ је да агенти буду интероперабилни и да могу да се крећу између различитих чворова умрежених рачунара, где чворови могу бити независни и имплементирани у различитим програмским језицима. Агенти морају бити способни да се приликом пребацивања путем мреже на платформе имплементиране у различитим програмским језицима аутоматски конвертују из ALAS језика у имплементациони језик одредишне платформе и на тај начин лако наставе да се извршавају на тој, одредишној платформи. Интероперабилност је била делимично решена редизајнирањем серверске стране Siebog-а у сервисно оријентисану инфраструктуру при чему су основни модули система били редефинисани као веб сервиси. Међутим, нови проблем је настао када су агенти требали да се крећу и извршавају на различитим платформама које нису имплементиране у истом програмском језику. Нови, универзални језик за писање софтверских агената који ће бити детаљно описан у дисертацији, у потпуности решава овај проблем. Увођењем подршке Siebog-у у виду DNARS-а који је заснован на концептима не-аксиоматске логике, у агенте су инкорпорирани елементи вештачке интелигенције које нуди DNARS па је предмет истраживања такође развој језика који ће поред претходних особина моћи да омогући развој интелигентних агената заснованих на дистрибуираном не-аксиоматском резоновању.

Како би се повећала продуктивност агената у специфичним доменима, дисертација се бави дизајнирањем домен-оријентисаног језика који ће моћи да подржи одговарајуће перформансе које су уско везане за домене у којима се агенти извршавају а план је да се ово реализује применом одговарајућих програмских конструката. Применом специјалних функција тј. конструката које ће нови језик понудити, програмерима ће у великој мери бити олакшано писање агената, при чему је сва функционална сложеност домена сакривена у одговарајућим конструктима што омогућује програмерима да се фокусирају на решавање логичких проблема. Примена таквих конструката омогућује да програмери који развијају агенте не морају бити експерти у тим специфичним доменима.

Као коначан резултат рада на овој дисертацији, очекује се нови, агентски, домен-оријентисани језик ALAS, који подржава све претходно наведене захтеве и који је интегрисан у мултиагентско окружење Siebog.

1.7 Могућност примене очекиваних резултата

Агентски, домен-оријентисани језик за развој интелигентних агената за дистрибуирано не-аксиоматско резоновање може наћи практичну примену у разним мултиагентским системима у којима постоји проблем интероперабилности и мобилности софтверских агената као и мултиагентским системима у којима постоји потреба за агентима са когнитивним способностима који могу самостално да изводе нове закључке на основу доступних информација и постојећих база знања.

1.8 Структура рада

Докторска дисертација опиује процес развоја агентског, домен-оријентисаног језика ALAS а у наставку је дата структура рада.

У првом поглављу описани су основни концепти који се односе на агентску технологију, интелигентне агенте, агентске и домен-оријентисане језике.

У другом полављу – *Стање у научној области*, дат је приказ постојећих решења из области агентских и домен-оријентисаних језика. У овом поглављу су набројани и описани примери језика који су у највећој мери утицали на развој ALAS-a.

Треће поглавље се односи на NAL систем за резоновање. Поред решавања проблема мобилности један од основних задатака ALAS-a је да подржи развој интелигентних агената. У овој дисертацији је описан поступак инкорпорације механизма који омогућује резоновање и извођење закључака у ALAS језику. Овај механизам се заснива на не-аксиоматској логици која је описана у овом поглављу.

Четврто поглавље описује мултиагентско окружење Siebog које је послужило као основа за рзвој ALAS-а. Сама идеја о настанку ALAS-а се појавила управо због проблема који је постојао у оквиру Siebog-а а који се односио на мобилност агената. Развојем агентског језика ALAS решен је тај проблем, а самим тим је Siebog постао један од најефикаснијих и најфункционалнијих мултиагентских система.

У петом поглављу је описан дистрибуирани систем за не-аксиоматско резоновање који се заснива на не-аксиоматској логици и који представља саставни део Siebog окружења. Овај систем омогућује развој интелигентних агената у оквиру Siebog-а. ALAS језик је развијен тако да подржи овај систем.

У шестом поглављу су представљене и укратко описане технологије које су коришћене у процесу развоја ALAS језика (Xtext, textX, Jinja2, Eclipse, Titan граф база података).

У седмом поглављу је описан процес дизајна и имплементације прототипа ALAS језика. У овом поглављу су описани граматика, синтакса и визуализација модела и мета-модела језика. Такође, описан је и начин на који је решен проблем мобилности Siebog агената а који је био основни мотив за настанак ALAS језика. Примери конверзије ALAS кода у Java и JavaScript програмске језике су такође дати у оквиру овог поглавља.

У осмом поглављу је представљена студија случаја са примерима агената који су способни да изводе закључке на основу база знања из реалног света.

Девето поглавље садржи закључак у којем се показује на који начин су решени проблеми и постављени циљеви докторске дисертације. У закључку су наведени и правци будућег развоја ALAS језика.

2 Стање у научној области

Интензивнији развој интелигентних агената почео је крајем 20. века што се поклапа са помацима у развоју система применом вештачке интелигенције. Истовремено са развојем интелигентних агената почињу да се развијају мултиагентски системи како би омогућили још ефикаснију употребу агената. Сврха сваког мултиагентског система је да побољша могућности агената тако што ће омогућити њихову међусобну комуникацију како би агенти могли заједно да извршавају сложене задатке. Највећи значај за постизање ефикасне међуагентске комуникације приписује се агентским програмским језицима.

У дисертацији је описан нов мултиагентски систем Siebog који је развијен применом најновијих техника које се користе у развоју мултиагентских система. Како би се решили недостаци Siebog-а – интероперабилност и хетерогена мобилност агената било је потребно развити одговарајући агентски језик који ће подржати ове захтеве. Да би се развио што ефикаснији језик један од начина је да се искористе предности постојећих језика из одговарајуће области примене. У наставку су представљени агентски језици који су на одређене начине утицали на развој новог језика ALAS чији је основни задатак да покрије недостатке Siebog-а.

Као што је раније наведено, за увођење агентских програмских језика у област агентских технологија заслужан је Shoham (1990, 1993).

Након појаве агентског програмирања, први језик који је служио за имплементацију агената био је AGENT0 (Shoham 1993). Агенти су на основу порука и одговарајућих предуслова креирали разне акције. Основни термини који су служили за међуагентску комуникацију су REQUEST, UNREQUEST и INFORM. AGENT0 се временом развијао стицањем нових особина па су тако настале нове верзије овог језика: PLACA (Thomas 1995) који је увео подршку за агентско планирање и Agent-K (Davies i Edwards 1994), који је заменио обичне облике комуникације са стандардизованим језиком и протоколом за комуникацију између агената - KQML, побољшавајући укупну интероперабилност. Ови језици нису нашли ширу практичну примену али су у одређеној мери имали утицај на развој каснијих AOPL-а (Mitrović i dr. 2012).

Прве кораке у области међуагентских комуникација остварила је Фондација за интелигентне физичке агенте – FIPA[6]. Од 2005. године, FIPA је стандард за агенте и мултиагентске системе у оквиру IEEE организације. Siebog агенти (Mitrović 2015; Mitrović i dr. 2016a; Mitrović i dr. 2016b;), како на серверској, тако и на клијентској страни, комуницирају разменом порука заснованој на FIPA ACL стандарду за међуагентску размену порука (FIPA 2018, Mitrović 2015). ALAS је дизајниран тако да подржи FIPA ACL стандард за међуагентску комуникацију.

Међу првим AOPL настао је и AgentSpeak (Rao 1996). Први и најважнији циљ овог језика је био програмирање BDI агената. Користи се као апстрактни оквир за BDI агенте (Georgeff i dr. 1999). Првобитно, AgentSpeak се користио за разумевање везе између практичних имплементација BDI архитектуре као што је процедурални систем за резоновање (енг. Procedural Reasoning System – PRS) (Georgeff i Lanski 1987) и формализације идеја у позадини BDI архитектуре користећи модалне логике (Rao i Georgeff 1998). AgentSpeak агенти међусобно комуницирају разменом порука. ALAS је инспирисан овим концептом међуагентске комуникације. Главни језички конструкти AgentSpeak језика су веровања, циљеви и планови. Циљеви могу бити *остварени* или у виду *теста*. Тест циљеви једноставно проверавају базу веровања и утврђују да ли су неке чињенице тачне или нетачне. *Остварени циљеви*, с друге стране, покрећу извршење планова (Bordini, Hübner i Wooldridge 2007). ALAS се такође заснива на језичким конструктима као и AgentSpeak. Први програмски конструкти AgentSpeak-а су скупови агената аналогно класама у OOP. Иако никад није био стављен у практичну употребу, неки од његових концепата, као што је разлика између приватних и јавних сервиса, утицали су на развој ALAS-а. Међутим, AgentSpeak постаје све популарнији због развоја Jason[7] интерпретера за проширену верзију AgentSpeak-а (Bordini i dr. 2009; Wang 2013).

Нови AOPL заснован на принципима AgentSpeak-а је ASTRA (2017) и развијен је како би олакшао програмерима који се баве OOP упознавање AOP технологија. Као и AgentSpeak, ASTRA се заснива на Jason интерпретеру. Синтакса ASTRA језика се заснива на Java програмском језику. Поља, методе, наредбе, петље и др. из Jave се мапирају у одговарајуће конструкте у ASTRA језику (Collier, Russell i Lillis 2015a, 2015b). На истом овом принципу функционише и ALAS чија је синтакса такође заснована на Javi.

Један од AOPL који се користи за развој когнитивних агената је GOAL (Hindriks 2017). GOAL агенти су повезани са својим окружењем и међусобно комуницирају преко порука. GOAL се заснива на правилима што је послужило као инспирација за развој ALAS-а чија је граматика такође заснована на правилима. Акције које нуди GOAL су дефинисане условима. Агенти добијају

[6] http://www.fipa.org
[7] http://jason.sourceforge.net

информације о свом окружењу кроз когнитивне функције опажања и на тај начин могу послати захтев окружењу да изврши одговарајуће акције. Когнитивни агенти су аутономни агенти који доносе одлуке и извршавају акције на основу својих веровања и циљева (Hindriks 2017). GOAL се појавио као проширена верзија 3APL језика (Dastani i dr. 2004) који је заснован на BDI архитектури, али се више не развија. Већина GOAL концепата, укључујући главне особине GOAL агената као што су веровања, знање и услови, у одређеној мери су послужили при развоју ALAS-а.

Поред GOAL језика, 2APL (Dastani 2008) је такође настао на концептима 3APL. Једна од најважнијих особина 2APL је раздвајање мултигентских концепата и концепата појединачних агената. Мултиагентски део садржи скупове агената, спољних окружења као и везе између агената и спољних окружења. Овај програмски језик олакшава имплементацију мултиагентских система који се састоје од појединачних когнитивних агената. 2APL се разликује од других агентских програмских језика тако што има формалну семантику, остварујући ефективну интеграцију декларативног и императивног стилског програмирања. То се постиже увођењем низа практичних програмских конструката, укључујући и декларативне циљеве и догађаје (који се користе на другим програмским језицима). Основни концепти 2APL агената - веровања, циљеви, планирања, догађаји, поруке и правила налазе се и у основи ALAS језика.

Поред когнитивних способности, мобилност агената је суштинска особина агената. Међутим, остваривање мобилности може бити прилично сложен посао. SAFIN (Dianxiang, Guoliang i Xiaocong 1998) и S-CLAIM (Baljak i dr. 2012) (проширена верзија CLAIM језика (Suna i Fallah-Seghrouchni 2007) сакривају сложену подршку за мобилност агената од програмера. Ови језици сакривају функционалну сложеност од програмера пружајући им једноставне, али моћне програмске конструкте. Перформансе ова два језика су у великој мери помогле приликом дизајна ALAS конструката.

SARL (Feraud i Galland 2017; Rodriguez, Gaud i Galland 2014) је нови агентски програмски језик. Његови основни задаци су обезбеђивање апстракција које се баве дистрибуцијом, конкурентношћу, аутономијом, интеракцијом, реактивношћу, децентрализацијом и динамичком реконфигурацијом. Заснива се на програмским конструктима а развија се применом Xtext оквира за развој DSL-ова тако да се ALAS, што се тиче технологије и дизајна у почетној фази развоја угледао на SARL јер је и он развијан применом Xtext оквира.

За развој продуктивнијих агентских језика, поред особина које нуде постојећи агентски језици могу се применити и предности домен-оријентисаних језика. Интеграцијом концепата које нуде агентски и домен-оријентисани језици могуће је добити језик са оптималним могућностима који поседује све предности агентских и домен-оријентисаних језика описаних у претходном поглављу. У даљем тексту за интегрисани агентски и домен-оријентисани језик биће коришћен краћи назив: агентски, домен-оријентисани језик.

У поређењу са језицима опште намене ови језици нуде програмске конструкте (који представљају основну предност DSL-ова) за развој интелигентних агената. Основна намена програмских конструката је да сакрију сложеност одређених делова кода те на тај начин омогуће програмерима да се фокусирају на решавање конкретног проблема из одговарајућег домена. Такође, програмски код је једноставнији и читљивији за друге програмере. Велики број агентских језика међу којима је и ALAS је засновано на овим конструктима.

Међутим, до сада је развијен јако мали број агентских, домен-оријентисаних језика а неки од њих су наведени и укратко описани у наставку.

Један од првих DSL-ова за агенте, Agent-DSL је уведен у (Kulesza i dr. 2005). Agent-DSL је дефинисао разне особине агента које су му служиле како би могао да извршава одговарајуће задатке. Основне особине Agent-DSL агената су знање, интеракција, прилагодљивост, аутономија и колаборација.

DSL за развој софтверских агената који раде у оквиру окружења заснованих на семантичком вебу – SEA_L (енг. Semantic web Enabled Agent Language) (Demirkol i dr. 2012, 2013) је имао изузетно јак утицај на развој ALAS језика. SEA_L и ALAS имају пуно заједничких особина. У почетној фази ALAS је као и SEA_L развијан применом Xtext оквира за развој DSL-ова (Bettini 2013; Sredojević i dr. 2015; Sredojević, Vidaković, i Okanović 2015). И један и други језик су дизајнирани за програмирање софтверских агената у специфичним доменима примене.

Постоје разне области примене интелигентих агената. Једна од области која се рапидно развија је област рачунарских игара где се за доношење одлука које се користе у игри против корисника користе интелигентни агенти. Систем поседује одговарајући степен интелигенције како би кориснику игру учинио што занимљивијом. У (Hastjarjanto, Jeuring i Leather 2013) описан је интерни DSL, који представља проширење тј. библиотеку функционалног Haskell[8] програмског језика. Основна намена овог DSL-а је опис вештачке интелигенције у *real-time* видео играма. Језик нуди специјалне конструкте за развој игре засноване на вештачкој интелигенцији.

У (Cosenza 2018) представљен је OpenABL[9] – DSL за моделовање агената на паралелним и дистрибуираним аритектурама високих перформанси. То је једноставан језик за програмирање који се ослања на апстракције високог нивоа за програмирање који за постизање високих перформанси експлицитно користи паралелизам агената. Циљ OpenABL језика је да обезбеди преносиво, ефикасно и лако за коришћење окружење за моделовање агената. Ово се постиже напредним језиком који подржава конструкте специфичне за одређене домене примене, омогућавајући корисницима брзу израду прототипова, репродукцију и

[8] https://www.haskell.org
[9] https://github.com/OpenABL/OpenABL

упоређивање различитих модела са различитим параметрима. Језик такође пружа имплицитну подршку за паралелизам и локализацију агената, тако да се OpenABL програми могу ефикасно мапирати у паралелне и дистрибуиране имплементације.

Један од језика који је по својој спецификацији најсличнији ALAS језику је нови агентски, домен-оријентисани језик JADEL, дизајниран да олакша развој JADE агената и мултиагентских система. У (Bergenti, Iotti i Poggi 2016) представљене су основне карактеристике овог језика које се користе за развој JADE агената, понашања и онтологија. Као и ALAS, JADEL је настао како би се смањила сложеност развоја агената што је постигнуто развојем конструката и одговарајућих нотација за специфичне домене примене. JADE (енг. Java Agent DEvelopment framework) је агентска платформа која омогућава развој дистрибуираних мултиагентских система путем Java развојног оквира (Bellifemine, Caire i Greenwood 2007). JADE је тренутно најстабилније и најкоришћеније мултиагентско окружење отвореног кода. Siebog мултиагентски систем има доста сличности са JADE окружењем а неке разлике су описане у поглављу 4.2. Програмирање агената у оба окружења је пре развоја ALAS и JADEL језика било ограничен само на Java програмски језик.

Међутим, за потребе писања агената у Siebog-у не може се применити ни један постојећи агентски, домен оријентисани језик. Поред потребе за повећањем продуктивности и експресивности, основни разлог за развој ALAS-а је била чињеница да је Siebog (у оквиру којег се извршавају ALAS агенти) заснован на DNARS систему за развој интелигентних агената кога не подржава ни један постојећи агентски, домен-оријентисани језик.

ALAS је развијен на основу кључних особина језика описаних у овом поглављу и применом концепата на којима се они заснивају. Најчешћа разлика између ALAS-а и других агентских језика је што се већина тих језика заснива на BDI архитектури за разлику од ALAS-а чији су агенти засновани на NAL формализму развијеном у домену вештачке опште интелигенције.

Са развојем специјалних алата тј. језичких радионица (Fowler 2010) у великој мери је олакшан развој DSL-ова. Велики број додатних агентских и домен-оријентисаних језика описано је у радовима: (Badica i dr. 2011; Bordini i dr. 2009; Dastani 2008; Dastani i dr. 2004; Davies i Edwards 1994; Feraud i Galland 2017; Shoham 1993; Thomas 1995) и (Fowler 2010; Hudak 1998; Kosar i dr. 2010; Kosar, Bohra, i Mernik 2016) респективно.

Прве верзије ALAS језика описане су у следећим радовима (Mitrović, Ivanović i Vidaković 2011; Mitrović i dr. 2014c; Mitrović i dr. 2012) и првобитна идеја је била да се применом ALAS-а реши проблем мобилности агената који су се извршавали у оквиру XJAF окружења које је имплементирано применом Java програмског језика. Прва верзија ALAS-а је развијена применом Javacc (2017) парсера. Како би се унапредио дизајн и омогућиле веће могућности, ALAS је настављен да се развија применом Xtext оквира за развој DSL који је свакако један од модернијих и најпознатијих алата за развој DSL. Xtext је најпогоднији за развој језика

заснованих на Java синтакси па је и то један од разлога зашто је баш Xtext одабран за развој ALAS-а јер су се и ALAS агенти извршавали у окружењу заснованом на Java језику. Xtext омогућује једноставну конверзију ALAS кода у Java језик (Sredojević i dr. 2015; Sredojević, Vidaković, i Okanović 2015).

Међутим, проблем је настао када је дошло до потребе да се ALAS агенти путем мреже пребацују и извршавају на разним платформама имплементираним у различитим програмским језицима. Овај проблем хетерогене мобилности је захтевао конверзију ALAS кода не само у Javu већ и у неке друге програмске језике као што су JavaScript, Python итд. Процес конверзије у не-Java језике применом Xtext окружења је прилично сложен јер захтева детаљно упознавање са могућностима које нуди Xtext када је генерисање кода у питању. Без обзира на то, дошло се до неких других решења која делимично решавају проблем конверзије. На пример, за процес генерисања JavaScript кода коришћен је прво Java код генерисан од стране Xtext-а а затим Google Web Toolkit – GWT[10] који на основу Java кода генерише JavaScript код (Sredojević i dr. 2016). Међутим, генерисани код је био у великој мери неразумљив и немогућ је био било какав даљи рад са њим. Како би решили, тј. олакшали процес конверзије ALAS кода у било који програмски језик и на тај начин решили проблем мобилности агената и платформске независности, последња верзија ALAS-а је развијена применом новог и знатно једноставнијег textX оквира за развој DSL (Dejanović 2019b; Sredojević, Vidaković i Ivanović 2018). За разлику од Xtexta, textX нуди једноставне механизме за конверзију и програмерима пружа велику слободу приликом конструкције генератора кода. Језици развијени применом textX-а као што су: pyFlies[11], pyTabs (Simić i dr. 2015), DProfLang (Vaderna i dr. 2015) и Applang (Kosanović, Dejanović i Milosavljević 2016) су у великој мери помогли у развоју ALAS-а.

[10] http://www.gwtproject.org
[11] http://www.igordejanovic.net/pyFlies

3 NAL

Не-аксиоматска логика представља формализам за резоновање у домену вештачке опште интелигенције (Wang i Awan 2011; Wang 2006, 2013). За разлику од неких других формализама у рачунарству, NAL је логика термова и реченице су дате у форми: *субјекат-релација-предикат*. NAL садржи скуп правила извођења на основу којих се изводе нова знања и дају одговори на постављена питања.

Појам *не-аксиоматска* означава да је логика погодна за развој система који функционишу у условима недовољног знања и ресурса (Wang i Awan 2011; Wang 2006, 2013). Применом NAL-а могу се доносити закључци на основу знања система које може бити несигурно и неконзистентно. Нови докази се могу појавити у било ком моменту, могу имати било какав садржај и могу променити истинитост било које постојеће реченице. Поред тога, систем често нема довољно ресурса (у смислу времена, меморијског простора, итд.) да консултује своје целокупно знање како би решио проблем и често се за доношење закључака користи недовољан број реченица из базе знања. Додатно, систем не може применити пун скуп правила за извођење, нити пратити неки унапред задати алгоритам (Mitrović 2015).

NAL се бави претходно наведеним проблемима применом уграђених механизама. NAL може ефикасно управљати неконзистентним базама знања и сумирати постојеће знање и на тај начин умањити укупан број сувишних реченица у бази знања. Међутим, с обзиром да је NAL логика термова, њен систем рада се разликује од многих других логика које се користе за резоновање у интелигентним системима (Smith 2017; Sommers i Englebretsen 2000; Wang 2013). Као што је раније наведено, NAL реченице су у облику: *субјекат-релација-предикат*, при чему субјекат и предикат могу бити атомски или сложени термови.

NAL логика се може представити у 9 нивоа:

- NAL-1: Правила наслеђивања (енг. Inference rules)
- NAL-2: Релација сличности (енг. Similarity copulas)

- NAL-3: Сложени термови (енг. Compound terms)
- NAL-4: Релациони термови (енг. Relational terms)
- NAL-5: Термови у виду реченица (енг. Statements as terms)
- NAL-6: Променљиве (енг. Variable terms)
- NAL-7: Догађаји као реченице (енг. Events as statements)
- NAL-8: Операције и циљеви као догађаји (енг. Operations and goals as events)
- NAL-9: Самоконтрола и самонадзор (енг. Self-control and self-monitoring)

при чему су сви нивои детаљно описани у (Wang 2006, 2013).

Сваки ниво нуди нове особине и правила извођења укључујући особине свих претходних нивоа при чему сваки ниво повећава експресивност логике а самим тим и степен интелигенције система који је на њој заснован.

3.1 Синтакса реченица прва 4 нивоа NAL

DNARS описан у (Mitrović 2015) имплементира концепте и правила извођења прва четири нивоа NAL што је довољно за обављање когнитивних задатака са практичном применом.

NAL-1 уводи реченице засноване на релацијама између тзв. атомских тј. простих термова. У оквиру овог нивоа дефинисана је само директна веза између термова. То је релација наслеђивања која је у DNARS означена '–>' симболом. Пример једне реченице коју чине два атомска терма и релација наслеђивања је: *наранџа –> воће*. Ова реченица се тумачи као: *наранџа је врста воћа*. У NAL је релација наслеђивања као и по дефиницији транзитивна и рефлексивна.

Поред релације наслеђивања, која постоји и у свим наредним нивоима, NAL-1 уводи дводимензионалну истинитосну вредност реченица (f, c) из опсега [0, 1], која се пише на крају сваког веровања тј. реченице, при чему f означава учесталост а c стабилност учесталости кад се појаве нови докази. Додавањем истинитосне вредности (нпр. f = 0.8 и c = 0.7), претходно поменута реченица имаће следећи облик: *наранџа –> воће* (0.8, 0.7).

Свака база знања може садржати и позитивна и негативна веровања. На основу формуле (2), учесталост (f) се дефинише као однос позитивних (ω^+) и укупних доказа (ω) а стабилност (c) представља проценат тренутних доказа (ω) међу свим расположивим доказима у блиској будућности ($\omega + 1$). На пример, ако у бази знања постоји $\omega^+ = 95$ реченица *лимун је жут* и 5 реченица *лимун је плав*, учесталост за реченицу *лимун је жут* ће бити $f = 0.95$. У овом случају стабилност је $c = 0.99$. С обзиром на високу учесталост и стабилност која тежи 1, веровање *лимун је жут* има велику истинитосну вредност а самим тим и тачност.

$$\omega = \omega^+ + \omega^- \tag{1}$$

$$f = {\omega^+}/{\omega} \tag{2}$$

$$c = {\omega}/{(\omega + 1)} \tag{3}$$

NAL изводи закључке и одговоре на питања на основу правила извођења при чему у зависности од врсте задатака постоје три групе правила извођења (Wang 2006, 2013): *извођења унапред* (енг. *forward inference*) изводе нове закључке, *извођења уназад* (енг. *backward inference*) дају одговоре на питања и *локална извођења* (енг. *local inference*) раде са неконзистентним реченицама.

Три основна правила *извођења унапред* у NAL-1 су дедукција, абдукција и индукција која су детаљније описана у (Mitrović 2015; Wang 2013). Применом ових правила закључци се изводе на следећи начин:

	дедукција	**абдукција**	**индукција**
прва реченица:	*M –> П*	*П –> M*	*M –> П*
друга реченица:	*C –> M*	*C –> M*	*M –> C*
***закључак*:**	***C –> П***	***C –> П***	***C –> П***

где *C*, *П* и *M* представљају просте или сложене термове. Разлика између дедукције, индукције и абдукције се највише огледа у стабилности доказа (*c*). Код дедукције, стабилност је најчешће већа од 0.5 и знатно се спорије мења (опада) при појави нових доказа за разлику од индукције и абдукције код којих је стабилност углавном мања од 0.5 и брже се мења (опада) услед нових доказа. Због веће стабилности, докази изведени применом правила дедукције се још зову и *јаки докази*, а докази изведени индукцијом или абдукцијом - *слаби докази*. Примери примене ових правила приказани су у табели 3.1.

Табела 3.1. Дедукција, абдукција и индукција у NAL-1

Правила извођења	**Веровања**	**Закључак**
дедукција	*наранџа –> воће* (f_1, c_1) *мандарина –> наранџа* (f_2, c_2)	*мандарина –> воће* (*f*, *c*)
абдукција	*цитрус –> воће* (f_1, c_1) *лимун –> воће* (f_2, c_2)	*лимун –> цитрус* (*f*, *c*)
индукција	*мандарина –> воће* (f_1, c_1) *мандарина –> наранџа* (f_2, c_2)	*наранџа –> воће* (*f*, *c*)

NAL-2 уводи релацију сличности која се у DNARS означава знаком ‘~’ при чему је сличност између термова рефлексивна, симетрична и транзитивна. Пример реченице са релацијом сличности је: *мандарина ~ клементина* (0.9, 0.8) при чему истинитосна вредност (0.9, 0.8) означава степен сличности мандарине и клементине. С обзиром да је релација сличности симетрична, реченица *мандарина ~ клементина* (0.9, 0.8) је еквивалентна реченици *клементина ~ мандарина* (0.9, 0.8).

Три основна правила *извођења унапред* која уводи NAL-2 су: поређење, аналогија и наликовање (енг. *resemblance*). Применом ових правила закључци се изводе на следећи начин:

	поређење	**аналогија**	**наликовање**
прва реченица:	*М –> П*	*М ~ П*	*М ~ П*
друга реченица:	*М –> С*	*С –> М*	*С ~ М*
закључак:	***С ~ П***	***С –> П***	***С ~ П***

где *С*, *П* и *М* представљају просте (атомске) или сложене термове. Примери примене ових правила су приказани у табели 3.2.

Табела 3.2. Поређење, аналогија и наликовање у NAL-2

Правила извођења	**Веровања**	**Закључак**
поређење	*клементина –> наранџа* (f_1, c_1) *клементина –> мандарина* (f_2, c_2)	*мандарина ~ наранџа* (f, c)
аналогија	*наранџа –> воће* (f_1, c_1) *мандарина ~ наранџа* (f_2, c_2)	*мандарина –> воће* (f, c)
наликовање	*мандарина ~ наранџа* (f_1, c_1) *клементина ~ мандарина* (f_2, c_2)	*клементина ~ наранџа* (f, c)

Поред основних правила извођења наведених у табелама 3.1 и 3.2, постоје и сложенија правила извођења која су описана у (Wang 2006, 2013). Међутим, у дисертацији ће се користити само основна правила која су довољна за доношење одговарајућих закључака. Применом одређеног правила, приликом извођења закључака, мења се истинитосна вредност закључка у односу на реченице на основу којих је тај закључак изведен. Истинитосне вредности (f, c) из табеле 3.1 и табеле 3.2 се рачунају на основу одговарајућих формула описаних у (Wang 2013). Ове формуле (наведене само за поменута правила извођења) су приказане у табели 3.3. k је позитивна константа чија је предложена вредност $k = 1$ (Wang 2013).

Табела 3.3. Рачунање истинитосних вредности за основна правила *извођења унапред*

Правила извођења	**Учесталост (*f*)**	**Стабилност (*c*)**
дедукција	$f_1 f_2$	$f_1 f_2\ c_1 c_2$
индукција	f_1	$\frac{f_2 c_1 c_2}{f_2 c_1 c_2 + k}$
абдукција	f_2	$\frac{f_1 c_1 c_2}{f_1 c_1 c_2 + k}$
поређење	$\frac{f_1 f_2}{f_1 + f_2 - f_1 f_2}$	$\frac{(f_1 + f_2 - f_1 f_2) c_1 c_2}{(f_1 + f_2 - f_1 f_2) c_1 c_2 + k}$
аналогија	$f_1 f_2$	$f_2 c_1 c_2$
наликовање	$f_1 f_2$	$(f_1 + f_2 - f_1 f_2) c_1 c_2$

Као што је већ наведено, за разлику од многих формализама који се користе у агентској технологији, NAL успешно може изводити закључке на основу неконзистентних знања и то применом два *локална* правила извођења, *ревизије* (енг. *revision*) и *избора* (енг. *choice*) (Wang 1996). Две реченице су неконзистентне ако имају исте субјекте, предикате и објекте а различите истинитосне вредности. *Ревизија* се користи када се у бази знања појави више истих веровања са различитим истинитосним вредностима. Применом *ревизије* од више веровања се изводи једно са новом истинитосном вредношћу при чему се учесталост и стабилност рачунају по формулама датим у наставку. Ако на пример, у бази знања постоје две реченице са истинитосним вредностима (f_1, c_1) і (f_2, c_2), применом правила *ревизије* добиће се реченица са новом истинитосном вредношћу (f, c) где се f и c рачунају по следећим формулама:

$$f = \frac{f_1 c_1 (1 - c_2) + f_2 c_2 (1 - c_1)}{c_1 (1 - c_2) + c_2 (1 - c_1)}$$

$$c = \frac{c_1 (1 - c_2) + c_2 (1 - c_1)}{c_1 (1 - c_2) + c_2 (1 - c_1) + (1 - c_1)(1 - c_2)}$$

NAL-3 уводи сложене термове који се састоје од једног или више атомских термова међусобно повезаних конектором.

Дефиниција 3.1. Сложени терм има форму {T_1 *кон* T_2 *кон* ... *кон* T_n}, где је *кон* конектор, а $T_1 \ldots T_n$ су термови, за $n \geq 1$ (Ванг 2006, 2013).

У дефиницији 3.1 је представљена инфиксна нотација сложеног терма. Префиксна нотација има следећи облик: {*кон* T_1 T_2 ... T_n}. T_1, T_2 ... T_n могу бити и прости и сложени термови.

NAL-4 уводи нови конектор - *производ*, који је означен са (*x*) и служи да подржи произвољне релације између термова.

Пример структуре реченице која се састоји од једног сложеног терма и једног атомског терма је (*слон x биљке*) –> *једе* (0.95, 0.9) при чему је (*слон x биљке*) сложени терм који се састоји од два атомска терма међусобно повезана релацијом *једе* где релација *једе* представља *релациони* терм описан у дефиницији 3.2.

Дефиниција 3.2. Релациони терм *R* се дефинише као атомски терм који је са сложеним производним термом повезан релацијом наслеђивања на следећи начин: (T_1 *x* T_2) –> *R* (*f*, *c*) или *R* –> (T_1 *x* T_2) (*f*, *c*) (Wang, 2006, 2013).

Међутим, како би NAL извео што ефикасније закључке, често је потребно променити структуру реченица. На пример, применом дедукције, на основу реченица (*биљојед x биљке*) –> *једе* (f_1, c_1) и *слон* –> *биљојед* (f_2, c_2) систем би требало да закључи: (*слон x биљке*) –> *једе* (*f*, *c*). Ово није изводљиво пошто систем не може да препозна терм *биљојед* као субјекат прве реченице. Да би овај проблем био решен уведена су два нова конектора, *екстензивни* (/) и *интензивни* (\\) на основу којих се добијају четири нове слике ((3) и (4)) исте реченице које су дефинисане на следећи начин:

$$((T_1\ x\ T_2)\ \text{–>}\ R\ (f, c)) \Leftrightarrow (T_1\ \text{–>}\ (/\ R\ *\ T_2)\ (f, c)) \Leftrightarrow (T_2\ \text{–>}\ (/\ R\ T_1\ *)\ (f, c)) \qquad (3)$$

$$(R\ \text{–>}\ (T_1\ x\ T_2)\ (f, c)) \Leftrightarrow ((\backslash\backslash\ R\ *\ T_2)\ \text{–>}\ T_1\ (f, c)) \Leftrightarrow ((\backslash\backslash\ R\ T_1\ *)\ \text{–>}\ T_2\ (f, c)) \qquad (4)$$

при чему (*) означава позиције термова T_1 или T_2.

Применом (3) на реченицу *слон једе биљке* добијају се три слике:

- (*слон x биљке*) –> *једе* (*f*, *c*)
- *слон* –> (/ *једе* * *биљке*) (*f*, *c*)
- *биљке* –> (/ *једе слон* *) (*f*, *c*)

Такође, применом правила (4), реченица *слон једе биљке* се може представити у још три нове слике (реченице) које имају исто значење али различиту структуру:

- *једе* –> (*слон x биљке*) (*f*, *c*)
- (\\ *једе* * *биљке*) –> *слон* (*f*, *c*)
- (\\ *једе слон* *) –> *биљке* (*f*, *c*)

ALAS језик подржава свих 6 формата синтаксе NAL реченица са сложеним термовима приказаних у (3) и (4). Поред ових формата, ALAS подржава још две форме са атомским термовима:

$$T_1 \text{ –> } T_2 \ (f, c) \tag{5}$$

$$T_1 \sim T_2 \ (f, c) \tag{6}$$

У наставку су приказана два примера реченица које се могу креирати на основу (5) и (6):

- *мандарина –> воће* (*f*, *c*)
- *мандарина ~ клементина* (*f*, *c*)

На основу реструктурирања реченица и креирањем нових слика систем лакше може да изводи закључке. Сада систем може на основу реченица *биљојед једе биљке* (са истинитосном вредношћу (0.98, 0.92)) и *слон је биљојед* (са истинитосном вредношћу (0.94, 0.9)) да изведе закључак да *слон једе биљке* и то је описано у наставку.

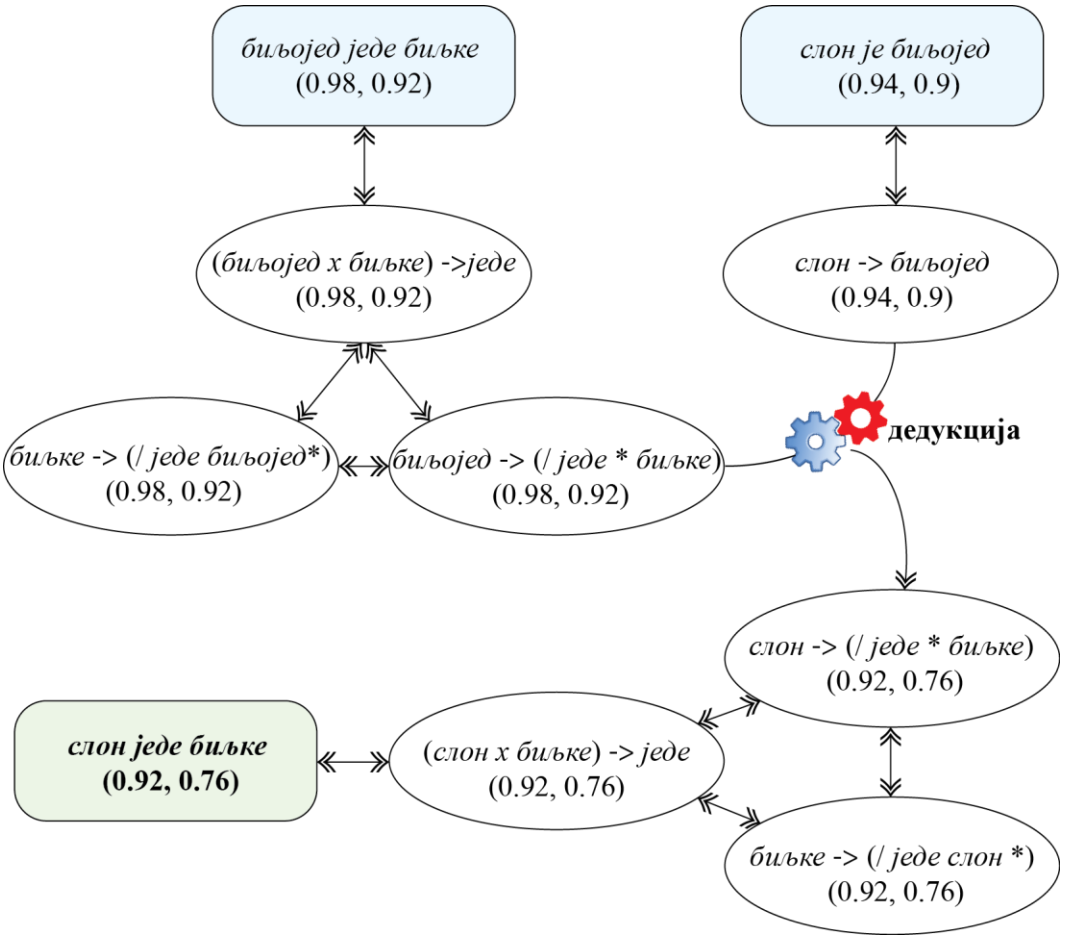
Реченица *слон је биљојед* се лако преводи у NAL формат: *слон –> биљојед* (0.94, 0.9).

Реченица *биљојед једе биљке* се применом NAL реструктурира у (*биљојед x биљке*) –> *једе* (0.98, 0.92). Применом правила (3) ова реченица се може превести у следећи облик: *биљојед –>* (*/ једе * биљке*) (0.98, 0.92). Сада систем применом дедукције лако може закључити следеће: *слон –>* (*/ једе * биљке*) (*f*, *c*) тј. применом правила (3): (*слон x биљке*) –> *једе* (*f*, *c*) а даљим превођењем у људски разумљив формат добија се: *слон једе биљке* (са истинитосном вредношћу (0.92, 0.76)). Систем аутоматски на основу одговарајућих формула за дедукцију рачуна учесталост и стабилност (*f*, *c*) новонасталог веровања (закључка) на следећи начин:

$$f = f_1 * f_2 = 0.94 * 0.98 = 0.92$$

$$c = f * c_1 * c_2 = 0.92 * 0.92 * 0.9 = 0.76$$

Визуелни приказ овог примера се може видети на слици 3.1.



Слика 3.1. Визуелни приказ извођења закључка применом дедукције

Као што је раније наведено, ALAS подржава прва четири нивоа NAL док остали нивои представљају шредмет будућих истраживања.

4 Siebog мултиагентска платформа

Siebog је мултиагентска платформа која пружа инфраструктуру за извршавање агената у веб окружењима у складу са савременим стандардима и спецификацијама. Siebog комбинује савремене принципе развоја серверских и клијентских система у јединствен програмски оквир за развој интелигентних агената. Siebog укључује две основне компоненте – серверску XJAF (прошириво Java EE агентско окружење) (Mitrović i dr. 2014b, 2016a; Vidaković i dr. 2013) и клијентску Radigost (Mitrović i dr. 2014a). Обе компоненте имају своје предности а њиховом комбинацијом Siebog добија додатне могућности.

Комбинујући особине које пружају XJAF и Radigost, Siebog у односу на друга мултиагентска окружења нуди значајне предности као што су:

- Вишеплатформска размена порука: клијентски агенти могу несметано комуницирати са серверским агентима, па чак и клијентским агентима на другим уређајима.
- Дељење кода: једном написан агент се може, без измена, извршавати и на серверу и на клијенту.
- Хетерогена мобилност: агенти се могу несметано кретати између сервера и клијената.

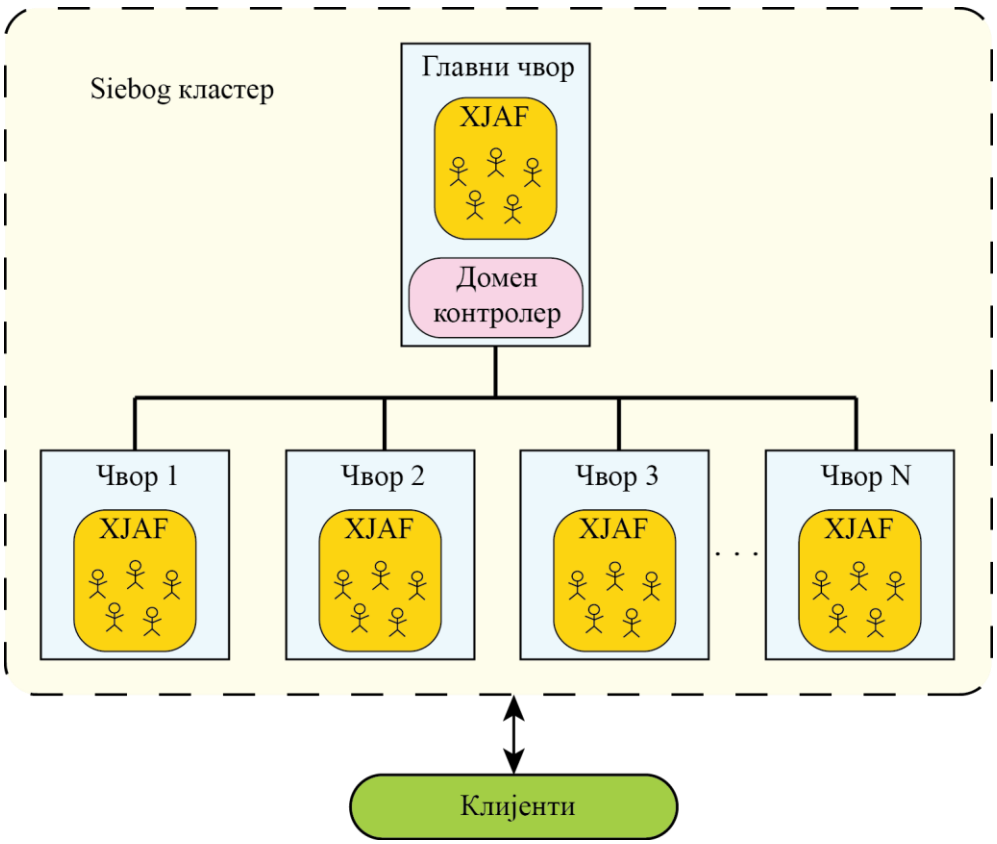
4.1 XJAF – серверска страна Siebog-а

Као што је приказано на слици 4.1, Siebog је дизајниран тако да се на серверској страни може извршавати у кластерима рачунара што му доноси две најбитније предности:

- Скалабилност, што подразумева аутоматско распоређивање агената по чворовима кластера чиме се смањује појединачно оптерећење рачунара. Ово омогућује извршавање огромног броја агената у рачунарском кластеру.

– Отпорност на грешке – стања агената се копирају на преостале чворове кластера тако да уколико дође до квара на неком рачунару, агент ће несметано наставити да се извршава на неком другом рачунару у кластеру.

Велики број других мултиагентских система такође подржава ове две особине али често на неефикасан начин као што је тај да корисници сами морају распоређивати агенте по рачунарима док се у случају Siebog-а то одвија аутоматски (Alberola i dr. 2013; Faci i dr. 2006).



Слика 4.1. Симетрични кластер чворова на серверској страни Siebog-а

У случају Siebog-а, чворови (рачунари) се налазе у симетричном кластеру где су сви чворови међусобно повезани. Главни - мастер чвор је повезан са свим другим чворовима и има исте особине, при чему он има и додатну могућност контролисања целог кластера кроз JBoss домен контролер (Mitrović i dr. 2014a).

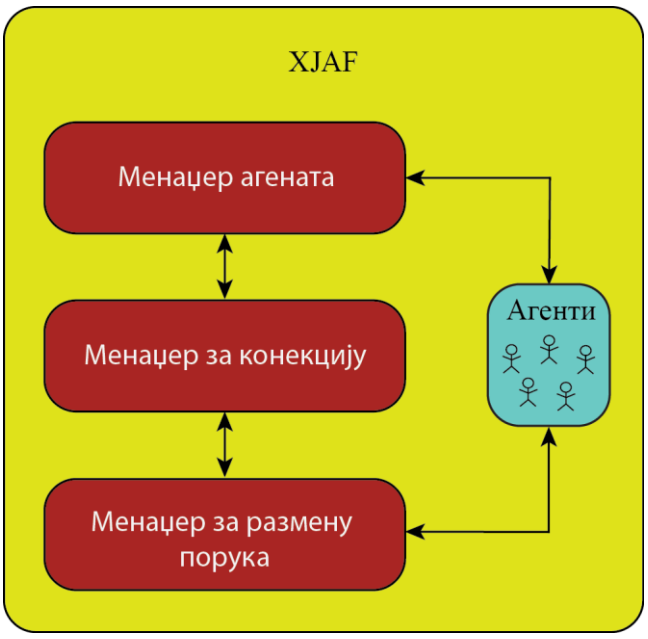
Као што је раније наведено, серверска страна Siebog-а је заснована на претходно развијеном систему XJAF који је први пут представљен у (Vidaković i Konjović 2002) али је у међувремену надограђиван (Mitrović i dr. 2014b; Vidaković i dr. 2013).

XJAF је агентско окружење засновано на Java EE технологији и то је једна од основних предности овог окружења у односу на друге мултиагентске системе. Његови основни задаци су да обезбеди ефикасно извршно окружење за своје агенте, као и да обезбеди спољним клијентима да лако приступају овој агентској

технологији. XJAF користи постојеће, стандардизоване Java EE технологије, алате и библиотеке као што су JNDI, JMS и EJB да би могао имплементирати велике подскупове функционалности неопходних за мултиагентске системе. Избор Java EE као XJAF имплементационе платформе се показао као врло користан. Основне предности овог приступа су краће развојно време самог система и овладавање напредних програмских функција као што је расподела оптерећења. Поред расподеле оптерећења, XJAF подржава и кластеризацију. Прецизније, XJAF се за своје функционисање ослања на следеће Java EE технологије (Vidaković i dr. 2013):

- JNDI (енг. Java Naming and Directory Interface): користи се за имплементирање директоријума агената и сервиса
- JMS (енг. Java Message Service): обезбеђује комуникациону инфраструктуру (размена порука између агената)
- EJB (енг. Enterprise JavaBeans): користе се као „чувари места - за агенте и сервисе
- Серијализација: подршка мобилности и перзистентности агената - Java PKI API: користи се за имплементацију неких безбедносних система
- JAXB (енг. Java Architecture for XML Binding): поједностављује коришћење задатака описаних XML језиком.

XJAF је пројектован као модуларни систем (слика 4.2) који се састоји од модула, односно менаџера.



Слика 4.2. Главне компоненте XJAF окружења

Сваки менаџер је релативно независан модул задужен за руковање посебним деловима целокупног процеса управљања агентима. Постоји неколико предности

модуларног приступа. На пример, функционалност система може се лако проширити додавањем нових менаџера. Три основна менаџера која постоје у оквиру XJAF окружења су: менаџер агената, менаџер за размену порука и менаџер за конекцију.

Главни менаџер XJAF окружења - менаџер агената (енг. Agent manager) је задужен за управљање животним циклусом агената и организовање директоријума агената. Менаџер агената укључује функционалности које одговарају FIPA сервису директоријума агената (FIPA 2018). Његови главни задаци су да управљају директоријумом агената и да активирају и деактивирају агенте на захтев. Да би подржао мобилност агента, менаџер агената користи два директоријума: један за агенте који су доступни локално, а други за агенте који су напустили тренутну инстанцу и пребацили се на удаљену инстанцу XJAF-а. Касније се агент може лако лоцирати вађењем евиденције из првог директоријума или прослеђивањем захтева удаљеном XJAF-у који је био сачуван у другом директоријуму.

За размену порука и комуникацију између агената задужен је менаџер за размену порука (енг. Message manager). Он користи Java сервис за размену порука и подржава три типа комуникације:

1. Агент ↔ XJAF, сваки агент има референцу на инстанцу XJAF-а и може директно позвати његове методе. С друге стране, систем може да комуницира са агентом тако што ће му послати поруку.
2. Агент ↔ Агент, где се оба агента налазе у истом XJAF-у. Ова локална комуникација између агената се обавља коришћењем порука, уз помоћ менаџера за размену порука.
3. Агент ↔ Агент, где се агенти дистрибуирају преко мреже. Ова удаљена међуагентска комуникација такође се ослања на размену порука применом менаџера за размену порука. Додатно, у овом сценарију се користи менаџер за конекцију који омогућује слање поруке удаљеном агенту.

Када је развијена прва верзија XJAF система, 2003. године (Vidaković 2003), за размену порука и међуагентску комуникацију коришћен је KQML протокол (Finin i dr. 1994). Међутим, временом је развијен нови стандард за комуникацију - FIPA ACL (FIPA 2018) који је поред XJAF-а усвојен и у многим другим мултиагентским системима.

За повезивање дистрибуираних инстанци XJAF-а у јединствен систем користи се менаџер за конекцију (енг. Connection manager) (Mitrović i dr. 2014b). Поред функционалности које нуди менаџер агената који служи као подршка за мобилност агента, XJAF укључује оквир за међусобно повезивање дистрибуираних инстанци окружења. Главна мотивација за мрежну организацију XJAF инстанци је расподела оптерећења. Уместо извршавања великог броја агената на једној машини, много је ефикасније користити више рачунара, а затим расподелити агенте међу њима.

На страни сервера, Siebog уводи још једног менаџера за управљање клијентским агентима – веб клијент менаџер. Он служи као посредник за слање порука између сервера и клијента а такође управља и чувањем стања клијентских агената.

У оригиналној имплементацији, дистрибуиране XJAF инстанце су организоване у структуру налик стаблу, са примарним XJAF-ом који представља корен стабла. Касније је структура налик стаблу замењена потпуно повезаним графом, у настојању да се повећа отпорност на грешке у оквиру XJAF мреже.

Недостатак XJAF компоненте Siebog-a је ограниченост на Java програмски језик и чињеница да Java агенти не могу комуницирати са агентима који се извршавају у неком другом окружењу које није имплементирано у Javi. Да би се решио проблем интероперабилности и омогућила његова шира употреба, XJAF је првобитно редизајниран као сервисно оријентисана инфраструктура (енг. Service-Oriented Architecture - SOA). Нови мултиагентски систем заснован на SOA назван SOM (SOA - based MAS), задржао је Java имплементациону платформу али су менаџери редефинисани као веб сервиси (Mitrović, Ivanović i Vidaković 2011). Међутим, јавио се нови проблем при извршавању агената у оквиру више платформи које нису имплементиране у истом програмском језику. Ако је агент написан у Javi и извршава се у мултиагентском окружењу имплементираном такође у Javi, не може се пребацити и извршавати у другом окружењу које је имплементирано нпр. у JavaScript или Python програмском језику. Да би се у потпуности решио проблем интероперабилности тј. платформске независности, развијен је универзални језик за развој софтверских агената – ALAS (Mitrović, Ivanović i Vidaković 2011; Mitrović i dr. 2014c; Okanović i dr. 2014). Основна сврха ALAS-a је да омогући имплементацију и подржи извршавање агената у хетерогеним окружењима. Да би се дефинисала структура језика, креирани су мета-модел и граматика ALAS језика у складу са потребама агената.

4.2 Radigost – клијентска страна Siebog-a

На клијентској страни Siebog нуди низ погодности које му омогућују платформску независност и функционисање на великом броју хардверских и софтверских платформи. За функционисање клијентске стране Siebog-a користе се веб претраживачи без инсталације било каквих додатака, што је највећа предност клијентске стране Siebog-a у односу на друге мултиагентске платформе међу којима су и неке од најзаступљенијих, JADE (Bellifemine, Caire i Greenwood 2007; JADE 2017), JaCaWeb (Minnoti, Santi i Ricci 2010) итд., које од корисника захтевају инсталацију одговарајућих додатака. Ова предност Sieboga је проистекла из чињенице да је Radigost, као клијентска страна Siebog-a, развијен применом савремених технологија као што су HTML5 и JavaScript који су подржани од стране скоро свих веб претраживача. То омогућује Radigost агентима да се извршавају на великом броју хардверских и софтверских уређаја

као што су паметни телефони, персонални рачунари, таблет уређаји, паметни телевизори итд.

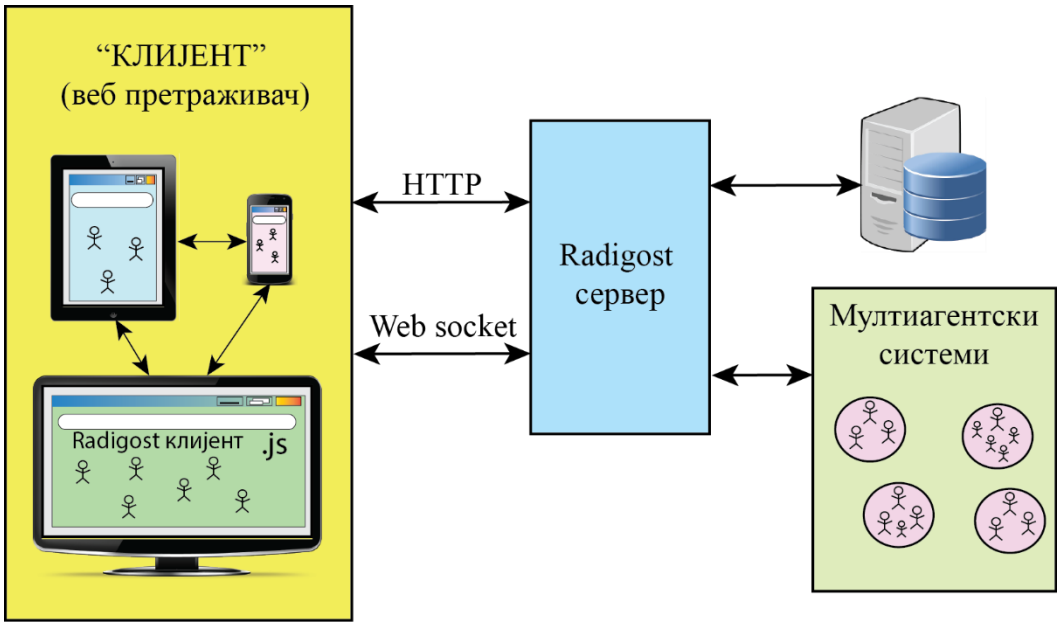
Једна од кључних карактеристика агената је *друштвена интеракција*. У Radigost-у, ова интеракција се постиже преко асинхроне размене порука. За максималне перформансе и интероперабилност, платформа користи веб воркере (енг. Web Workers) - постојећу, стандардизовану инфраструктуру за размену порука.

Radigost, као мултиагентска платформа, пружа архитектурну подршку за извршавање и интеракцију својих агената. Његове основне функционалности укључују управљање животним циклусом агената, комуникацијску инфраструктуру и услугу жутих страна. На пример, агенти могу пронаћи друге (активне или неактивне) агенте, створити нове инстанце агената и ступити у интеракцију са агентима који се извршавају не само унутар система, већ и у неким другим компатибилним мултиагентским системима у мрежи.

Општа архитектура Radigost-а је приказана на слици 4.3. Платформа укључује клијентску и серверску страну. Клијентска страна се извршава унутар веб претраживача и састоји се од агената и клијентске библиотеке. Клијентска библиотека нуди већину функционалности платформе агентима и програмерима. На пример, обезбеђује прототип агента који дефинише основне функционалности за све агенте. Библиотека такође укључује неопходне функције које подржавају међуагентску комуникацију путем размене порука. Поред тога, библиотека пружа подршку за комуникацију између агента и клијентске апликације (нпр. веб страница), која се остварује кроз добро познати шаблон за надгледање дизајна софтвера (енг. Observer software design pattern).

На серверској страни Radigost-а постоје три основне компоненте: менаџер стања, менаџер директоријума и гејтвеј (енг. Gateway) (Mitrović i dr. 2014a).

Један од проблема са којим се суочавају мултиагентски системи а који се јавио и приликом развоја Siebog-а је чињеница да животни век агента директно зависи од веб странице у којој се извршава. На пример, ако корисник затвори страницу или учита неку другу, агент престаје да постоји. У случају Siebog-а овај проблем се решава помоћу менаџера стања који омогућује аутоматско копирање стања клијентских агената на сервер. Када корисник касније поново учита страницу, стање се аутоматски враћа у агента тако да агент наставља да се извршава као да и није дошло до затварања странице.



Слика 4.3. Општа архитектура Radigost платформе

Менаџер директоријума имплементира стандардну услугу жутих страна. Он омогућује агентима да региструју и објављују описе својих функционалности и претражују друге регистроване агенте.

Гејтвеј је компонента на серверској страни Radigost-а која повећава интероперабилност система омогућујући његовим агентима да без проблема интерагују са агентима који се извршавају у оквиру других мултиагентских система. Гејтвеј се састоји од две специјализоване под-компоненте: *bridge* и *socket*. Bridge пресликава име агента и трансформише FIPA ACL поруке. Socket компонента представља канал за размену порука између Radigost-а и неког другог мултиагентског система. Socket, заснован на WebSocket протоколу, омогућује потпуну обострану комуникацију: на пример, Radigost агенти не само да могу слати поруке JADE агентима, већ и JADE агенти могу покренути интеракцију са било којим Radigost агентом који ради у било којем клијенту у мрежи. Ова особина је веома важна, јер значајно повећава практичну примену Radigost-а а самим тим и Siebog-а.

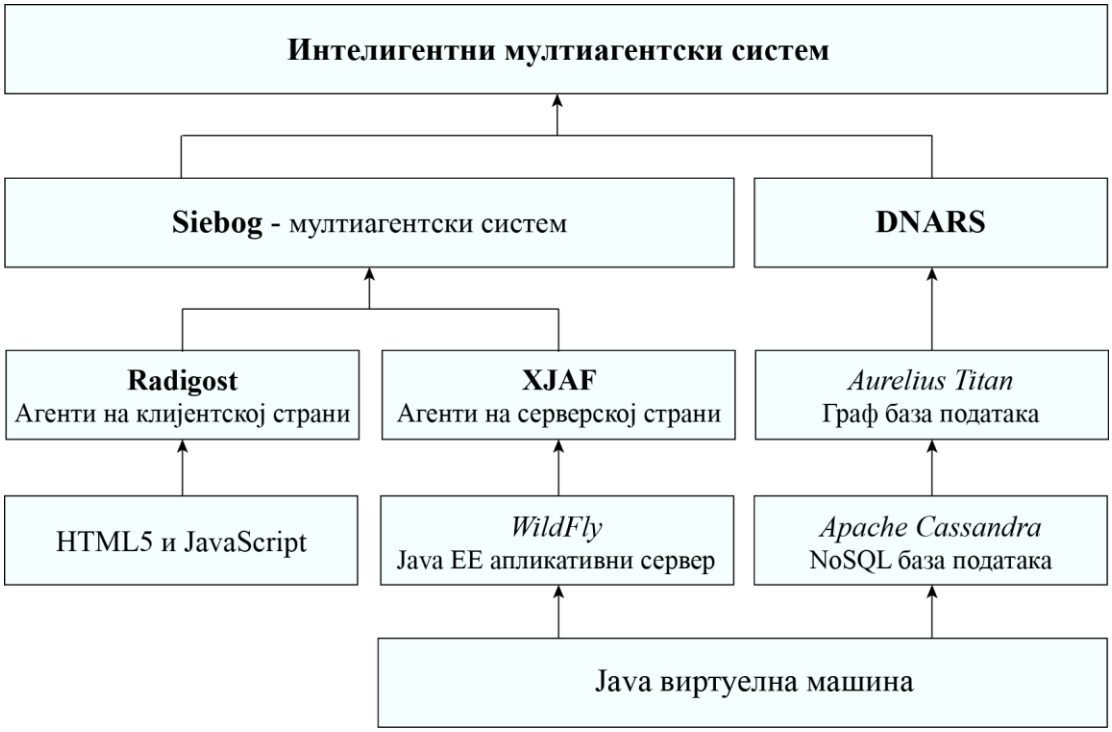
4.3 Интелигентни мултиагентски систем

За разлику од већине мултиагентских система који користе BDI архитектуру за своје интелигентне агенте (Bordini i dr. 2009; Hoek i Wooldridge 2013; Hindriks 2017; Pokahr i dr. 2014; Weiss 2013; Wooldridge 2009), Siebog је добио подршку за расуђивање у виду дистрибуираног система за не-аксиоматско резоновање који је детаљније описан у поглављу 5 (Mitrović 2015). Главна предност DNARS-а у поређењу са постојећим когнитивним и резонујућим архитектурама је примена најсавременијих техника за управљање и обраду великих база података. Овај

приступ омогућује DNARS-у да ради са великим базама знања и даје одговоре у реалном времену великом броју клијената.

Приликом интеграције серверског дела Siebog-а и DNARS-а омогућено је коришћење анотација као једне од основних техника мета-програмирања у Java EE платформама. То значи да програмери применом анотација могу нпр. истаћи циљеве агената, при чему систем у позадини аутоматски преводи сигнале које шаље DNARS-ов руководилац догађајима (енг. Event manager) у позиве одговарајућих метода.

Након спајања XJAF и Radigost компоненти у јединствен Siebog мултиагентски систем и додавањем логике за не-аксиоматско резоновање добијен је нови, интелигентни мултиагентски систем. Структура овог система је приказана на слици 4.4.



Слика 4.4. Структура новог, интелигентног мултиагентског система

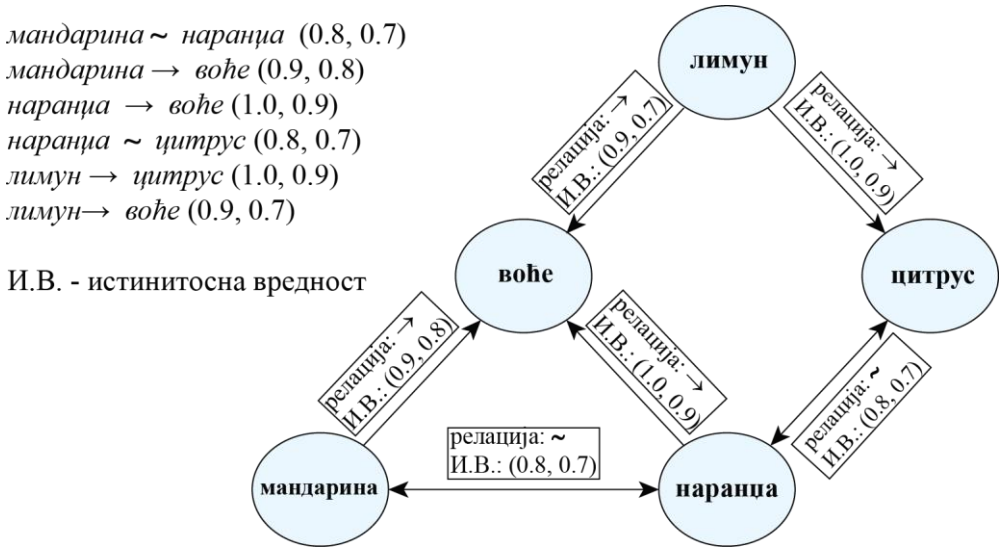
ALAS је проширен да подржи све захтеве новог система. Да би се дизајнирао што ефикаснији програмски језик који ће омогућити развој агената из уско специјализованих домена, као и интелигентних агената способних да доносе закључке на основу одговарајућих база знања, коришћене су предности DSL и AOPL наведених у поглављу 2.

5 DNARS

DNARS уведен у (Mitrović 2015) је нова архитектура за резоновање која се може користити за развој интелигентних агената. Ова архитектура се појавила као подршка Siebog окружењу уводећи агенте са когнитивним способностима. Основни циљ ових агената је да доносе одлуке на основу великих база знања при чему неки делови база знања често могу бити неконзистентни или недовољни за доношење закључака. Да би закључци били што ефикаснији, интелигенција DNARS-а је заснована на претходно описаној NAL логици за резоновање која веома успешно решава проблеме неконзистентности и недовољног знања (Mitrović 2015; Wang i Awan 2011; Wang 2006, 2013).

Потреба за обрадом великих количина података захтева примену модерних стандарда и технологија. С обзиром да подаци из великих база знања које користи DNARS углавном немају статичку структуру, број корисника се мери у милионима и велика динамичност условили су DNARS да за обраду ових података користи систем за управљање нерелациним базама података – NoSQL и базу података у облику графа (Robinson, Webber i Eifrem 2013; Sadalage i Fowler 2012; Tiwari 2011). DNARS је заснован на *Titan* (2015) дистрибуираној граф бази података која функционише у оквиру нерелационе Apache Cassandra базе података[12]. База знања заснована на NAL реченицама је представљена тзв. усмереним графом особина (Robinson, Webber i Eifrem 2013) кога чине чворови, везе између чворова (гране) и својства чворова и грана. У DNARS-у, атомски и сложени термови NAL реченица су представљени чворовима, релације наслеђивања и сличности одговарајућим гранама а истинитосне вредности реченица представљају особине (својства) закачене на ове реченице графа особина. Слика 5.1 приказује неколико NAL реченица представљених графом особина.

[12] http://cassandra.apache.org



Слика 5.1. Скуп произвољних NAL реченица и одговарајући граф особина

Процес извођења у DNARS-у је реализован скупом алгоритама за обилазак и приступ елементима графа особина. За ову сврху *Titan* граф користи домен-оријентисани експресивни *Gremlin* (2019) упитни језик који је оријентисан према путањама за комплексно обилажење графа особина и помоћу њега се може приступити елементима графа, анализирати и модификовати граф.

DNARS укључује концепте и правила извођења прва четири нивоа NAL логике која су описана у поглављу 3:

- Релације наслеђивања и сличности,
- Дедукција, индукција, абдукција, поређење, аналогија и наликовање,
- Избор и ревизија и
- Релациони термови.

Дизајниран као самосталан систем, DNARS пружа једноставан интерфејс који могу користити разни спољни клијенти (људи, софтверски агенти или неки слични интелигентни системи). Међутим, његова основна сврха је да пружа подршку за интелигентне агенте (да им омогући да самостално доносе одлуке и закључке) у Siebog мултиагентском окружењу.

Две основне компоненте DNARS-а описане у (Mitrović 2015) су DNARS-ов механизам извођења и позадинска база знања.

5.1 DNARS-ов механизам извођења

DNARS-ов механизам за извођење садржи две компоненте:

- Механизам за извођење унапред (енг. Forward inference engine) који изводи нове закључке и

– Механизам одлучивања (енг. Resolution engine) који одговара на питања клијената.

Механизам за извођење унапред обезбеђује директну имплементацију правила извођења унапред дефинисаних у NAL. Циклус закључивања могу покренути спољни клијенти или неки интерни процеси.
Механизам одлучивања даје одговоре на два типа питања:

– Питања која садрже знак '*?*' и имају следећи формат, '*C релација ?*' или '*? релација П*', где *C* представља субјекат а *П* предикат NAL реченице, док *релација* може бити или релација наслеђивања (–>) или релација сличности (~). У овом случају, механизам одлучивања проверава базу знања и изводи најбољи могући одговор за '*?*'.
– Питање у форми '*C релација П*'. Одговор на ово питање је веровање у форми '*C релација П* (*f*, *c*)', где (*f*, *c*) представља истинитосну вредност одговора (закључка). У случају да одговор није директно доступан у бази знања система, механизам даје одговор на основу правила *извођења уназад*.

Систем може имати више одговора на исто питање. Најбољи могући одговор се добија рачунањем формуле описане у дефиницији 5.1.

Дефиниција 5.1. *Очекивање* учесталости (енг. *expectation*) - '*e*' означава која је вероватноћа да ће се вредност учесталости поновити у блиској будућности. Вредност *очекивања* је у опсегу [0, 1] и рачуна се по формули: $e = \left(f - \frac{1}{2}\right)c + \frac{1}{2}$ (Wang 2006, 2013).

5.2 Позадинска база знања

Позадинска база знања представља целокупно знање и искуство система. Да би се испунили кључни захтеви вештачке интелигенције, укључујући и обраду великих количина података у реалном времену, DNARS посебну пажњу посвећује организацији базе знања која је дизајнирана у виду дистрибуиране, скалабилне архитектуре која се састоји из три нивоа:

– *Хоризонтално скалирање* (енг. horizontal scaling): како се количина знања повећава, перформансе система се одржавају једноставним додавањем нових рачунара у кластер. Један рачунар је означен као главни а сви остали као споредни. Омогућена је отпорност система на грешке, јер се подаци копирају на остале рачунаре кластера. У случају квара једног рачунара систем успешно наставља да функционише.
– *Области знања* (енг. knowledge domain): постоји могућност поделе целокупног знања на више области (домена). Током извршавања, систем може консултовати једну или више области. Конкретно, знање које

припада једном клијенту може бити смештено у једну област при чему више клијената може радити са истом облашћу и тиме делити знање и стечено искуство.

– *Краткотрајна меморија* (енг. short-term memory): садржи знање које је директно доступно правилима за извођење, те представља полазну основу за расуђивање у DNARS-у. Главни задатак краткотрајне меморије је да служи као модул за оптимизацију извршавања. Њен садржај би требало у потпуности да стане у радну меморију једне машине, пружајући максималне перформансе. Када се скуп релевантних циклуса расуђивања заврши, садржај краткотрајне меморије се уписује у одговарајућу област знања.

За клијенте је веома важно да буду обавештени о променама у бази знања, нарочито ако више клијената дели исту *област знања*. На пример, клијент може пожелети да предузме низ акција као одговор на новостечено знање. Да би подржао ове функционалности, DNARS укључује *Руководиоца догађајима*. Промена у *области знања* може генерисати један или више догађаја, који ће затим бити прикупљени од стране *Руководиоца* и послати одговарајућим клијентима.

6 Опис коришћених технологија

Првобитна верзија ALAS језика и пратећи скуп алата су примарно били усмерени на развој агената за различите имплементације SOM окружења (Mitrović, Ivanović i Vidaković 2011; Mitrović i dr. 2012). У почетку је ALAS развијан применом JavaCC парсер генератора (Javacc 2017). Развој нових технологија довео је до појаве нових, ефикаснијих алата за дизајн и развој програмских и домен-оријентисаних језика. Због појединих ограничења код JavaCC парсер генератора али пре свега због све сложенијих захтева при развоју ALAS-а, прешло се на нови оквир за развој програмских и домен-оријентисаних језика – Xtext.

6.1 Xtext

Појава Xtext-а, могло би се рећи, довела је до праве револуције када је развој домен-оријентисаних језика у питању. До пре 10-ак година, пре појаве Xtext-а развијен је релативно мали број ефикасних и применљивих домен-оријентисаних језика јер је развој једног DSL-а захтевао пуно труда и експертско познавање одговарајућих технологија. Xtext је бесплатан оквир отвореног кода, тј. алат за развој екстерних текстуалних DSL-ова и настао је као део oAW (енг. openArchitectureWare) пројекта (Efftinge i Völter 2006).

Применом Xtext-а веома је једноставно креирати наменски језик писањем текстуалне граматике засноване на EBNF нотацији. На основу граматике, Xtext аутоматски креира парсер генератор заснован на ANTLR[13] парсер генератору, оквир за моделовање (енг. Eclipse modeling framework - EMF), стабло апстрактне синтаксе (мета-модел) као и потпуно прилагођен текст едитор као додатак за Eclipse окружење. Развој са Xtext-ом је је оптимизован за мале измене, тако да се додавање нових особина постојећем DSL-у може обавити веома брзо. Новије верзије омогућују имплементацију софистицираних програмских језика и нуде

[13] http://www.antlr.org

доста могућности (слично неким ефикасним језицима опште намене) као што су валидација кода (енг. code validation), бојење синтаксе (енг. syntax highlighting), поље за приказ структуре пројекта (енг. outline), навигација, допуна кода (енг. code completion), генерисање кода (енг. code generation) итд. Новија верзија Xtext-а, поред текст едитора за Eclipse омогућује генерисање текст едитора за IntelliJ IDEA као и веб едитора.

За развој ALAS-а у почетној фази је коришћен управо Xtext пошто је подржавао све потребне захтеве. Пошто је идеја била да ALAS буде 'Java-like' језик, то је био један од кључних разлога зашто је баш Xtext коришћен, с обзиром да је заснован на Javi и да од свих постојећих окружења пружа најбољу подршку када је у питању развој DSL који имају 'Java-like' синтаксу.

Када је развој ALAS-а у питању, првобитни циљ је био да се ALAS агенти извршавају у Java окружењима као што су XJAF, JADE итд. Због тога је било потребно конвертовати ALAS код агента у Java програмски језик како би се агенти могли извршавати на платформама имплементираним у Javi и да би се несметано могли кретати између тих платформи.

За генерисање Java кода на основу модела у оквиру Xtext-а, користи се Xtend језик[14]. Xtend је концизан и флексибилан дијалект Jave, који се компајлира у читљив Java изворни код. Омогућује коришћење било које постојеће Java библиотеке без проблема. Због своје уске повезаности са Javom, применом Xtend-а је врло једноставно развити конвертер ALAS језика у Javu.

У оквиру Xtext алата постоје још два језика настала на основу језика Xtend. То су Check, који се користи за спецификацију ограничења (креирање алата за проверу валидности кода) и Xpand који се користи за креирање шаблона за конверзију постојећег кода.

Упоредо са развојем и побољшањем Siebog платформе и ALAS језик се надограђивао добијајући нове могућности. Пре свега, Siebog је добио подршку за извршавање својих агената у веб окружењима где су агенти најчешће имплементирани у JavaScript језику. Поред тога, значајно побољшање је интеграција Siebog-а са дистрибуираним системом за не-аксиоматско резоновање. Стога, нова верзија Siebog-а омогућује развој интелигентних агената заснованих на не-аксиоматској логици који се могу извршавати како на серверској страни тако и на клијентској. Серверска страна Siebog-а је имплементирана у Javi а клијентска у JavaScript језику. Да би агенти могли да се извршавају и на клијентској и на серверској страни морају бити способни да се трансформишу у језик одредишне платформе. Управо је ово био главни мотив за настанак ALAS језика. Применом Xtext-а веома је једноставно конвертоавати ALAS код агента у Java изворни код али трансформација ALAS-а, али и других домен-оријентисаних језика у неки не-Java језик применом Xtext-а је често веома

[14] https://www.eclipse.org/xtend

сложен процес који захтева доста труда. Након потребе за конверзијом ALAS-а у JavaScript и нових побољшања Siebog-а у виду повезивања са разним другим мултиагетнским окружењима имплементираним у различитим програмским језицима јасно је да ће у будућности ALAS морати да има могућност да се, поред Java и JavaScript језика, конвертује и у неке друге језике.

Због комплексног процеса генерисања не-Java кода применом Xtext-а дошло је до потребе да се ALAS развија применом неког другог, једноставнијег алата. Један од алата који у почетку није имао велике могућности као Xtext (пре свега у виду генерисања текст едитора и других облика подршке за Eclipse окружење) је textX (Dejanović i dr. 2016; Dejanović 2019b) који је коришћен за развој последње верзије ALAS језика која укључује подршку за не-аксиоматско резоновање.

6.2 textX

textX је мета-језик (тј. језик за дефиницију језика) за спецификацију домен-оријентисаних језика у Python-у а развијен је на Катедри за информатику Факултета техничких наука у Новом Саду. textX има сличну нотацију и обезбеђује скоро исте могућности као и Xtext тако да није било потребно уложити пуно труда како би се ALAS редизајнирао и прилагодио textX-у. За разлику од Xtext-а, textX је једноставнији али и знатно флексибилнији и испуњава све захтеве ALAS-а.

На основу граматике textX динамички креира мета-модел (у форми Python класа) и парсер за дефинисани језик. Парсер анализира изразе у језику и аутоматски генерише граф Python објеката (тј. модела) који одговара мета-моделу. textX нуди додатне карактеристике:

- аутоматско повезивање – инстанца неког објекта класе ће се аутоматски референцирати на одговарајући Python објекат
- модуларизација граматике – граматика се може поделити на више датотека које се затим могу позвати (укључити) у оквиру других модула
- визуализација мета-модела и модела - и мета-модел и модел могу се визуализовати применом програмског пакета GraphViz (2018).
- Модификатори понављања израза – применом ових модификатора може се на једноставнији и краћи начин дефинисати неко правило које подразумева понављање одговарајућих елемената
- Модификатори правила – применом одговарајућих модификатора може се изменити парсер на нивоу правила
- Осетљивост на велика и мала слова – textX садржи подесиви параметар који омогућује парсеру да ли да разликује велика и мала слова

- Руковање празним местима – слично као и код претходне карактеристике, textX нуди могућност подешавања параметра који ће указати парсеру да ли да узима у обзир празна места (празно место, табулатор и нова линија)
- Подршка за коментаре
- Накнадно процесирање модела објеката - textX нуди накнадно процесирање модела објеката регистровањем Python позива који ће примити објекат онакав какав је конструисан. Пост-обрада се користи за разне ствари, од семантичке валидације модела до аугментације модела.
- Детаљан извештај о грешкама (енг. debuging) – применом одговарајућег параметра може се омогућити/онемогућити генерисање темељног извештаја о грешкама.

textX се константно развија пружајући нове могућности. Новије верзије textX-а нуде скоро исте могућности као и Xtext пре свега у погледу спецификације текст едитора са подршком за textX. У наставку су набројана нека побољшања која нуде новије верзије textX-а:

- развијен је додатак за Visual Studio Code који пружа подршку за развој textX текстуалних DSL-ова, у виду бојења синтаксе, приказа структуре кода
- додата је и подршка за Emacs едитор (GitHub 2016a)
- развијен је додатак viewX за Visual Studio Code који омогућује визуализацију кода у виду графова (GitHub 2017)
- подршка у оквиру Vim едитора у виду бојења синтаксе, таг бар подршке, TODO листе (GitHub 2016b)
- развијен textX језички сервер применом протокола језичког сервера (енг. Language Server Protocol – LSP) (GitHub 2018). LSP је протокол који се користи између едитора и језичког сервера обезбеђујући напредну подршку за развој језика у одговарајућем едитору. Применом овог протокола омогућене су следеће карактеристике: валидација кода (енг. linting), пребацивање на дефиницију (енг. go to definition), проналажење свих референци (енг. find all references), допуна кода, приказ структуре кода, генерисање *.dot* фајлова модела и мета-модела
- развијен је textX клијент, тј. додатак за Visual Studio Code који користи textX језички сервер.
- textX додатак за Ninja IDE који омогућује бојење синтаксе и визуализацију кода

Поред свих напредних могућности које textX пружа за развој DSL-ова, textX нуди и могућност развоја језика без икаквих додатака, при чему је довољно користити обичан текст едитор за писање граматике, генератора и текстуалног модела а цео процес компајлирања и генерисања се може извести коришћењем командне линије.

textX зависи само од Arpeggio (Dejanović, Milosavljević i Vaderna 2016; Dejanović 2019a) парсера. Arpeggio је рекурзивни силазни парсер са памћењем међурезултата – мемоизацијом (енг. memoization) (Dejanović, Perišić i Milosavljević 2010). Као и textX, развијен је на катедри за информатику на Факултету техничких наука у Новом Саду. Граматика језика се може креирати у виду Python исказа или применом PEG (енг. parsing expression grammar – PEG) формализма за писање граматика језика. Arpeggio нуди неке од следећих особина:

- неограничен '*поглед унапред*' (енг. *lookahead*),
- линеарно време парсирања,
- не дозвољава вишезначност, за разлику од контекстно слободних граматика (енг. context-free grammar - CFG) (приликом парсирања постоји само једно могуће стабло парсирања),
- ради као интерпретер (не генерише се програмски код за написану граматику јер се граматика интерпретира динамички за време парсирања)

Као и textX, имплементиран је у Python-у. Arpeggio динамички креира парсер из описа граматике при чему није потребно генерисати програмски код. Омогућена је семантичка анализа применом семантичких акција које се имплементирају као Python класе. Arpeggio је намењен интегрисаним окружењима за развој домен-оријентисаних језика у којима се дефиниција синтаксе мења релативно често.

6.3 Jinja2

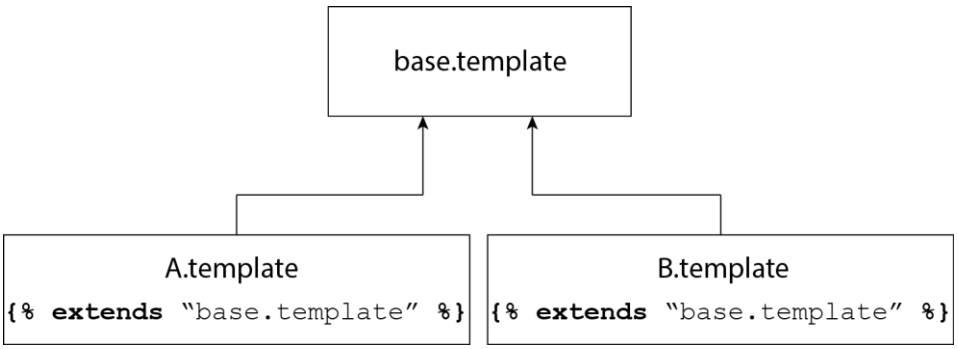
За обраду модела језика и генерисање потребних изворних кодова, textX користи Jinja2 шаблон (Jinja2 2018) пошто је он развијен такође у Python-у. С обзиром на ту чињеницу, у шаблону је релативно једноставно користити постојеће textX класе и објекте преведене у Python приликом компајлирања.

Jinja2 је модеран, прилагодљив, али и једноставан дизајнерски језик са широком применом. Настао је по узору на Django шаблоне[15]. То је текстуални, шаблонски језик опште намене са могућношћу извршавања шаблона у sandbox окружењу. Jinja2 је библиотека за Python која је дизајнирана да буде флексибилна, брза и безбедна. За разлику од Django шаблона, Jinja2 омогућује коришћење Python-like израза.

Шаблони могу у потпуности бити дизајнирани као string компоненете или се могу учитати из датотеке. Jinja2 подржава наслеђивање тако да се може дизајнирати универзални шаблон који може бити наслеђен од стране других

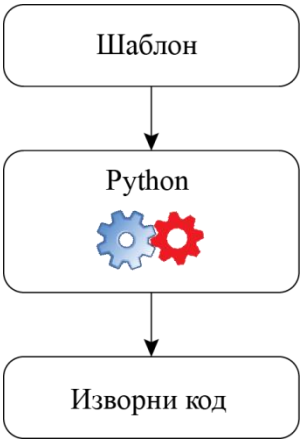
[15] https://www.djangoproject.com

шаблона као што је приказано на слици 6.1 где шаблони *A* и *B* наслеђују раније дефинисан шаблон *base*.



Слика 6.1. Jinja2 наслеђивање

Слика 6.2 приказује принцип генерисања изворног кода применом Jinja2 шаблона и одговарајућег Python кода.



Слика 6.2. Принцип генерисања изворног кода

Jinja2 подржава промењљиве, блокове, филтере (са могућношћу прослеђивања аргумената у виду објеката), цеви (енг. pipes), изразе и тагове који специфицирају функционалну логику шаблона, наредбе за контролу тока као што су *if / elif / else*, *while*, *do-while* и *for* петље, при чему нуди и велики број уграђених функција.

Jinja2 шаблон који је коришћен у развоју ALAS језика је детаљније описан у поглављу 7.4.

6.4 Eclipse

За симулацију рада целог система коришћен је Eclipse. Eclipse је програмско развојно окружење развијено у Javi и највише је намењено Java корисницима али пружа веома добру подршку и за друге програмске језике. Основна предност

Eclipse окружења је што функционише на бази додатака при чему се додаци могу укључити у пројекат само ако су потребни, не морају бити инсталирани приликом инталације Eclipse окружења што значајно доприноси уштеди времена и меморијског простора.

Eclipse подржава развој за разне сервере, JBoss, GlasFish, Apache Tomcat и друге. За Siebog мултиагентски систем коришћен је JBoss апликациони сервер, иначе најкоришћенији Java EE сертификовани апликациони сервер на тржишту. Користи се за развој, тестирање и инсталацију сложених веб апликација, пословних апликација и сервиса базираних на сервисно-оријентисаној архитектури.

Дистрибуирани систем за не-аксиоматско резоновање који није имплементиран у Javi такође је развијен применом Eclipse окружења. Због специфичности и комплексности не-аксиоматске логике, за развој је знатно флексибилнији Scala језик тако да је имплементација за DNARS урађена у Scala програмском језику применом Eclipse додатка за Scala језик.

То је функционални програмски језик чија популарност расте уназад пар година и све се више почиње користити у индустрији, а не само у академским круговима. Неке од компанија које користе Scalu су LinkedIn, Twitter, Netflix, Foursquare, The Guardian, Sony и друге, што довољно говори да то више није неприменљив језик који користе студенти и истраживачи на факултетима, него и велике мултинационалне компаније како би послуживале милионе корисника својим услугама.

6.5 Titan граф база података

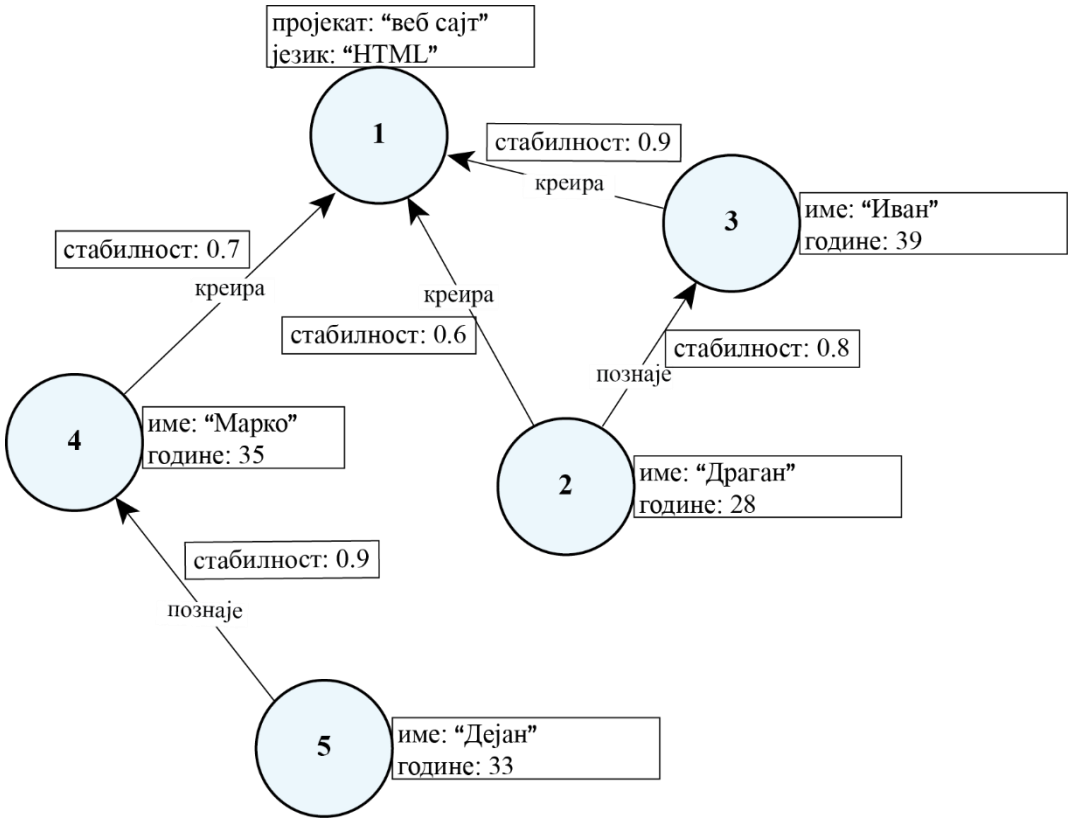
Titan (2015) база података је коришћена у оквиру имплементације DNARS-а као граф база података за складиштење позадинске DNARS базе знања тј. структурираних NAL реченица.

Titan дистрибуирана граф база података функционише у оквиру Apache Cassandra[12] нерелационе (NoSQL), базе података. Apache Cassandra је прави избор када је потребна скалабилност и висока доступност без угрожавања перформанси. Линеарна скалабилност и обезбеђена толеранција на грешке на хардверу или *облаку* (енг. cloud) чине је савршеном платформом за критичне податке. Од свих база података у том рангу, Cassandra пружа најбољу подршку за репликацију тј. расподелу оптерећења по умреженим чворовима што доприноси бржем одзиву система корисницима а такође и сигурност у случају неког квара (прекида).

Titan је скалабилна граф база података оптимизована за складиштење и испитивање графова који садрже стотине милијарди чворова и грана распоређених по кластеру са великим бројем чворова. То је трансакциона база података која може подржати стотине хиљада истовремених корисника који извршавају сложене обиласке графова у реалном времену.

Граф базе података користе граф структуре са чворовима (који представљају ентитете), гранама (које повезују чворове) и својствима (представљају атрибуте) за репрезентацију и чување података. По дефиницији граф база је сваки систем код којег сваки елемент директно показује на суседни елемент при чему није потребан никакав преглед индекса. Захваљујући способности чувања и управљања великом количином једноставних, али и сложених података, ова категорија NoSQL база података се употребљава на познатим друштвеним мрежама попут Facebooka, Twittera или претраживачима попут Google-a. Граф базе података на важности добијају и због развоја концепта Интернета ствари (енг. Internet of Things - IoT) јер он у својој у дефиницији укључује повезивање уређаја, а тиме и података, што чини основу граф база података (Savić 2018; Šestak 2014).

У основи DNARS-a је тзв. усмерени граф особина (енг. Property graph) који уједно представља један од најчешће примењиваних облика за представљање нерелационих граф база података. На слици 6.3 је приказан један пример усмереног графа особина са одговарајућим чворовима, гранама и особинама.



Слика 6.3. Модел усмереног графа особина

За обилазак графа и приступ елементима и атрибутима, DNARS користи Gremlin упитни језик заснован на Scali (Gremlin 2019). Gremlin је експресивни језик који има конструкте засноване на повезаним операцијама за ефикасан обилазак графа. Оријентисан је према путањама за комплексно обилажење графа,

а помоћу њега се могу преузимати, анализирати и модификовати подаци у графу. Омогућује извршавање комплексних функционалности које се називају кораци (енг. steps), а који су придружени обиласку графа.

Упит написан у Gremlin језику је ланац операција које се евалуирају с лева на десно, чиме се ствара путања за обилазак графа, на чему се и заснива овај упитни језик. Путања се креира спајањем операција оператором '**.**' тако да је свака операција корак обиласка у којем се обрађује објекат добијен у операцији која је претходила тренутном кораку. Могуће је дефинисати неограничен број оваквих корака, док се променом редоследа дефинисаних операција мења семантика упита. За писање Gremlin упита потребно је најпре позвати функцију која ће креирати граф одређеног типа (Šestak 2016).

Операције/функције у Gremlin упитном језику су подељене у следеће категорије:

- Функције за трансформисање
- Функције за филтрирање
- Функције за гранање
- Функције за споредни учинак (енг. Side effect)
- Методе

7 ALAS – дизајн и имплементација

ALAS је агентски, домен-оријентисани језик високог нивоа апстракције за писање софтверских и интелигентних агената који се извршавају у оквиру интелигентног мултиагентског система Siebog (Mitrović i dr. 2016a; Mitrović i dr. 2016b). Неке од најважнијих особина Siebog-а као што су аутоматско распоређивање оптерећења по чворовима кластера, хардверска и софтверска отпорност на грешке, дељење кода, хетерогена мобилност агената, вишеплатформска размена порука, заснованост агената на вештачкој интелигенцији у виду не-аксиоматског резоновања итд., затевају скуп техника које постојећи агентски и домен-оријентисани језици не могу да обезбеде. То је довело до потребе за развојем новог агентског језика који може да подржи све ове захтеве.

Циљ је био омогућити развој интелигентних агената који се могу извршавати на платформама које су имплементиране у различитим програмским језицима. Управо је *hot compilation* главна карактеристика ALAS-а. То подразумева да се приликом пристизања агента у неки мултиагентски систем, који је имплементиран у неком програмском језику X, његов ALAS изворни код аутоматски, у *лету* (енг. on-the-fly) преведе у X изворни код. Овај код се даље прослеђује матичном компајлеру који производи извршни код након чега агент може да настави са извршавањем свог задатка. Ово омогућује да се ALAS агенти успешно могу извршавати у разним окружењима имплементираним нпр. у Java, JavaScript, Python и другим језицима. Цео процес трансформације ALAS изворног кода је детаљно описан у поглављу 7.4.

Заштита од злонамерних напада је важно питање, посебно у системима који користе мобилност агената. Да би заштитили код агента током процеса миграције, могу се користити сертификати кода (Vidaković, Sladić i Konjović 2003). Сертификат садржи хеш (енг. hash) изворног кода ALAS-а, као и дигитални запис интерног стања агента. Сигурносна провера се изводи пре процеса парсирања, а ако не успе, агент се одбацује.

Програмски конструкти за контролу тока, као што су `if – elseif – else` наредбе и `for`, `while` и `do – while` петље су такође подржани. Њихова синтакса је заснована на синтакси програмског језика Java.

Као што је раније наведено, један од основних проблема Siebog окружења био је проблем интероперабилности тј. хетерогене мобилности агената и немогућност међусобне комуникације и извршавања агената на разним платформама уколико су оне имплементиране у различитим програмским језицима. Поред других захтева Siebog-а, то је био главни разлог за развој универзалног, агентског и домен-оријентисаног језика.

ALAS је DSL подржан елементима вештачке интелигенције. Siebog не користи традиционалну BDI архитектуру. Уместо тога, Siebog мултиагентска платформа је проширена тако да укључује подршку за расуђивање засновану на DNARS-у који укључује агенте са иновативним вештинама расуђивања. Интеграција клијентске стране Siebog-а са DNARS-ом је имплементирана на исти начин као и интеграција са серверском страном, тј. кроз комуникацију са одговарајућим веб сервисима. На овај начин, Siebog и DNARS представљају јединствени мултиагентски оквир за развој и извршавање интелигентних агената у веб окружењима, нудећи нове и занимљиве могућности за практичну примену агентске технологије.

Пошто су анотације стандардни мета-програмски конструкти Java EE, исти приступ се може применити за дефинисање DNARS агената на страни сервера. Програмерима је омогућено коришћење анотација за означавање циљева агената. Развој интелигентних агената заснован на анотацијама се користи и у неким другим агентским системима. Тренутно, Siebog агенти који су засновани анотацијама и DNARS расуђивању користе неколико анотација:

- анотације које се односе на веровања (енг. beliefs)
- анотација која се односи на акције (енг. actions) а која служи за дефинисање одговарајућих предуслова које агент треба да обезбеди приликом извршавања неке акције
- анотација која се односи на домен функционисања агента

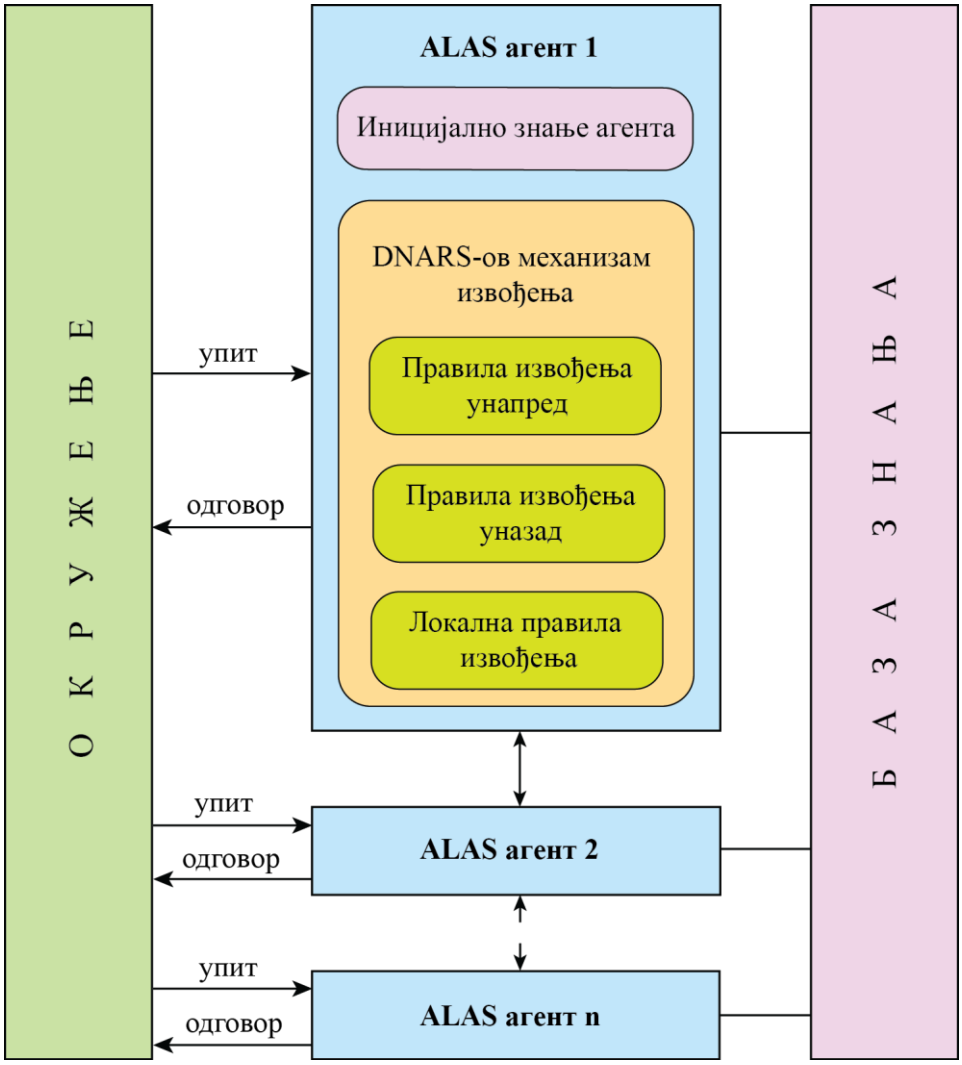
Основне анотације за управљање веровањима су `@Beliefs`, `@BeliefAdded` и `@BeliefUpdated`. За дефинисање предуслава у оквиру акција користи се `@PreCondition` анотација. Анотација `@Domain` се користи за означавање домена тј. дефинисање базе знања коју ће агент да користи.

На основу анотација систем у позадини аутоматски преводи сигнале упућене од DNARS-овог менаџера догађаја (енг. *event manager*), у позиве одговарајућих метода.

Други важан циљ ALAS-а је да се његови агенти могу програмирати применом одговарајућих конструката специфичних за домене у којима се агенти извршавају. Главна сврха ових конструката је да сакрију укупну комплексност домена и на тај начин омогуће програмерима да се више фокусирају на решавање конкретног проблема. Приликом позива неког конструкта, у позадини ће се извршавати скуп свих функционалности које покрива тај конструкт при чему програмер не мора да зна шта се све извршава у оквиру тог конструкта. Често се приликом развоја агената користи помоћ експерата из одговарајућих домена

примене. Да би се то избегло, приликом развоја DSL и његових конструката, препоручљиво је користити помоћ експерта за одговарајући домен за који се DSL развија како би се развили што ефикаснији конструкти које ће програмери касније моћи самостално, без помоћи експерта да користе како би решили комплексне задатке.

На слици 7.1 приказана је апстрактна архитектура ALAS-а.



Слика 7.1. Апстрактна архитектура ALAS језика за подршку DNARS-у

7.1 Граматика ALAS језика

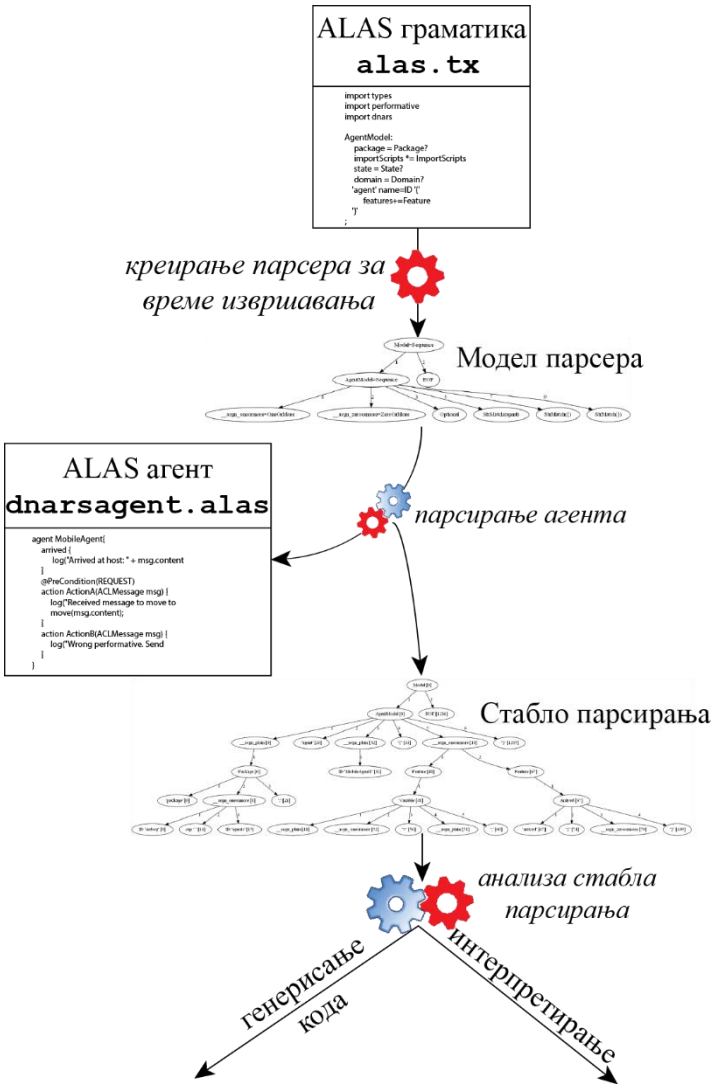
Као стандардна средства за опис синтаксе програмских језика користе се синтаксни дијаграми и мета-језици. Синтаксни дијаграм представља графички приказ поступка формирања синтаксно правилних конструкција програмског језика. Други начин за опис синтаксе програмских језика су мета језици. Мета-језик је посебно дефинисан језик са скупом простих правила помоћу којих се

дефинишу синтаксно правилне конструкције програмског језика који се описује. textX, алат који је коришћен за развој ALAS-а, је мета-језик.

Синтакса ALAS језика је дефинисана применом граматике написане у textX-у која садржи скуп одговарајућих правила. Неки оквири за развој DSL-ова нуде одређени број уграђених правила која се могу користити у оквиру дефинисања нових правила. textX нуди неколико уграђених правила која представљају основне типове: `STRING`, `NUMBER`, `INT`, `FLOAT`, `BOOL` и `ID` (Dejanović 2019b). textX нуди могућност дефинисања граматике језика применом Python израза или путем PEG нотације. Због комплексности граматике ALAS језика, лакше је било дефинисати граматику применом PEG нотације.

PEG нотација је као и већина других, заснована на проширеној Бакус-Науровој форми (енг. Extended Backus-Naur form - EBNF) (EBNF; Fowler 2010). EBNF додаје одређене синтаксне изразе BNF (Backus-Naur form) нотацији и омогућује једноставнији запис правила граматике. EBNF нотација се састоји од коначног броја реченица које представљају синтаксна правила. Њиме се дефинише облик синтаксно правилних конструкција програмског језика. Једна од основних предности PEG граматика у поређењу са другим типовима граматика је уређен избор правила или кључних речи што омогућује једнозначно парсирање - постоји само једно синтаксно стабло и граматика не може бити двозначна.

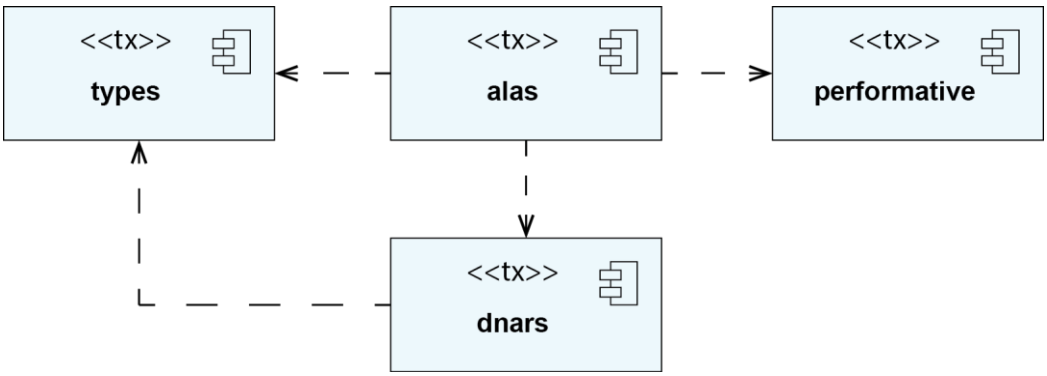
Основна технологија за парсирање, која је саставни део textX-а је Arpeggio парсер који је заснован на PEG формализму за описивање структуре језика па је и из тог разлога у основи граматике ALAS језика PEG конвенвенција за записивање правила граматике језика. Принцип рада Arpeggio PEG парсера на примеру ALAS језика приказан је на слици 7.2.



Слика 7.2. Принцип рада Arpeggio парсера

Једна од предности textX-а је модуларизација граматике што омогућује једноставнију спецификацију и лакше разумевање саме граматике. У случају ALAS-а, различити сегменти граматике су раздвојени у неколико фајлова (модула) који заједно чине граматику ALAS језика, при чему се сви фајлови граматике завршавају екстензијом *.tx*. Слика 7.3 приказује хијерархију фајлова граматике ALAS језика. Да би граматика сепцифицирана у једном фајлу имала приступ елементима (правилима) из граматике специфицирене у другом фајлу неопходно је укључити другу граматику што се шостиже применом кључне речи *import*. Након ове кључне речи следи име фајла у коме је дефинисано одговарајуће правило. На пример, у фајлу *alas.tx* се позивају правила из неколико других фајлова: *dnars.tx*, *performative.tx* и *types.tx* па ће они на почетку фајла *alas.tx* бити укључени помоћу кључне речи *import*, с тим што приликом укључивања није потребно наводити екстензију фајлова. У случају ALAS граматике у оквиру *alas.tx* на почетку је потребно написати:

```
import types
import performative
import dnars
```



Слика 7.3. Дијаграм компоненти хијерархијске организације фајлова граматике ALAS језика

У првој фази развоја ALAS-а дефинисани су општи, не-агентски програмски конструкти за контролу тока као што су *if* и *if-else* наредбе, *while*, *do-while* и *for* петље, промењљиве итд. У последњој верзији језика, ове компоненте су побољшане и описане су у наставку.

Кључна компонента ALAS језика је агент. Део граматике који садржи правила помоћу којих се дефинише агент приказан је у листингу 1.

Листинг 1. Правила граматике језика за дефинисање агента

```
1. AgentModel:
2.     package = Package?
3.     importScripts *= ImportScripts
4.     state = State?
5.     domain = Domain?
6.     'agent' name=ID '{'
7.     features+=Feature
8.     '}'
9. ;
10. Package:
11.     'package' pack += ID['.'] ';'
12. ;
13. ImportScripts:
14.     'importScripts' '(' script=STRING ')' ';'
15. ;
16. State:
17.     'stateful' | 'stateless'
18. ;
19. Domain:
20.     'domain' '(' name=ID ')'
21. ;
22. Feature:
23.     AgentStates
24.     | Variable
```

```
25.      | (Assignment ';')
26.      | Function
27.      | Action
28.      | Init
29.      | Arrived
30.      | DnarsBeliefs
31.      | DnarsAddedUpdated
32.      | DnarsQuestion
33.      | DnarsInference
34. ;
35. Comment:
36.      /\/\/.*$/
37. ;
```

Као и сваки Java програм, ALAS агент може припадати неком пакету, а то омогућује правило `Package`. Име пакета се наводи након кључне речи `package` при чему навођење пакета није обавезно.

У оквиру textX граматике, применом кардиналитета означава се да ли неки елемент мора да се наведе и ако мора, наводи се колико пута се може поновити. textX подржава неколико типова кардиналитета:

- једно или више понављања (+=)
- нула или више понављања (*= или само *)
- елемент може да се наведе једном или ни једном (?= или само ?).

Пошто је, поред Java окружења омогућено да се ALAS агенти извршавају и у JavaScript окружењима, у граматици су дефинисана поједина правила која ће омогућити дефинисање одговарајућих особина које су карактеристичне за JavaScript окружења. Једно од таквих правила је `ImportScripts` и служи за укључивање одговарајућих JavaScript фајлова у оквиру агента. Након кључне речи `importScripts` у заградама се наводи путања до одговарајуће датотеке.

Једна од кључних компоненти је стање агента које се у граматици дефинише применом правила `State`. Ово правило, у складу са Java граматиком, нуди две могућности за стање агента:

- `stateful` – постоји евиденција о целокупном стању агента са свим претходним интеракцијама
- `stateless` – не постоји евиденција о стању агента

Ако се не наведе ни један параметар подразумевана вредност за стање агента је `stateless`.

Приликом дефинисања агента опционо се може навести домен примене (знање које агент користи) коришћењем кључне речи `domain`. Ово је омогућено применом `Domain` правила у граматици језика. Ако овај конструкт није дефинисан, подразумевано име домена ће бити исто као и име агента. Дефинисање домена је важно ако више агената користи исто знање. У том случају, сваки агент мора имати исто име домена.

Након претходно наведених параметара наводи се име агента које се дефинише помоћу кључне речи `agent` (листинг 1, линија 6).

У граматици се може дефинисати правило које омогућује писање коментара. За писање коментара у ALAS језику користи се правило `Comment` чија се дефиниција може видети у листингу 1, линија 35.

Све особине тј. целокупна логика агента дефинисана је у оквиру правила `Feature` које се састоји од низа других правила (листинг 1, линија 22).

У листингу 2 се може видети пример дефиниције `stateful` агента чије је име `TestAgent` и који користи базу знања из домена `Test`. Агент припада пакету `siebog.agents` и укључена је скрипта `index.js`.

Листинг 2. Пример дефиниције ALAS агента

```
1. package siebog.agents;
2. importScripts("/siebog-war/index.js");
3.
4. stateful domain(Test) agent TestAgent {
5.    . . .
6. }
```

Основна сврха `AgentStates` конструкта је да сачува стање агента услед прекида тренутне активности (нпр. затварање претраживача у коме се агент извршава), или за време пребацивања агента на другу платформу.

Дефинисање промењљивих у агенту омогућује правило `Variable` (листинг 3, линија 1) Промењљиве се у агенту дефинишу потпуно исто као и у Javi. Прво се наводи тип промењљиве а затим, назив (може бити и више назива одвојених зарезом) и опционо, вредност (израз - `Expression`), која се додељује промењљивој. ALAS подржава девет типова података. Осам типова су преузети из скупа простих типова из програмског језика Java: `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` и `char`. Поред наведених типова подржан је и тип `String`.

Листинг 3. Део граматике ALAS језика

```
 1. Variable:
 2.   type=Type varnames+=ID[','] ( '=' exp = Expression )? ';'
 3. ;
 4. Assignment:
 5.   varname=ID op=Operator exp=Expression
 6. ;
 7. Operator:
 8.   '==' | '<=' | '>=' | '!=' | '=' | '<' | '>' | '!' | '&&' | '||'
 9. ;
10. Expression:
11.   op=Word (op=PlusOrMinus op=Word)*
12. ;
13. PlusOrMinus:
```

```
14.   '+' | '-'
15. ;
16. Word:
17.   op=Factor (op=MulOrDiv op=Factor)*
18. ;
19. MulOrDiv:
20.   '*' | '/'
21. ;
22. Factor:
23.   (op=PlusOrMinus)? op=Operand
24. ;
25. Operand:
26.   op=BaseType
27.   | op=CallFunction
28.   | op=ACLMessageProperty
29.   | ('(' op=Expression ')')
30. ;
31. ACLMessageProperty:
32.   aclmessage = ID '.' property = Properties
33. ;
34. Properties:
35.   'performative'
36.   | 'receivers'
37.   | 'sender'
38.   | 'content'
39.   | 'replyTo'
40. ;
41. CallFunction:
42.   func=[Function] '(' params*=FunctionParams[','] ')'
43. ;
44. FunctionParams:
45.   CallFunction | BaseType
46. ;
47. BaseType:
48.   (INT !'.') | FLOAT | BOOL | STRING | ID
49. ;
```

Додела вредности промењљивој врши се применом правила `Assignment`. Прво се наводи име промењљиве, затим оператор доделе и на крају вредност која се додељује (`Expression`). У граматици, у оквиру правила `Operator` дефинисано је десет различитих типова оператора доделе (листинг 3, линија 7).

Изрази у ALAS језику се дефинишу помоћу правила `Expression`. У изразу се може навести више различитих типова операнада повезаних основним аритметичким операцијама (+, -, *, ÷) које су дефинисане применом `PlusOrMinus` тј. `MulOrDiv` правила. Операнди су дефинисани у оквиру правила `Operand` (листинг 3, линија 25). Основни операнди су дефинисани у оквиру правила `BaseType` а то су подаци који могу бити следећег типа: `int`, `float`, `String`, `boolean` или `ID` (називи дефинисаних промењљивих). Поред основних, ALAS изрази могу да садрже и сложене операнде као што су

`CallFunction` и `ACLMessageProperty`. Правило `CallFunction` омогућује позив раније дефинисаних функција са одговарајућим параметрима (ако постоје). Применом `ACLMessageProperty` правила наводи се одређено својство које се односи на одговарајућа поруку коју један агент шаље другом приликом међуагентске комуникације.

Листинг 4. Део ALAS граматике са основним својствима агената

```
 1. Function:
 2.      type = Type name=ID
 3.      '(' paramlist*=Parameter[','] ')'
 4.      '{' s*=Statements '}'
 5. ;
 6. Parameter:
 7.      (type=Type | type=ObjectType) name=ID
 8. ;
 9. Action:
10.      preCondition=PreCondition?
11.      'action' name=ID '(' param=Parameter ')'
12.      '{' s*=Statements '}'
13. ;
14. PreCondition:
15.      '@PreCondition' '('performative=Performative ')'
16. ;
17. Init:
18.      'init' '{'s*=Statements'}'
19. ;
20. Arrived:
21.      'arrived' '{'s*=Statements'}'
22. ;
23. Statements:
24.      Variable
25.      | (Assignment ';')
26.      | IfStatement
27.      | ForStatement
28.      | WhileStatement
29.      | DoWhileStatement
30.      | (CallFunction ';')
31.      | ACLMessage
32.      | AgentClass
33.      | AID
34.      | ACLMessagePropertyDefinition
35.      | Move
36.      | Post
37.      | Log
38.      | (IncDec ';')
39.      | Return
40. ;
```

У листингу 4 дефинисана су правила (конструкти) која садрже основна својства Siebog агената.

`Function` конструкт се користи за извршавање одговарајућих операција у агенту и има својства приватних метода у ООР. Дефиниција `Function` правила почиње навођењем типа повратне вредности, затим следе јединствено име функције, листа параметара (сваки параметар се састоји од типа и имена) и тело функције. Тип параметра може бити основни (примитивни), као што је већ наведено, или кориснички. У оквиру ALAS језика постоје три корисничка типа: `ACLMessage`, `AID` и `AgentClass` који иначе представљају имена одговарајућих класа у оквиру Siebog-а.

Један од најчешће коришћених конструката који се користи за програмирање ALAS агената је `Action` који се користи за извршавање операција у оквиру агената при чему мора бити испуњен одговарајући предуслов. Предуслов се наводи у виду анотације применом кључне речи `@PreCondition` а наводи се непосредно пре дефиниције саме *акције* (листинг 4, линије 10, 14). Након предуслова, дефинише се акција на чијем почетку се мора наћи кључна реч `action` иза које следи име акције са обавезним тачно једним параметром. Параметар се дефинише исто као и код функције тако што се наведе тип и име параметра.

За иницијализацију одговарајућих компоненти агента користи се конструкт `Init`. Овај конструкт се дефинише навођењем кључне речи `init` након које се наводи блок одговарајућих наредби дефинисаних у оквиру правила `Statements`.

За дефинисање одговарајућих информација након пристизања агента на одредишну локацију користи се кључна реч `arrived` након које следи блок наредби. Овај конструкт је дефинисан у оквиру правила `Arrived` (листинг 4, линија 20).

У оквиру правила `Statements` налази се списак правила која се користе за дефинисање одговарајућих наредби у блоку тј. телу одговарајућег конструкта (листинг 4, линија 23). У листингу 5 приказана је имплементација граматичких правила за дефинисање наредби за контролу тока програма. Управо су то неке од најчешће коришћених компоненти из блока наредби. То су `if – elseif – else` наредбе и `for`, `while` и `do – while` петље.

Листинг 5. Скуп граматичких правила за дефинисање наредби за контролу тока

```
 1. IfStatement:
 2.   'if' '('condition=Assignment')' '{'s*=Statements'}'
 3.   elseif *= Elseif
 4.   else = Else?
 5. ;
 6. Elseif:
 7.   'else if' '('condition=Assignment')' '{'s*=Statements'}'
 8. ;
 9. Else:
10.   'else' '{'s*=Statements'}'
11. ;
```

```
12. ForStatement:
13.   'for' '('param=Variable
14.     param=Assignment ';'
15.     param=IncDec | param=Assignment) ')'
16.     '{'s*=Statements'}'
17. ;
18. IncDec:
19.   var1=IDType? ('++' | '--') var2=IDType?
20. ;
21. WhileStatement:
22.   'while' '('condition=Assignment')' '{'s*=Statements'}'
23. ;
24. DoWhileStatement:
25.   'do' '{'s*=Statements'}'
26.   'while' '(' condition=Assignment ')' ';'
27. ;
```

Једна од главних карактеристика ALAS-а су програмски конструкти. У листинзима 4, 5 и 6 приказана је имплементација правила која служе за дефинисање неких Siebog конструката као што су: `action`, `init`, `arrived` (листинг 4), `move`, `post` (листинг 6). Поред наведених конструката у листингу 6 су имплементирана додатна, сложенија правила за подршку мултиагентском окружењу Siebog.

Помоћу правила `ACLMessage`, `AgentClass` и `AID` могу се креирати инстанце истоимених класа. `ACLMessage` представља FIPA стандардизован начин за креирање порука које се размењују између агената. FIPA ACL (енг. Agent communication language - ACL) поруке садрже један или више параметара. Обавезан параметар је *перформатива* (енг. `performative`), а поред њега неки од параметара које ACL поруке могу садржати су:

- пошиљалац (енг. `sender`)
- прималац (енг. `receiver`)
- садржај (енг. `content`)
- коме одговорити (енг. `reply-to`)

Сви претходно наведени параметри имплементирани су у оквиру правила `ACLMessagePropertyDefinition`. Прво се помоћу повезаног референцирања (опција коју нуди textX, енг. link rule reference) наводи име неке инстанце класе `ACLMessage` након чега следи тачка, после које се наводи одговарајући параметар (`performative`, `sender`, `receiver`, `content` или `reply-to`) и његова вредност.

Свакако један од најбитнијих конструката ALAS језика је `move` пошто је то кључна компонента која се користи приликом кретања агената између клијентских и серверских страна. Након навођења једне од две кључне речи (`moveToServer` или `moveToClient`), у заградама се опционо наводи неки од

параматера ACL поруке (углавном садржај - `content`) или локација на коју агент треба да се пребаци. За управљање порукама користи се менаџер порука који за објављивање креиране поруке користи конструкт `post` који је имплементиран у оквиру граматичког правила `Post`. Након кључне речи `post` у заградама се наводи једна од инстанци класе `ACLMessage` која садржи одговарајуће информације на основу којих агент изводи предвиђене активности.

Правило `Log` се користи за испис наведених информација а правило `Return` омогућује навођење повратне вредности у блоку наредби у оквиру одговарајућих конструката. Повратна вредност се наводи у облику израза (`Expression`) који је раније описан.

Листинг 6. Скуп граматичких правила за дефинисање Siebog конструката

```
ACLMessage:
  type=ObjectType name=ID '=' 'new' type=ObjectType '(' ')' ';'
;
AgentClass:
  type=ObjectType name=ID '=' 'new' type=ObjectType
  '(' module=STRING ',' ejbname=STRING ',' path=STRING ')' ';'
;
AID:
  type=ObjectType name=ID '=' 'new' type=ObjectType
  '('aid=STRING ',' host=STRING ',' agclass=[AgentClass] ')' ';'
;
ACLMessagePropertyDefinition:
  aclmessage= [ACLMessage] '.' property=Property ';'
;
Property:
  PerformativeDef | Receivers | Sender | Content | ReplyTo
;
PerformativeDef:
  'performative' '=' performative=Performative
;
Receivers:
  'receivers' '=' '{' receivers+=ID[','] '}'
;
Sender:
  'sender' '=' sender=[AID]
;
Content:
  'content' '=' content=STRING
;
ReplyTo:
  'replyTo' '=' replyTo=[AID]
;
Move:
  ('moveToServer' | 'moveToClient')
  '(' (host = ACLMessageProperty | host = STRING)? ')' ';'
;
Post:
```

```
39.   'post' '(' msg=ID ')' ';'
40. ;
41. Log:
42.   ('log'|'print') '('params+=Params['+'] (',' param=BOOL)?')' ';'
44. ;
45. Params:
46.   ACLMessageProperty | CallFunction | ID | STRING
47. ;
48. Return:
49.   'return' exp=Expression ';'
50 ;
```

7.1.1 Подршка за DNARS

Најновија верзија ALAS-а, имплементирана применом textX оквира, укључује подршку за DNARS и заснована је на претходној верзији развијеној применом Xtext оквира (Bettini 2013; Sredojević, Vidaković i Okanović 2015; Sredojević i dr. 2017). Пошто textX и Xtext користе исте нотације за опис граматика језика, није било тешко прилагодити Xtext верзију ALAS-а textX оквиру (Dejanović i dr. 2016; Dejanović 2019b).

Поред минималне модификације постојећих особина и правила како би се прилагодили textX-у, најновија верзија ALAS-а имплементира нове концепте који уводе елементе вештачке интелигенције засноване на DNARS-у. Имплементирана су четири нова правила која представљају најбитније компоненте DNARS-а: `DnarsBeliefs`, `DnarsAddedUpdated`, `DnarsQuestion` и `DnarsInference`. Ова правила су имплементирана у оквиру `Feature` правила (листинг 1, линија 22). Део граматике језика у којем је имплементирана подршку за DNARS је приказан у листингу 7.

Листинг 7. Део граматике ALAS језика за подршку DNARS-у

```
 1. DnarsBeliefs:
 2.   'beliefs' '{' statements += DnarsStatement[','] ';' '}'
 3. ;
 4. DnarsAddedUpdated:
 5.   annotation = BeliefAU '{'s*=Statements'}'
 6. ;
 7. BeliefAU:
 8.   ('beliefadded' | 'beliefupdated')
 9.   '(' (judgement = Judgement ',')? param = ID')'
10. ;
11. DnarsQuestion:
12.   'Terms' answer = ID '=' 'question'
13.   '(' (question=Question1 | question=Question2)(','num=INT)?')'';'
14. ;
15. DnarsInference:
16.   'Statements' answer = ID '=' 'inference'
17.   '(' judgement = Judgement (',' num=INT)?')' ';'
```

```
18. ;
19. Judgement:
20.   subj = Term (copula = '->' | copula = '~') pred = Term
21. ;
22. DnarsStatement:
23.   judgement = Judgement truth = Truth
24. ;
25. Term:
26.   AtomicTerm | CompoundTerm
27. ;
28. Truth:
29.   '(' number = NUMBER ',' number = NUMBER ')'
30. ;
31. AtomicTerm:
32.   type = BaseType
33. ;
34. CompoundTerm:
35.   PrefixCompTerm | InfixCompTerm | ImgCompTerm
36. ;
37. PrefixCompTerm:
38.   '(' connector = Connector terms+=AtomicTerm ')'
39. ;
40. InfixCompTerm:
41.   '(' term=AtomicTerm cts+=CompTerm ')'
42. ;
43. ImgCompTerm:
44.   '(' (image = '/' | image = /[\\][\\]/) term = AtomicTerm
45.   (option = Term1 | option = Term2) ')'
46. ;
47. CompTerm:
48.   connector=Connector term=AtomicTerm
49. ;
50. Term1:
51.   plc = '*' term = AtomicTerm
52. ;
53. Term2:
54.   term = AtomicTerm plc = '*'
55. ;
56. Connector:
57.   'x'| '/' | /[\\][\\]/
58. ;
59. Question1:
60.   '?' (copula = '->' | copula = '~') term=Term
61. ;
62. Questio2:
63.   term=Term (copula = '->' | copula = '~') '?'
64. ;
```

`DnarsBeliefs` компонента дозвољава агентима да додају нове доказе (веровања) у базу знања. Ова компонента дозвољава само специфичне формате веровања подржаних NAL синтаксом. `DnarsBeliefs` компонента дозвољава

дефинисање осам типова реченица чија је синтакса описана у одељку 3.1. У оквиру `DnarsBeliefs` правила прво се наводи кључна реч `beliefs`, а затим скуп веровања. Синтакса веровања је дефинисана применом правила `DnarsStatement` помоћу којег је представљена структура NAL реченица. Ово правило се састоји из нова два правила: `Judgement` и `Truth`. `Judgement` правило имплементира структуру веровања без истинитосне вредности која је имплементирана у оквиру правила `Truth`. Као што је раније описано, обавезни делови NAL веровања су субјекат, релација наслеђивања (–>) или релација сличности (~) и предикат при чему су субјекат и предикат термови који су дефинисани у оквиру правила `Term`. Субјекат и предикат реченице могу бити атомски или сложени термови. Атомски термови су дефинисани правилом `BaseType` док су сложени термови дефинисани `PrefixCompTerm`, `InfixCompTerm` и `ImgCompTerm` правилима чије су дефиниције приказане у листингу 7. `Truth` правило је дефинисано са две нумеричке вредности које се налазе у заградама и раздвојене су зарезом.

`DnarsAddedUpdated` компонента се састоји од два конструкта. Први конструкт, означен кључном речи `beliefadded` користи се за обавештавање агента када се нови докази додају у базу знања. Други конструкт који је означен са `beliefupdated`, обавештава агента када је неко веровање из базе знања измењено. `beliefadded` конструкт може имати два аргумента при чему је први опцион. Ако овај аргумент није наведен, агент ће бити обавештен о свим новим доказима додатим у базу знања. Ако је параметар наведен, онда он има структуру NAL реченице. Једноставним упоређивањем новог веровања са осталим веровањима из базе знања закључује се да ли ново веровање већ постоји у бази или не и агент добија информацију о томе. Други аргумент је потребан и то је име променљиве. Ова променљива садржи листу доказа додатих у базу знања. Након аргумената, дефинише се тело `DnarsAddedUpdated` компоненте које се састоји из одговарајућих наредби дефинисаних у оквиру правила `Statements` (листинг 4, линија 23). Основни циљ ових конструката је да обавештавају агента о променама у бази знања. На основу ових промена, агент ће извршити операције дефинисане у телу `beliefadded` или `beliefupdated` конструкта.

`DnarsQuestion` компонента представља DNARS конструкт који омогућује агенту да доноси најбоље могуће закључке на постављена питања. Питања могу бити постављена на два начина: ‘*? релација П*’ или ‘*С релација ?*’, где су С и П субјекат и предикат, а *релација* је или релација наслеђивања (–>) или релација сличности (~). Овај конструкт се састоји од кључне речи `Terms` (што значи да повратна вредност треба да буде листа одговарајућих термова), коју следи име променљиве у коју се смешта резултат. Затим, следи кључна реч `question` и један или два параметра у загради. Први параметар је обавезан и представља питање у једном од претходно наведена два формата. Други параметар је опцион и служи да клијент сигнализира агенту колико одговора жели. Ако је параметар изостављен, подразумевани број одговора је 1. Систем може понудити више

различитих одговора при чему агент узима одговоре са најбољим истинитосним вредностима. На пример, ако у бази постоје 3 веровања: *голуб –> птица (0.9, 0.7), орао –> птица (0.9, 0.8)* и *врабац –> птица (0.8, 0.7)* и клијент од агента тражи да му наброји 2 птице, тј. применом ALAS синтаксе: *Terms t = question*(*? –> птица*, 2), агент ће на основу истинитосних вредности и дефиниције 5.1 закључити да су најбољи одговори *орао* и *голуб*, у том редоследу.

`DnarsInference` конструкт на основу постављеног питања у форми '*C релација П*' даје најбољи могући одговор у форми '*C релација П (f, c)*' где се као и код `DnarsQuestion` конструкта у случају више одговора, најбољи одговори добијају рачунањем очекивања учесталости (дефиниција 5.1). Синтакса `DnarsInference` конструкта је следећа: прво се пише кључна реч `Statements` (што значи да одговор треба да буде у виду листе одговарајућих реченица (*statements*)) а затим име променљиве у коју се смешта резултат. Након тога следи кључна реч `inference` и један или два параметра у загради. Први параметар је обавезан и то је питање које поставља клијент у форми '*C релација П*'. Други параметар је опцион и он представља број одговора које клијент жели. Ако други параметар није наведен, подразумевана вредност је 1. Систем претражује базу знања и на основу правила *извођења уназад* проналази најбоље могуће одговоре на постављено питање.

Комплетна граматика, као и истале компоненте ALAS језика заједно са визуалним приказом дати су у пројекту отвореног кода који је доступан на GitHub-y[16].

7.2 Синтакса и семантика ALAS језика

Синтакса неког програмског језика представља форму исправности програма, при чему машина на основу синтаксе не може закључити какво је значење програма нити који су резултати његовог извршавања за разлику од семантике, која представља смисао (значење) који има у програму нека правилна конструкција програмског језика. Синтакса и семантика језика су уско повезане, мада добро написана синтакса не подразумева и исправну семантику као и у обрнутом случају, добра семантика нема никакво значење ако синтакса није исправна.

Синтакса ALAS језика је дефинисана скупом једнозначно дефинисаних правила граматике која су настала комбинацијом регуларних израза и EBNF нотације. Граматика ALAS језика је дизајнирана тако да поред синтаксно исправног програма омогућује писање делимично, семантички исправног,

[16] https://github.com/sredojevic/ALAS

програма. Као и код већине програмских језика, обрада семантике ALAS језика се врши тек након синтаксне анализе. Главна сврха ALAS програма (агента) је извршавање у хетерогеним окружењима. То подразумева обавезно трансформисање ALAS кода у језик одредишне платформе. Потпуна провера семантичке исправности ALAS кода се проверава за време генерисања изворног кода који ће се извршавати на одговарајућој платформи. Пошто је конверзија ALAS кода обавезна то подразумева и обавезну проверу семантике ALAS језика. Међутим, када је семантика у питању, скоро ју је немогуће формализовати у потпуности, па се може десити да неке семантичке грешке компајлер не може да открије а један од начина за решавање овог проблема је тестирање програма.

Пример синтаксе једноставног, мобилног ALAS агента који се може пребацивати са клијентске стране на серверску и обрнуто, може се видети у листингу 8.

Листинг 8. Пример синтаксе мобилног ALAS агента

```
 1. package siebog.agents;
 2.
 3. agent MobileAgent{
 4.   int num = 0;
 5.
 6.   arrived {
 7.      num++;
 8.      if (isServer == true) {
 9.         print("I'm at server, and the number is: " + num);
10.         log("I'm at server, and the number is: " + num);
11.      } else {
12.         log("I just came from to my browser,
13.               and the number is: " + num);
14.      }
15.   }
16.
17.   action AgentAction(ACLMessage msg) {
18.      String content = msg.content;
19.      if (isServer == true) {
20.         print("I'm on the server and I received the
21.                        following message: " + msg);
22.         if (content == "move") {
23.            moveToClient();
24.         } else {
25.            moveToServer(msg.content);
26.         }
27.      } else {
28.         log("A'm in a client browser,
29.               and I have received the following message: " + msg);
30.         if (content == "move") {
31.            moveToServer();
32.         }
33.      }
34.   }
```

```
35. }
```

На почетку агента наведен је пакет у којем се агент налази. Агент је дефинисан помоћу кључне речи `agent` након које следи име агента - `MobileAgent`.

У телу агента су дефинисана два конструкта, `arrived` и `action`. Пре имплементације ових конструката дефинисана је променљива `num` којој је додељена вредност `0`. Конструкт `arrived` служи за дефинисање низа наредби које треба да се изврше када агент пристигне на одредиште (сервер или клијент). То су углавном команде које се користе за обавештавање да је агент стигао на одредиште. У случају агента из листинга 8, вршиће се провера да ли је агент стигао на сервер или на клијент и на основу те информације исписаће се одређене поруке и на серверској страни (помоћу кључне речи `print`) и на клијентској страни (применом кључне речи `log`).

За извршавање одговарајућих активности користи се конструкт `action` у оквиру кога се дефишу наредбе које је потребно извршити на клијентској или серверској страни уколико су испуњени одговарајући предуслови који се, како је наведено у опису граматике језика, дефинишу у виду анотације применом кључне речи `@PreCondition`. У овом случају конструкт `action` нема предуслове што значи да ће се команде које су у њему дефинисане извршити без обзира на садржај поруке коју агент добије као списак активности које треба да изврши. Слично као и код конструкта `arrived`, проверава се да ли је агент на клијентској или на серверској страни. Поред тога, проверава се садржај поруке која указује агенту шта да ради.

Ако је агент на серверској страни прво ће се исписати одговарајућа порука у серверском прозору за испис порука, а затим ће се проверити да ли је садржај поруке једнак речи `"move"`, па ако јесте, помоћу команде `moveToClient()` агент ће се пребацити на клијентску страну, у супротном, ако је садржај поруке различит од `"move"`, агент ће се помоћу команде `moveToServer(content)` пребацити на чвор у кластеру чија је адреса наведена у садржају поруке.

Даље, ако је агент на клијентској страни исписаће се одговарајућа порука у клијентском прозору за испис порука а затим ће се проверити да ли је садржај поруке једнак речи `"move"`, па ако јесте, помоћу команде `moveToServer()` агент ће се пребацити на серверску страну.

Пример синтаксе ALAS агента који садржи одговарајуће конструкте за подршку DNARS-у је приказан у листингу 9. Овај текстуални модел језика је написан на основу граматике описане у претходном поглављу.

Листинг 9. Пример синтаксе ALAS агента са подршком за DNARS

```
1. domain(dnarsdomain) agent DnarsTestAgent {
2.   beliefs {
3.      mandarina -> voće (0.9, 0.7),
4.      narandža -> voće (0.9, 0.8),
5.      jabuka -> voće (0.8, 0.7),
```

```
 6.      pčela -> insekt (1.0, 0.9),
 7.      citrus ~ limun (0.76, 0.83)",
 8.      (x pčela med) -> pravi (1.0, 0.9),
 9.      (x Ernest_Hemingvej 1899) -> datum_rođenja (1.0, 0.9),
10.      jede -> (x zec šargarepa) (1.0, 0.9)
11.   }
12.   beliefadded(beliefs){
13.      log('Sva nova verovanja: ' +beliefs);
14.   }
15.   beliefadded(banana -> voće, beliefs){
16.      log('Banana je vrsta voća: ' +beliefs);
17.   }
18.   beliefupdated(beliefs){
19.      log('Sva izmenjena verovanja: ' +beliefs);
20.   }
21.   beliefupdated((x pčela med) -> pravi, beliefs){
22.      log('Izmenjeno verovanje: ' +beliefs);
23.   }
24.   Terms t1 = question((\\ datum_rođenja Ernest_Hemingvej *) -> ?);
25.   Terms t2 = question(? -> voće, 3);
26.   Statements s1 = inference(pčela -> insekt);
27.   Statements s2 = inference(jabuka -> voće, 2);
28.   }
29. }
```

Након имена пакета у који се смешта агент, наводи се име агента помоћу кључне реци `agent`. Као што је раније описано, на почетку се може навести име домена који представља базу знања коју агент користи. У овом случају применом `domain` конструкта наведено је да агент користи базу знања из `dnarsdomain` домена.

Помоћу кључне речи `beliefs` наводе се нови докази који се додају у базу знања и написани су у форми NAL реченица.

Затим, дефинисана су два `beliefadded` конструкта. Први конструкт је пример обавештавања агента о свим новим додатим доказима у базу знања представљену графом особина. У променљиву `beliefs` се смешта листа свих доказа дефинисаних применом `beliefs` конструкта. Конструкт `log` служи за испис резултата смештеног у `beliefs` променљиву. Поред конструкта `log`, у телу `beliefadded` конструкта може се наћи и низ других операција дефинисаних правилом `Statements` у граматици језика. Други конструкт има два параметра при чему је први параметар NAL реченица *банана –> воће* што значи да ће агент бити обавештен само ако је додат нови доказ *банана –> воће*. У случају агента из листинга 9, тај доказ не постоји у бази веровања па агент неће извршавати операције које се налазе у телу овог конструкта.

`beliefupdated` конструкт ради на истом принципу као и `beliefadded` само што он обавештава агента када се неко веровање измени. Нпр. ако у бази знања постоји реченица *мандарина –> воће* (*0.9, 0.8*) и затим се помоћу `beliefs`

конструкта дода ново веровање *мандарина –> воће* (*0.82*, *0.91*) у бази знања ће остати само једна реченица *мандарина –> воће* чија ће истинитосна вредност на основу правила *ревизије* описаног у поглављу 3.1 бити (*0.84*, *0.93*).

Следећи конструкт наведен у приказаном ALAS агенту је `question`. Агенту се прослеђује питање у форми '*? релација П*' или '*С релација ?*' и број који означава колико одговора клијент жели, при чему овај број није обавезан и ако се не наведе, подразумевана вредност је 1. У првој реченици, клијент поставља питање агенту '*када је рођен Ернест Хемингвеј?*' у форми NAL реченице: (\\ *датум_рођења Ернест_Хемингвеј* *) –> *?* без додатног бројног параметра. Агент претражује базу знања (граф особина) и у променљиву t_1 уписује годину рођења Ернеста Хемингвеја под условом да у бази знања постоји таква информација. У овом случају постоји веровање које говори када је рођен Ернест Хемингвеј па ће резултат бити 1899. У другом примеру, клијент тражи од агента да наброји 3 воћке. Након претраге базе знања, агент ће у листу t_2 уписати 3 резултата који имају највећи степен истинитости који се одређује рачунањем очекивања учесталости (дефиниција 5.1). У случају агента из листинга 9, у листу ће се уписати *наранџа* (*e* = 0.82), *мандарина* (*e* = 0.78) и *јабука* (*e* = 0.71), тим редоследом.

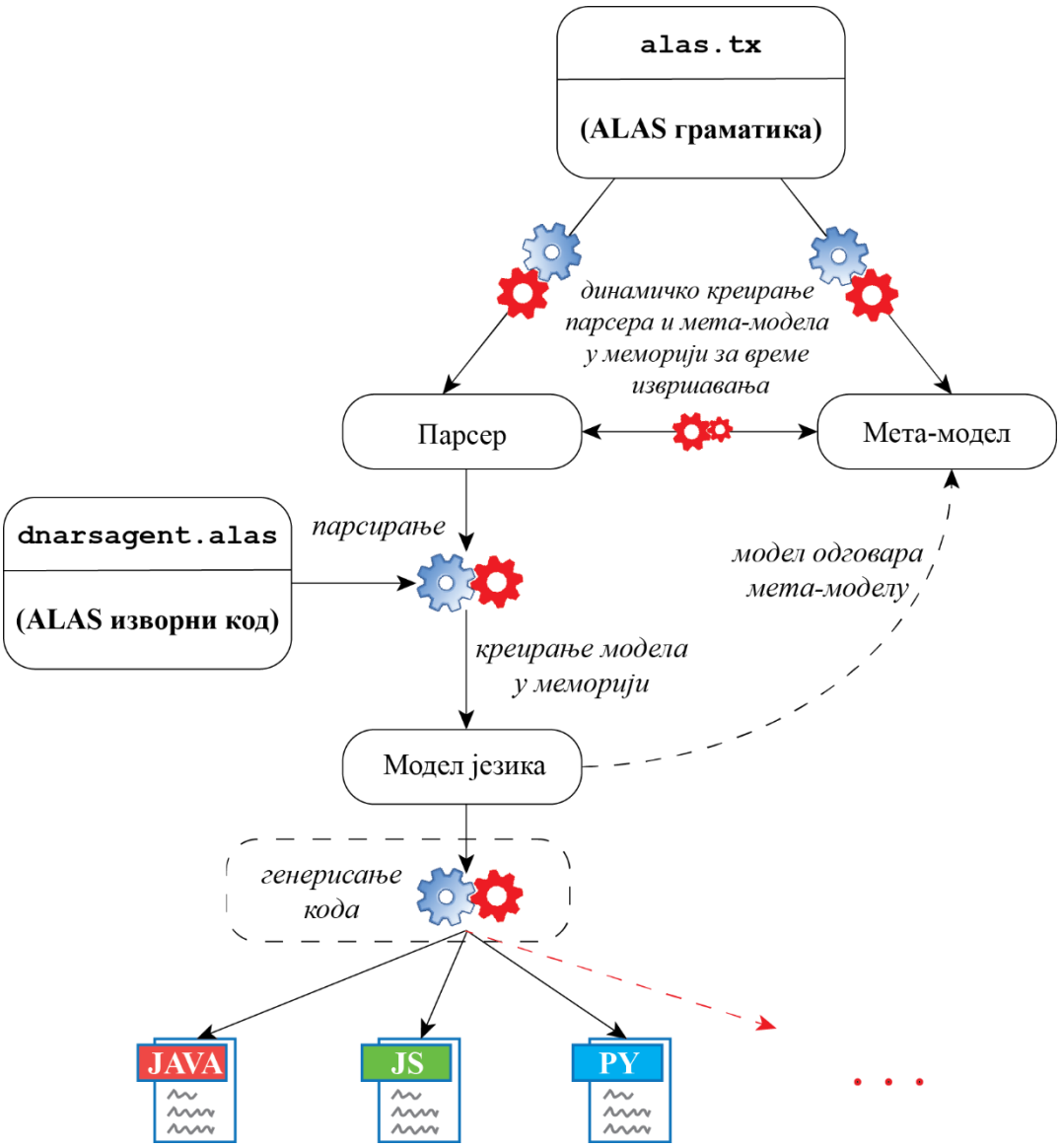
На сличан начин функционише и конструкт `inference`. Клијент агенту прослеђује питање у форми '*С релација П*' и опционо број који означава колико одговора жели. У примеру: `Statements s1 = inference(pčela -> insekt)`, клијент жели један одговор који ће се уписати у листу s_1 у форми *пчела –> инсект* (*f*, *c*) при чему се учесталост (*f*) и стабилност (*c*) одговора добијају применом правила *извођења уназад*.

7.3 ALAS компајлер, моделовање језика и визуализација

Компајлер је рачунарски програм који на свом улазу прихвата неки програм који је написан углавном на језику вишег нивоа апстракције и који се обично зове изворни код а на свом излазу генерише одговарајући код нижег нивоа апстракције (углавном машински тј. објектни код који се може директно извршавати на хардверу рачунара).

Програм може да се преведе с вишег програмског језика само ако у њему нема синтаксних грешака, што на почетку процеса превођења такође проверава компајлер (преводилац или анализатор) и издаје поруке о грешкама. Иако не постоје синтаксне грешке, програм може бити преведен на језик нижег нивоа али то није гаранција да ће се при извршавању програма добити добри резултати пошто се може десити да програм семантички није исправан. За проналажење грешака у преведеном програму могу се користити програми за отклањање грешака - *дебагери*.

Имајући ALAS као специфичан програмски језик и мета-модел изражен граматиком језика, агенти се могу лако писати помоћу једноставних окружења за унос текста (текст едитора). Поред других предности textX-а, велика предност је независност од развојних окружења што пре свега доприноси бржем извршавању. Коришћењем одговарајућих команди, програм (агент) се може компајлирати из командне линије и на тај начин проверити синтаксу тј. да ли модел одговара граматици (мета-моделу) и евентулано постојање грешака. На слици 7.4 приказана је архитектура и принцип рада textX компајлера.



Слика 7.4. Принцип рада и архитектура textX-а за ALAS језик

Основни део компајлера је синтаксни модул где се наводи граматика ALAS језика и модел агента у текстуалном облику. TextX на основу граматике језика у тренутку извршавања динамички креира модел парсера, апстрактну синтаксу

језика у виду мета-модела и апстрактну синтаксу мета-језика тј. textX-а која представља мета-метамодел (Dejanović i dr. 2016).

Мета-модел садржи све информације о језику као и скуп Python класа конструисаних на основу правила граматике. Приликом парсирања текстуалног модела (ALAS изворног кода агента), парсер динамички креира конкретно синтаксно стабло тј. меморијски модел у виду графа Python објеката који ће одговарати мета-моделу.

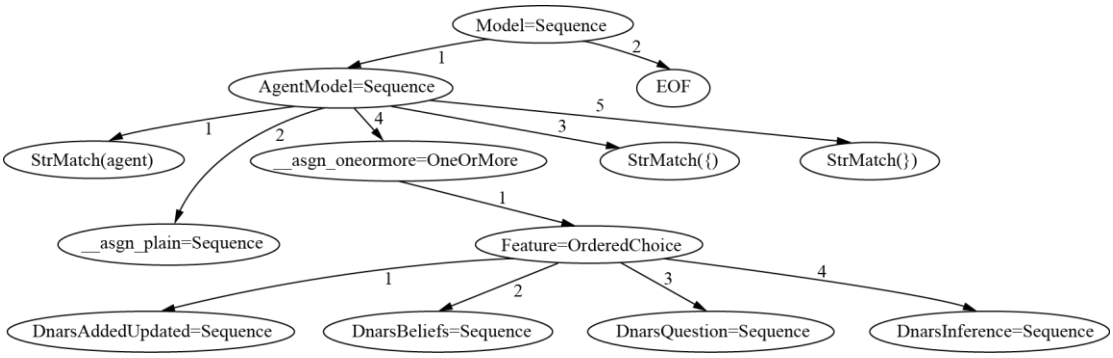
Парсер проверава валидност граматике и компатибилност модела (агента) са мета-моделом (граматиком). У сваком тренутку развоја могу се проверити валидност граматике и модела језика. Ако модел одговара мета-моделу, у конзоли ће се након извршавања команде исписати порука да су модел и мета-модел исправни а уколико постоји грешка, у конзоли ће се исписати која грешка је у питању и тачна локација грешке. Тако добијени модел се може користити нпр. за генерисање изворног кода за неки други програмски језик, може се интерпретирати, детаљно анализирати итд. У наставку дисертације (поглавље 7.4) описан је процес конверзије ALAS кода у Java и JavaScript језике.

Једна од важних ствари када је дизајн језика у питању је визуализација одговарајућих компоненти како би се омогућила боља прегледност. Применом *GraphViz* (2018) софтверског пакета textX омогућује визуализацију. *GraphViz* одговарајуће графове креира на основу *dot* фајлова који се генеришу приликом превођења програма. Приликом компајлирања, генеришу се одређене компоненте које се могу визуализовати (парсер, стабло парсирања, модел, мета-модел итд.).

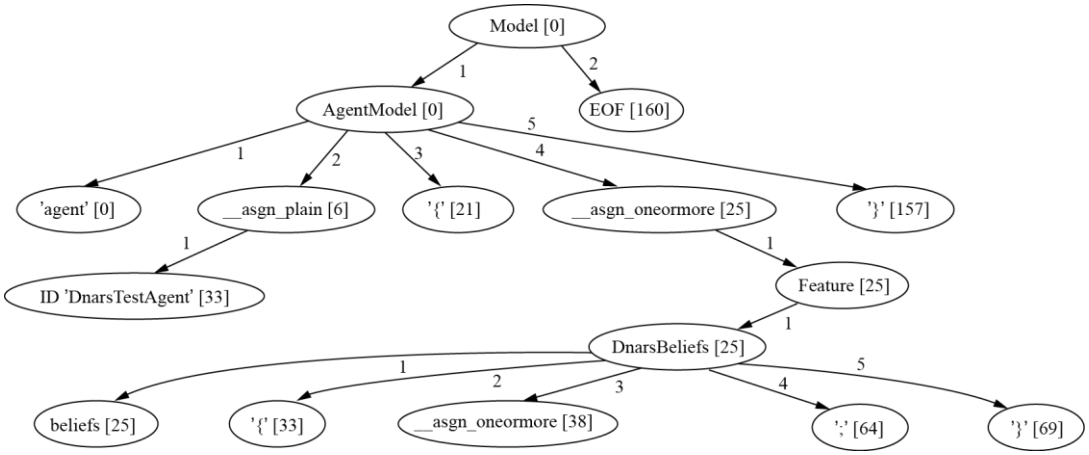
У наставку су дати делимични визуелни прикази парсера (слика 7.5), стабла парсирања тј. конкретног синтаксног стабла (слика 7.6), мета-модела (слика 7.7) и модела (за пример агента из листинга 9, слика 7.8). Комплетна визуализација свих компоненти се налази у оквиру ALAS пројекта који је доступан на GitHub-у[16].

textX такође пружа могућност проналажења и решавања грешака или неких других недостатака на нивоу мета-модела и модела (Dejanović 2019b). Ако је омогућена опција (`debug = True`), textX ће штампати разне поруке о грешкама.

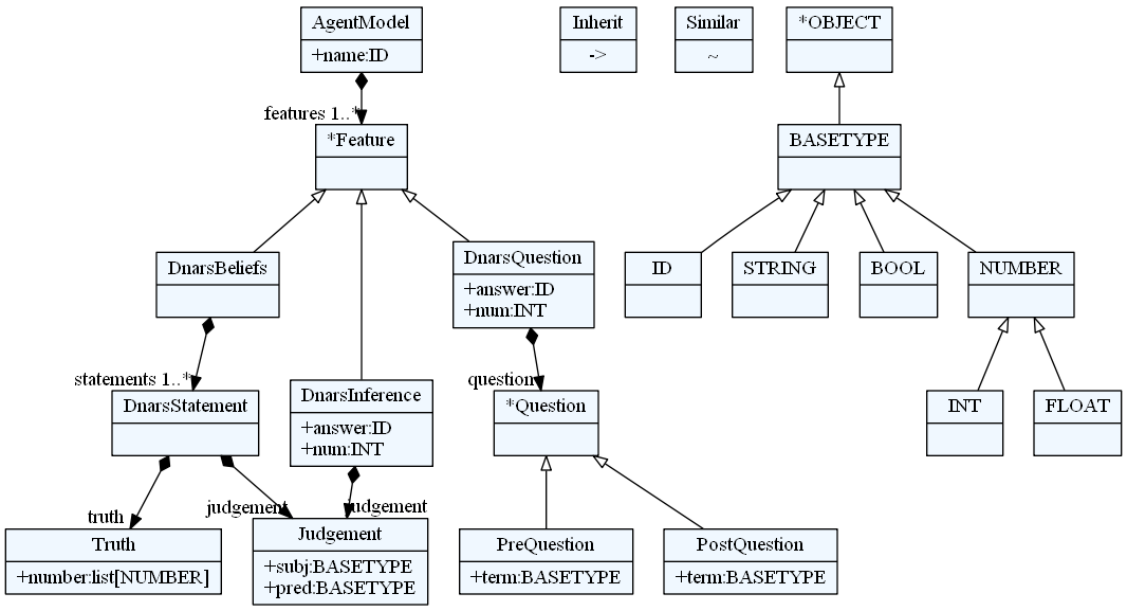
Цео процес превођења програма, визуализација свих компоненти језика, као и генерисање нових изворних кодова могу се аутоматизовати писањем одговарајуће Python скрипте. Python скрипта за визуализацију ALAS компоненти се такође налази у оквиру ALAS пројекта на GitHub-у.



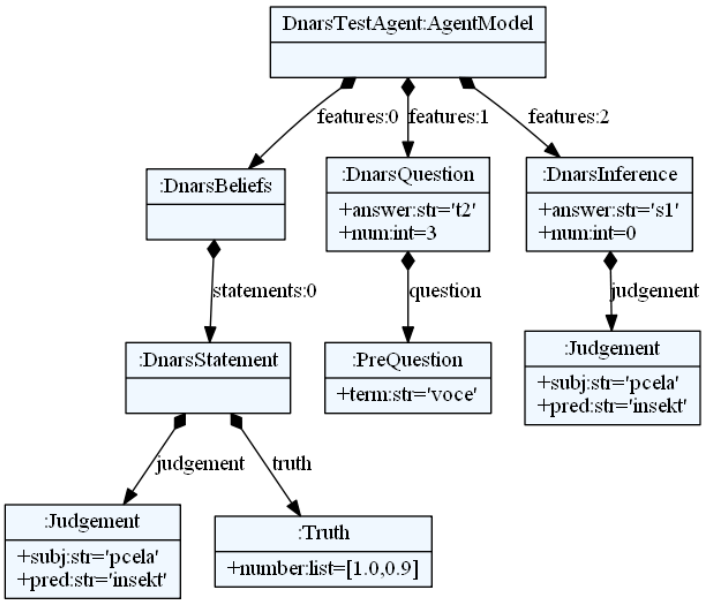
Слика 7.5. Део модела парсера ALAS језика



Слика 7.6. Део конкретног синтаксног стабла



Слика 7.7. Део мета-модела ALAS језика



Слика 7.8. Део ALAS модела за пример агента из листинга 9.

7.4 Мобилност ALAS агената

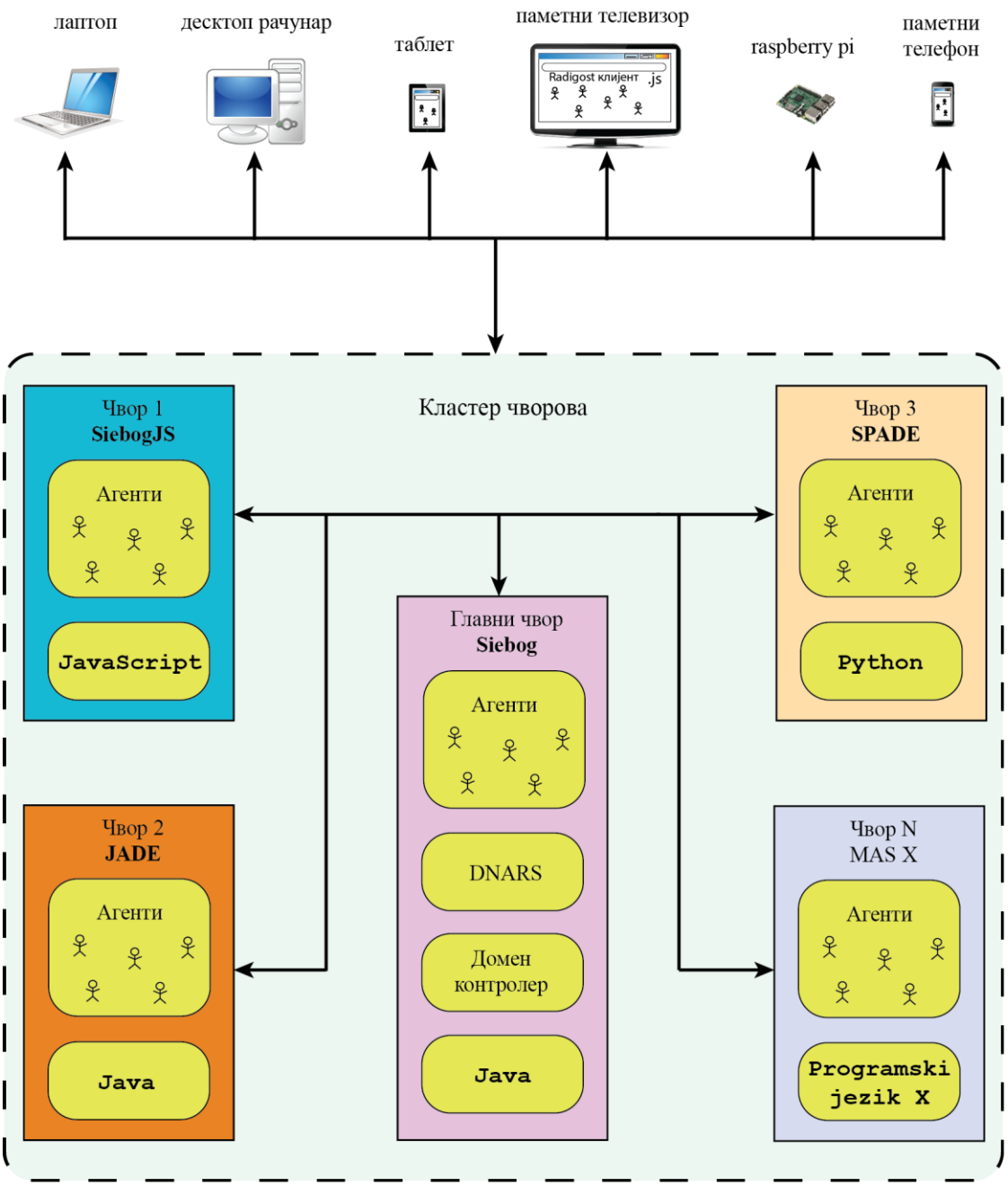
Као што је раније наведено, главни мотив за развој ALAS језика је био да се омогући мобилност агената у хетерогеним окружењима. Уз помоћ мобилности, агент може напустити своју *host* машину на којој се тренутно извршава и наставити извршавање на некој другој машини у мрежи. Постоје различите врсте мобилности агената у зависности од платформи које чине кластер (Mitrović i dr. 2011). Оне су класификоване на следећи начин:

- *Хомогена* (енг. Homogeneous): сви чворови MAS-а у којима се извршавају агенти, имплементирани су у истом програмском језику (нпр. Java) нудећи исти скуп API-а (енг. Application Programming Interface – интерфејс за програмирање апликација), итд. Приликом кретања између чворова мултиагентског система, није потребно мењати агентски код. Овај тип мобилности је најлакши за имплементацију.
- *Вишеплатформска* (енг. Cross-platform): агенти се крећу између различитих мултиагентских система заснованих на истим виртуелним машинама. У овом случају једино треба прилагодити API позиве.
- *Регенерација агената* (енг. Agent-regeneration): агенти се крећу између различитих инстанци истог MAS-а које могу бити имплементиране применом различитих технологија (Java, JavaScript, Python итд.). Због ове чињенице, агентски изворни код се мора регенерисати у код одредишног MAS чвора. У овом случају сви чворови нуде исте скупове API-а.
- *Хетерогена* (енг. Heterogeneous): чворови између којих се крећу агенти су инстанце различитих MAS-а и могу се заснивати на различитим виртуелним

машинама. Неопходна је регенерација агентског кода и прилагођавање различитим скуповима API-а.

Мреже мултиагентских система сматрају се хетерогеним ако укључују системе са различитим скуповима интерфејса, који се извршавају на различитим виртуелним машинама. Развијање агента који може да ради у таквим окружењима је тежак задатак, јер процес захтева регенерацију извршног кода агента, као и модификације у начину на који комуницира са окружењем. Са главним циљем пружања ефективног решења за проблем мобилности хетерогених агената, предложен је нови агентски оријентисани програмски језик, назван ALAS. Нови језик такође обезбеђује скуп програмских конструката који ефикасно скривају комплексност целокупног процеса развоја агената.

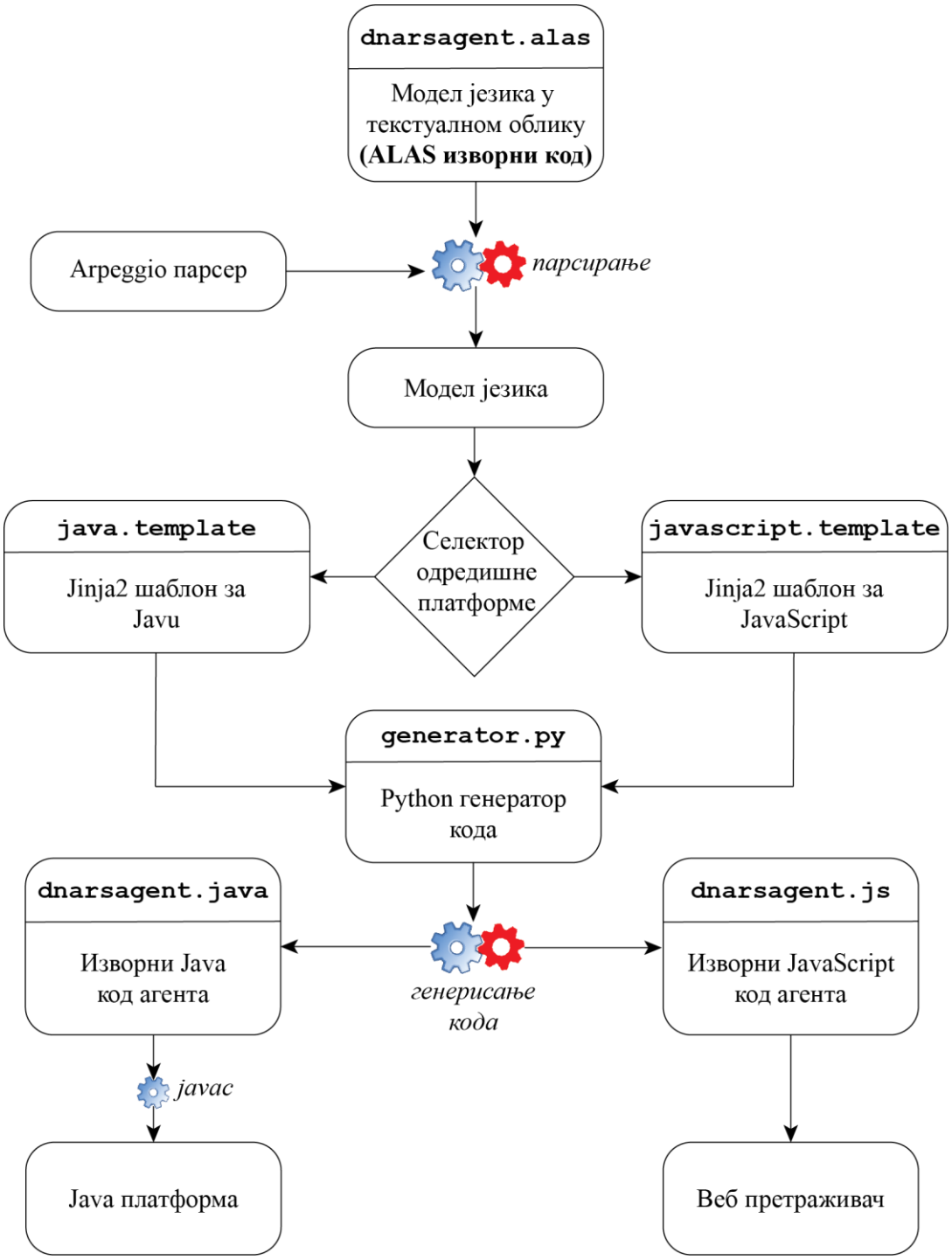
Основни задатак ALAS-а је омогућавање агентима да се крећу између разних патформи без обзира на ком су језику имплементиране. Као што је већ наведено, ALAS агенти имају способност конверзије у имплементациони језик одредишне платформе и тако без проблема могу наставити своје извршавање на тој платформи. Раније је већ речено да је применом Xtext оквира требало доста труда да би се ALAS код конвертовао у неки други (не-Java) програмски језик. Због ове чињенице настављено је са развојем ALAS-а применом textX оквира за развој DSL чије су предности описане раније у раду. Arpeggio парсер на коме се textX заснива је знатно флексибилнији, омогућује приступ било којем сегменту модела језика и то је једна од главних предности textX-а кад је у питању конверзија ALAS кода у неки други програмски језик. На слици 7.9 приказана је мрежа хетерогених чворова а ALAS је дизајниран управо тако да омогући несметано кретање агената у оваквом окружењу.



Слика 7.9. Мрежа хетерогених чворова

7.4.1 Конверзија ALAS кода у Javu

textX је заснован на Python-у и за време парсирања textX аутоматски генерише скуп Python класа на основу правила граматике. Управо је због ове чињенице генератор Java кода креиран у Python програмском језику.



Слика 7.10. Процес конверзије ALAS изворног кода у Javu и JavaScript

За генерисање кода textX користи *Jinja2* (2018) шаблон. Овај шаблон омогућује једноставно и ефикасно генерисање кода. Међутим, textX не зависи од *Jinja2* шаблона тако да се за генерисање кода могу користити и други шаблони за разлику од Xtext-а који дозвољава примену само сопственог, уграђеног шаблона. Поред једноставности, једна од главних особина *Jinja2* шаблона због које је он коришћен, јесте његова потпуна прилагодљивост Python језику с обзиром да се

он користи као генератор кода. Цео процес конверзије ALAS кода у изворни Java, тј. JavaScript код приказан је графички на слици 7.10.

У Python код (*alas_generator.py*) се учитају модел језика (модел се аутоматски парсира након чега се генерише скуп Python објеката) и одговарајући *Jinja2* шаблон за Java код (*java.template*) након чега се у Python генератору врши обрада и као завршна фаза генерише се одговарајући Java изворни код. Листинг 10 и листинг 11 приказују неке делове фајлова *java.template* и *alas_generator.py*.

Сви конструкти из ALAS кода се трансформишу у одговарајуће компоненте у Java језику и у наставку је укратко описан поступак мапирања неких ALAS конструката.

Листинг 10. Део *Jinja2* шаблона за Javu

```
{% if agent.package != None %}
  package {{agent.package.pack|join('.')}};
{% endif %}
{{agent|imports}}

{% if agent.state == 'stateful' %} @Stateful
{% else %} @Stateless {% endif %}
@Remote(Agent.class)
{% if agent.domain != None %}
@Domain(name = "{{agent.domain.name}}")
{% endif %}
public class {{agent.name}} extends XjafAgent {
  {{agent|logger}}

  {% for feature in agent.features %}
     {% if feature.__class__.__name__ == 'Variable' %}
        {% if defined_variables_pairs.append
              (feature|defined_variables_list) %} {% endif %}
        {% if all_variables_pairs.append
              (feature|all_variables_list) %}{% endif %}
        {{feature|global_variable(func_info)}}
     {% endif %}
  {% endfor %}

  @Override
  protected void onInit(AgentInitArgs args){
     {% for feature in agent.features %}
        {% if feature.__class__.__name__ == 'Init' %}
           {{feature|body(defined_variables_pairs
              |one_dimensional_list(defined_variables_pairs),
              all_variables_pairs|one_dimensional_list
              (all_variables_pairs), func_info)}}
        {% endif %}
     {% endfor %}
  }

  @Override
```

```
38.   protected void onMessage(ACLMessage msg) {
39.      {{agent|onMessage_conditions(defined_variables_pairs
40.            |one_dimensional_list(defined_variables_pairs),
41.            all_variables_pairs|one_dimensional_list
42.            (all_variables_pairs), func_info)}}
43.   }
44. }
```

Jinja2 шаблон садржи статичке делове који се само пресликавају у Java код и динамичке делове чији садржај зависи од модела језика. У свим листинзима у дисертацији у којима су приказани делови *Jinja2* шаблона, статички код је наглашен подебљаним текстом. Пошто је агент у ALAS-у еквивалент класи у OOP, генератор ће за сваки написан агент генерисати одговарајућу Java класу чије ће име бити исто као и име агента.

У *Jinja2* шаблону променљиви делови се пишу у оквиру двоструких витичастих заграда, као на пример `{{agent.name}}`. На овај начин се приступа одговарајућим компонентама модела језика а у овом случају приступа се имену агента. Логика извршавања (као што су петље и услови) се постиже таговима у оквиру једноструких витичастих заграда са знаком процента (%) на почетку и на крају услова као на пример:

```
{% if agent.state == 'stateful' %}
  // ...
{% endif %}
```

Као што је раније наведено, Arpeggio парсер генерише модел језика у форми Python објеката (слика 7.8) који нису читљиви и не могу се лако мапирати у Java код. За решавање овог проблема користе се *Jinja2* филтери који служе за претварање нечитљивог објекта у читљив стринг (низ знакова) који се може даље лако користити за генерисање Java кода. Пример филтера у *Jinja2* шаблону је `{{feature | beliefs_to_str}}`, где `feature` представља Python објекат који се прослеђује `beliefs_to_str` филтеру који има особине функције која након обраде у Python генератору враћа одговарајући читљив стринг. Као што се може видети у овом примеру, филтери се од објеката раздвајају симболом `'|'`. У *Jinja2* постоје и кориснички и уграђени филтери при чему *Jinja2* садржи велики број уграђених филтера. Као и функције у стандардним програмским језицима, филтери такође могу имати аргументе. Већина филтера из *Jinja2* шаблона је имплементирана у ALAS генератору (у листингу 11 приказана је делимична имплементација `imports` и `body` филтера). У наставку је описано неколико корисничких филтера који се користе за конверзију ALAS кода мобилног агента у еквивалентан Java код.

На почетку шаблона се проверава да ли је у ALAS коду наведен одговарајући пакет, а ако јесте, применом уграђеног *Jinja2* филтера `join` креираће се пакет по стандардној Java синтакси.

Филтер `imports` (листинг 10, линија 4) служи за импортовање свих потребних класа у Java изворни код. Један део имплементације `imports`

филтера је приказан у листингу 11, а у листингу 12 (линије 3 и 4) се могу видети неке класе импортоване применом овог филтера.

Након пакета и импорта генерише се класа чије је име исто као и име агента. Такође, уз класу се генеришу одговарајуће анотације у зависности како је агент дефинисан. Ако се стање агента чува, тј. уз њега је наведена кључна реч `'stateful'` генерисаће се `@Stateful` анотација. Ако није наведена ни једна кључна реч за стање агента (или је наведена кључна реч `'stateless'`), аутоматски ће се генерисати `@Stateless` анотација. Ако је дефинисана кључна реч `'domain'` генерисаће се `@Domain` анотација са одговарајућим параметром. Анотација `@Remote(Agent.class)` се обавезно генерише без обзира како је агент дефинисан.

Сви конструкти дефинисани у оквиру агента биће мапирани у одговарајуће компоненте у телу класе (промењљиве, методе, итд.). На почетку ће се генерисати глобалне промењљиве применом филтера `global_variable`. Ако глобалне промењљиве имају неке вредности, те вредности ће се генерисати применом филтера `assignment`.

ALAS конструкти `init` и `function` ће се мапирати у одговарајуће приватне методе. Садржај свих `init` конструката ће се мапирати у Java код у оквиру једне методе `onInit`, као што је приказано у листингу 12 (линија 12).

Обавезне методе које ће се аутоматски, без обзира на дефиницију агента, генерисати у Javi су:

```
protected void onMessage(ACLMessage msg)
protected void onInit(AgentInitArgs args) и
protected void doTasks(AgentInitArgs args).
```

Поред `onMessage` методе, у Javi ће се аутоматски генерисати још две методе. У оквиру `onMessage` методе генеришу се поједини конструкти у виду одређених услова применом `switch` или `if/else` наредби. Основни конструкти који се генеришу у оквиру `onMessage` методе су `action`, `arrived` и `move` при чему се целокупан Java код генерише применом филтера `onMessage_conditions` (листинг 10, линија 39).

На почетку `onMessage` методе ће се генерисати садржај свих `action` конструката код којих није наведен одговарајући предуслов применом `@PreCondition` анотације. Тако се садржај из листинга 8, линија 17, конвертује у Java изворни код у листингу 12, линија 16. Уколико `action` конструкти имају одговарајуће анотације, њихов садржај ће се такође трансформисати у Java код али у оквиру одговарајућег `if` или `switch` услова. На пример, ALAS код:

```
1. @PreCondition(REQUEST)
2. action AgentAction(ACLMessage msg) {
3.    // ...
4. }
```

ће се трансформисати у следећи Java код (у оквиру `onMessage` методе):

```
1. switch (msg.performative) {
2.    case REQUEST:
3.       // ...
4. }
```

Садржај свих `arrived` конструката се мапира у Java код у оквиру једне `if` наредбе са следећим условом: `if(msg.performative == Performative.RESUME)`. На пример, ALAS код агента из листинга 8, линија 6, мапира се у Java код у листингу 12, линија 33).

Конструкт `move` се конвертује у позив методе `move` која може имати један параметар.

Најсложенији конструкт који се састоји из великог броја мање сложених конструката је `Body`. Овај конструкт је мапиран у Java код применом `body` филтера који садржи више аргумената као што се види у листингу 10 (линија 29). У оквиру овог конструкта су дефинисане компоненте за контролу тока програма, променљиве, конструкти за креирање порука за међуагентску комуникацију и све остале компоненте дефинисане у граматици језика у оквиру `Statements` правила.

Листинг 11. Део Python генератора за Java код

```
 1. def imports(self):
 2.   if self.domain == 'domain':
 3.      imports += '\nimport dnars.siebog.annotations.Domain;'
 4.      for feature in self.features:
 5.         if feature.__class__.__name__ == 'DnarsBeliefsAnnotation':
 6.            imports += '\nimport dnars.siebog.annotations.Beliefs;'
 7.   #...
 8. def body(self, defined_variables_pairs, all_variables_pairs,
 9.           func_information):
10.   initialize_variables(self, defined_variables_pairs,
11.           all_variables_pairs, func_information)
12.   return statements(self.s)
13.
14. jinja_env = jinja2.Environment(
15.              Loader = jinja2.FileSystemLoader(this_folder),
16.              trim_blocks = True,
17.              lstrip_blocks = True)
18. jinja_env.filters['imports'] = imports
19. jinja_env.filters['beliefs_to_str'] = beliefs_to_str
20. template = jinja_env.get_template('java.template')
```

Листинг 12 садржи део генерисаног Java кода (на основу изворног кода ALAS агента из листинга 8) добијеног применом ALAS генератора и шаблона за Java језик.

Листинг 12. Део генерисаног изворног Java кода за мобилног агента

```
 1. package siebog.agents;
 2.
 3. import dnars.siebog.annotations.Domain;
 4. import dnars.siebog.annotations.Beliefs;
 5. //...
 6. @Stateless
 7. @Remote(Agent.class)
 8. public class MobileAgent extends XjafAgent {
 9.   int num = 0;
10.
11.   @Override
12.   protected void onInit(AgentInitArgs args){
13.   }
14.
15.   @Override
16.   protected void onMessage(ACLMessage msg) {
17.      String content = msg.content;
18.      if(this.isServer == true){
19.         System.out.println("I'm on the server and I received the
20.                        following message: " +msg);
21.         if(content == "move"){
22.            this.move();
23.         }else{
24.            this.move(msg.content);
25.         }
26.      }else{
27.         LoggerUtil.log("A'm in a client browser, and I have
28.                     received the following message: " +msg);
29.         if(content == "move"){
30.            this.move();
31.         }
32.      }
33.      if (msg.performative == Performative.RESUME) {
34.         this.num++;
35.         if(isServer == true){
36.            System.out.println("I'm at server,
37.                        and the number is: " +this.num);
38.            LoggerUtil.log("I'm at server,
39.                        and the number is: " +this.num);
40.         }else{
41.            LoggerUtil.log("I just came from to my browser,
42.                        and the number is " +this.num);
43.         }
44.      }
45.   }
46.
47.   @Override
48.   protected void doTasks(AgentInitArgs args) {
49.   }
50. }
```

7.4.1.1 Конверзија ALAS кода са подршком за DNARS у Javu

У претходном поглављу описан је процес конверзије ALAS кода у Javu. У оквиру овог поглавља биће приказан пример конверзије ALAS кода са подршком за дистрибуирано не-аксиоматско резоновање у Javu а укратко ће бити описани и неки филтери тј. функције које су коришћене за генерисање. У поглављу 7.1.1 описан је део граматике ALAS-а који омогућује подршку за DNARS а у листингу 9 дат је конкретан пример агента за DNARS чији ће Java еквивалент бити приказан у листингу 15. У листингу 13 приказан је део *Jinja2* шаблона који садржи филтере који омогућују конверзију кода DNARS агената у Javu.

Листинг 13. Део *Jinja2* шаблона за Javu

```
{% if agent.domain != None %}
  @Domain(name = "{{agent.domain.name}}")
{% endif %}
public class {{agent.name}} extends XjafAgent {
  @Override
  protected void doTasks(AgentInitArgs args) {
  {% for feature in agent.features %}
     {% if feature.__class__.__name__ == 'DnarsQuestion' %}
        List<Term> {{feature.answer}} = {{feature|question_to_str}}
     {% elif feature.__class__.__name__ == 'DnarsInference' %}
        List<Statement> {{feature.answer}} =
              {{feature|inference_to_str}}
     {% endif %}
  {% endfor %}
}

{% for feature in agent.features %}
  {% if feature.__class__.__name__ == 'DnarsBeliefs' %}
     @Beliefs
     public String[] {{feature|beliefs_to_str}}
  {% elif feature.__class__.__name__ == 'DnarsAddedUpdated' %}
     {{feature|belief_Added_Updated}}{
        {{feature|body(defined_variables_pairs|one_dimensional_list
              (defined_variables_pairs),
                 all_variables_pairs|one_dimensional_list
                    (all_variables_pairs), func_info)}}
     }
  {% endif %}
{% endfor %}
}
```

Листинг 14. Пример дела имплементације једног *Jinja2* филтера (`beliefs_to_str`) у оквиру ALAS генератора

```
def beliefs_to_str(self):
  global count
  count1 = 0
  temp = 'initialBeliefs'
```

```
 5.   if count == 0: temp += ''
 6.   else: temp += str(count)
 7.   temp += '() {' + '\n\t\treturn new String[] {'
 8.   for statement in self.statements:
 9.      if count1 > 0: temp += ', '
10.      temp += '\n\t\t\t"' + judgement(statement.judgement)
11.      temp += ' (' + str(statement.truth.number[0]) + ', ' +
12.         str(statement.truth.number[1]) + ')"'
13.      count1 += 1
14.   count += 1
15.   return temp + '\n\t\t};' + '\n\t}'
```

У листингу 15 је приказан део Java кода који је генерисан на основу ALAS агента приказаног у листингу 9.

Листинг 15. Део генерисаног изворног Java кода за агента са подршком за DNARS

```
 1. import siebog.agentmanager.Agent;
 2. import dnars.siebog.annotations.Domain;
 3. //...
 4. @Stateless
 5. @Remote(Agent.class)
 6. @Domain(name = "dnarsdomain")
 7. public class DnarsTestAgent extends XjafAgent {
 8.   private static final long serialVersionUID = 1L;
 9.
10.   @Override
11.   protected void onInit(AgentInitArgs args){
12.   }
13.   @Override
14.   protected void onMessage(ACLMessage msg) {
15.   }
16.   @Override
17.   protected void doTasks(AgentInitArgs args) {
18.      List<Term> t1 = Arrays.asList(graph().answer
19.         (StatementParser.apply(
20.            "(\\ datum_rodjenja Ernest_Hemingvej *) -> ?"), 1));
21.      LOG.info("The best answer: " +t1.get(0));
22.
23.      List<Term> t2 = Arrays.asList(graph().answer
24.         (StatementParser.apply("? -> voće"), 3));
25.      LOG.info("The best answers: ");
26.      for(Term term: t2)
27.         LOG.info("" +term);
28.
29.      List<Statement> s1 = Arrays.asList(graph().backwardInference
30.         (StatementParser.apply("pčela -> insekt (1.0, 0.9)"), 1));
31.      LOG.info("Inference: " +s1.get(0));
32.
33.      List<Statement> s2 = Arrays.asList(graph().backwardInference
34.         (StatementParser.apply("jabuka -> voće (1.0, 0.9)"), 2));
35.      LOG.info("Inferences: ");
```

```
36.      for(Statement statement: s2)
37.         LOG.info("" +statement);
38.   }
39.
40.   @Beliefs
41.   public String[] initialBeliefs() {
42.      return new String[] {
43.         "mandarina -> voće (0.9, 0.7)",
44.         "narandža -> voće (0.9, 0.8)",
45.         "jabuka -> voće (0.8, 0.7)",
46.         "pčela -> insekt (1.0, 0.9)",
47.         "citrus ~ limun (0.76, 0.83)",
48.         "(x pčela med) -> pravi (1.0, 0.9)",
49.         "(x Ernest_Hemingvej 1899) -> datum_rođenja (1.0, 0.9)",
50.         "jede -> (x zec šargarepa) (1.0, 0.9)"
51.      };
52.   }
53.
54.   @BeliefAdded()
55.   public void beliefAdded(List<Statement> beliefs){
56.      LoggerUtil.log("Sva nova verovanja: " +beliefs);
57.   }
58.
59.   @BeliefAdded(subj="banana", copula="->", pred="voće")
60.   public void beliefAdded1(List<Statement> beliefs){
61.      LoggerUtil.log("Banana je vrsta voća: " +beliefs);
62.   }
63.
64.   @BeliefUpdated()
65.   public void beliefUpdated(List<Statement> beliefs){
66.      LoggerUtil.log("Sva izmenjena verovanja: " +beliefs);
67.   }
68.
69.   @BeliefUpdated(subj="(x pčela med)", copula="->", pred="pravi")
70.   public void beliefUpdated1(List<Statement> beliefs){
71.      LoggerUtil.log("Izmenjeno verovanje: " +beliefs);
72.   }
73. }
```

Иницијално знање агента тј. иницијална веровања као и ALAS конструкти који се користе за обавештавање агента о насталим променама у бази знања се мапирају у јавне методе у Javi као што се може видети у листингу 15 (линије 40-72). Сви ALAS конструкти који агенте чине интелигентним и омогућују извођење закључака се мапирају у листе у које се смештају одговарајући резултати закључивања. Све листе се генеришу у оквиру приватне методе `doTasks`.

Филтер `beliefs_to_str` који је претходно поменут у поглављу у којем је описана граматика језика и чија се делимична имплементација може видети у листингу 14, служи за мапирање ALAS `beliefs` конструкта у јавну методу у Java језику чија је повратна вредност низ иницијалних веровања наведених у

оквиру `beliefs` конструкта у ALAS коду (листинг 9). Приликом мапирања, метода се у Java језику означава `@Beliefs` анотацијом.

`belief_added_updated` филтер служи за конверзију `DnarsAddedUpdated` објекта из модела ALAS језика у читљив стринг. Овај објекат ће се у ALAS генератору Java кода трансформисати у јавне методе у Java језику са `@BeliefAdded` или `@BeliefUpdated` анотацијама у зависности да ли је позван `beliefadded` или `beliefupdated` ALAS конструкт.

`question_to_str` филтер конвертује `DnarsQuestion` објекат у читљив стринг. Овај филтер је заснован на `question` ALAS конструкту који као аргумент прослеђује питање на које DNARS механизам за одлучивање треба да да одговор. Одговор се добија позивом DNARS функције `answer` која враћа одговор у виду атомских или сложених термова на основу веровања у бази знања. На пример, `question` ALAS конструкт:

```
Terms t = question
   ((\\ datum_rođenja Ernest_Hemingvej *) -> ?);
```

ће се мапирати у следећи Java код:

```
List<Term> t = Arrays.asList(graph().answer(StatementParser
   .apply("(\ datum_rođenja Ernest_Hemingway *) -> ?"), 1));
```

Поред `question` конструкта, најзначајнији ALAS конструкт за подршку DNARS-у је `inference`, који служи за доношење закључака на основу неконзистентног или недовољног знања. `inference_to_str` филтер мапира код ALAS `inference` конструкта у Java код тј. у позив DNARS функције `backwardInference` која на основу DNARS правила извођења враћа одговарајуће закључке у форми NAL реченица. Пример Java кода добијеног из `inference` конструкта:

```
Statements s = inference(pčela -> insekt);
```

је:

```
List<Statement> s = Arrays.asList(graph().backwardInference
   (StatementParser.apply("pčela -> insekt (1.0, 0.9)"), 1));
```

7.4.2 Конверзија ALAS кода у JavaScript

JavaScript шаблон је сличан шаблону за Javu, пре свега у функционалном смислу, основна разлика је у деловима кода који се односе на синтаксу језика. Свакако, главни механизам који служи за генерисање синтаксно и семантички исправних Java и JavaScript изворних програма се налази у ALAS генератору кода.

Део *Jinja2* шаблона за JavaScript је приказан у листингу 16. На почетку сваког JavaScript кода се генерише код из шаблона (линија 1). Применом `if` и `for` наредбе укључују се и додатни фајлови (скрипте) ако су наведени приликом писања агента. За разлику од Jave, на почетку се не генерише име пакета.

Након укључивања фајлова, генерише се конструкторска функција чије је име исто као и име агента. У оквиру ове функције генеришу се сва стања и глобалне промењиве.

Конструкти као што су `init` и `arrived` мапирају се у прототипске методе које заједно са конструкторском функцијом чине прототип објекта а пример JavaScript синтаксе за ове конструкте се може видети у листингу 17 (линије 7 и 11).

Конструкти `action` и `move` се трансформишу у сличан код као у Javi, само је синтакса прилагођена JavaScript језику. Садржај ових конструката се такође генерише у оквиру `onMessage` методе која је већ описана у претходном поглављу.

Линија 43 из листинга 17 се аутоматски генерише и она представља доделу новог `Agent` објекта постојећем прототипу објекта `MobileAgent` који на основу тога аутоматски приступа ланцу прототипског наслеђивања.

Линија 45 из листинга 17 се такође аутоматски генерише и служи за повезивање агента на своју веб воркер (енг. web worker) скрипту.

Листинг 16. Део *Jinja2* шаблона за JavaScript

```
1. importScripts("/siebog-war/js/radigost/agent.js");
2.
3. {% if agent.importScripts != None %}
4.   {% for import in agent.importScripts %}
5.     importScripts("{{import.script}}");
6.   {% endfor %}
7. {% endif %}
8.
9. function {{agent.name}}() {
10.   {% for feature in agent.features %}
11.     {% if feature.__class__.__name__ == 'Assignment' %}
12.       {% if defined_variables_pairs.append
13.             (feature|defined_global_var(all_variables_pairs)) %}
14.       {% endif %}
15.       {{feature|assignment}}
16.     {% endif %}
17.   {% endfor %}
18. };
19. {{agent.name}}.prototype = new Agent();
20. {{agent.name}}.prototype.onArrived = function(host, isServer) {
21.   {% for feature in agent.features %}
22.     {% if feature.__class__.__name__ == 'Arrived' %}
23.       {{feature|body(defined_variables_pairs|one_dimensional_list
24.             (defined_variables_pairs),
25.             all_variables_pairs|one_dimensional_list
26.             (all_variables_pairs), func_info)}}
27.     {% endif %}
28.   {% endfor %}
29. };
30. setAgentInstance(new MobileAgent());
```

Листинг 17. Део генерисаног изворног JavaScript кода за мобилног агента

```
importScripts("/siebog-war/js/radigost/agent.js");

function MobileAgent() {
  this.num = 0;
};

MobileAgent.prototype.onInit =
        function(args, _post, _log, _moveToServer) {
};

MobileAgent.prototype.onArrived = function(host, isServer) {
  this.num++;
  if(isServer === true){
     print("I'm at server, and the number is: " +this.num);
     console.log("I'm at server, and the number is: " +this.num);
  }else{
     console.log("I just came from to my browser, and the number
                  is " +this.num);
  }
};

MobileAgent.prototype.onMessage = function(msg) {
  if (typeof msg == "string") {
     msg = JSON.parse(msg);
  }
  var content = msg.content;
  if(this.isServer === true){
     print("I'm on the server and I received the following
              message: " +msg);
     if(content === "move"){
        this.moveToClient();
     }else{
        this.moveToServer(msg.content);
     }
  }else{
     console.log("A'm in a client browser, and I have received the
                  following message: " +msg);
     if(content === "move"){
        this.moveToServer();
     }
  }

MobileAgent.prototype = new Agent();

setAgentInstance(new MobileAgent());
```

7.5 Валидација кода

textX нуди добру подршку кад је у питању провера исправности написаног кода. Међутим, поједине компоненте ипак не могу бити проверене за време парсирања кода. Пошто је ALAS настао како би се подржала мобилност софтверских агената и могућност извршавања на независним платформама имплементираним у различитим програмским језицима, подразумева се да се код агента написан у ALAS језику мора конвертовати у језик одредишне платформе. Самим тим, могуће је додатно побољшати проверу валидности написаног кода у оквиру ALAS генератора кода.

ALAS генератор кода је добро оптимизован и развијен тако да проверава исправност, тј. валидацију (укључујући и синтаксу и семантику) написаног кода. С обзиром да је ALAS заснован на Java програмском језику, поруке о свим грешкама су формиране на основу порука у Javi. ALAS генератор је развијен тако да се при генерисању одговарајућег изворног кода врше разне провере при чему су неке наведене у наставку.

Променљиве – приликом навођења неке променљиве у коду, проверава се да ли је она претходно дефинисана (и иницијализована), води се рачуна о опсегу важења променљивих. textX помоћу опције повезаног референцирања може да омогући проверу да ли је променљива декларисана, али је прилично компликовано креирати граматику тако да се врши и провера дефинисаности, типа и опсега важења променљиве. Због тога су ове провере имплементиране у оквиру ALAS генератора. Најкомпликованија имплементација валидности се односи на опсег важења променљивих. У листингу 18 дат је део кода који проверава да ли је променљива дефинисана и ако није, генератор ће избацити грешку и одговарајућу поруку: "*Exception: The variable 'var_name' may not have been initialized!*".

Листинг 18. Провера дефинисаности променљиве

```
1. try:
2.    if var not in definedvariables:
3.       raise Exception('(' + var + ')')
4. except Exception as e:
5.    print('Exception: The variable', e.args[0],
6.             'may not have been initialized!\n')
7.    raise
```

Једна од најсложенијих имплементација везаних за валидацију кода односи се на функције. Синтакса се проверава приликом парсирања изворног кода агента а у оквиру ALAS генератора се проверава валидност функције приликом њеног позива. Прво се проверава име функције тј. да ли је функција са тим именом дефинисана. Након тога, проверавају се параметри (ако постоје), њихов број и типови. У листингу 19 дат је део кода који служи за приказ одговарајуће поруке

уколико се утврди да позвана функција има погрешне параметре (нпр. тип неког параметра није исти као у дефиницији функције).

Листинг 19. Провера валидности позване функције

```
1. try:
2.    if not valid_param:
3.       raise Exception('('+ function_name +')')
4.    except Exception as e:
5.       print('Exception: The method',e.args[0],
6.                   'is not applicable for its arguments\n')
7.       raise
```

ALAS генератор проверава и креиране инстанце тј. објекте одговарајућих класа. Проверава се име објекта па ако објекат са тим именом већ постоји биће генерисана грешка и следећа порука: "*Error: Duplicate object name 'obj_name'*". Такође, ако је неки објекат наведен негде у коду а није претходно дефинисан, генератор ће избацити грешку: "*Error: 'obj_name' is undefined*". Поред имена објекта, проверава се и тип, тј. име одговарајуће класе.

У листингу 20 дат је део кода који служи за проверу типа аргумента акције чија је дефиниција детаљније описана у поглављу 7.1.

Листинг 20. Провера типа аргумента акције

```
1. try:
2.    if feature.action.param.type != 'ACLMessage':
3.       raise Exception('('+feature.action.param.type+')')
4.    except Exception as e:
5.       print ('The Action parameter type',e.args[0],
6.                'is not valid!\nExcepted "ACLMessage".')
7.       raise
```

Комплетан изворни код ALAS генератора у оквиру кога су имплементиране све провере, може се видети на GitHub-у[16].

8 Студија случаја

Сврха овог поглавља је да представи студију случаја која ће да прикаже неке од могућности агентског, домен-оријентисаног језика ALAS.

Конкретно, у овој студији случаја ће бити описана два ALAS агента који користе реалну DBpedia базу знања како би извели оптималне закључке на постављена питања из реалног света.

Пројекат DBpedia се заснива на преузимању знања из Wikipedie и чини то знање широко доступним применом успостављених стандарда семантичког веба. Циљ семантичког веба је да сви подаци који се налазе на њему буду разумљиви рачунару. Технологије семантичког веба могу се користити у апликацијама разних намена, на пример у повезивању података, где подаци с различитих локација и различитих формата могу бити повезани у једну целину. Семантички веб је без сумње једна од најбитнијих ставки када је у питању развој интелигентних агената.

Семантички веб користи онтологије, тј. машински читљив речник који је довољно прецизан да би се креирале логичке везе између појединих ресурса. Онтологије дефинишу концепте и релације које се користе да се представи подручје знања. Онтологија у семантичком вебу помаже при интеграцији података, нпр. када у изразима постоје нејасноће коришћене у различитим скуповима података, или када се с врло мало знања могу открити нове везе међу њима.

Онтологије су рачунарски употребљиве дефиниције основних појмова у одређеном домену и односима међу њима. OWL (енг. Web Ontology Language) је онтологијски језик написан за семантички веб са формалним значењима односно да би се знање на вебу приказивало у облику онтологија. OWL онтологија пружа на располагање класе, својства, ентитете и податке који се чувају као семантички веб документи. Формат записа OWL документа заснива се на RDF (енг. Resource

Description Framework) формату[17]. RDF је језик за представљање информација о веб ресурсима и обезбеђује интероперабилност међу апликацијама које размењују информације разумљиве рачунарима.

DBpedia је пројекат који се заснива на структурираним реченицама изведеним из једне од највећих слободних база података Wikipedia која се редовно ажурира. Циљ пројекта је поједноставити огромне количине података тако да буду разумљивији, приступачнији и једноставнији за обраду од стране рачунара. DBpedia нуди различите скупове података[18] у којима су подаци структурирани на разне начине и на различитим језицима. DBpedia садржи неколико милијарди уређених RDF тројки које садрже податке о људима, државама, градовима, компанијама, филмовима, књигама, рачунарским играма итд.

DBpedia садржи велики број различитих домена у виду скупова података у форми RDF (2017) реченица. Реченице су уређене тројке у форми `<субјекат> <предикат> <објекат>`. На овај начин је описана веза између два ресурса (Schreiber i Raimond 2017). Скуп RDF реченица може се приказати у облику означеног усмереног графа што доводи до бољег приказивања информација у односу на друге релацијске моделе. Често се приликом структурирања текста јављају проблеми при чему је један од најчешћих проблема појава речи које се исто пишу али имају потпуно различито значење. Да би се избегао тај проблем, за представљање ресурса користи се већ постојеће решење у виду јединственог представљања ресурса – URI (енг. Uniform Resource Identifier). Све RDF реченице су представљене одговарајућим графом при чему субјекат и објекат представљају чворове а предикати гране графа. Уколико неки URI не садржи конкретну информацију о неком ресурсу, чвор предвиђен за смештање тог URI-а ће остати неименован (енг. blanc node). Гране не могу бити неименоване и обавезно садрже одговарајући URI. Међутим, неки ресурси се не морају представити преко URI-а пошто већ имају јединствену вредност. Они се зову *литерали* и представљају се низом знакова (бројева), нпр. неке особине човека као што су висина, тежина, професија којом се бави, датуми рођења итд. У RDF формату, субјекат може бити URI или неименован ресурс, предикат мора бити URI а објекат може бити или неименован ресурс или URI или *литерал*. У следећем примеру, субјекат и предикат су URI а објекат је *литерал* (датум рођења):

```
<http://dbpedia.org/resource/Ernest_Hemingway>
   <http://dbpedia.org/ontology/birthDate> 1899-7-21
```
(8.1)

[17] https://www.w3.org/RDF
[18] https://wiki.dbpedia.org/develop/datasets

У листингу 21 приказано је неколико RDF реченица из једног DBpedia-иног скупа података који садржи преко 30 милиона уређених RDF тројки. Знање се односи на писца Ернеста Хемингвеја.

Листинг 21. RDF реченице из DBpedia базе знања

```
<http://dbpedia.org/resource/Ernest_Hemingway>
  <http://dbpedia.org/ontology/birthDate> "1899-07-21" .

http://dbpedia.org/resource/Ernest_Hemingway
  <http://dbpedia.org/ontology/birthPlace>
     <http://dbpedia.org/resource/Oak_Park,_Illinois> .

<http://dbpedia.org/resource/Ernest_Hemingway>
  <http://dbpedia.org/ontology/child>
     <http://dbpedia.org/resource/Gregory_Hemingway> .

<http://dbpedia.org/resource/Ernest_Hemingway>
  <http://dbpedia.org/ontology/child>
     <http://dbpedia.org/resource/Jack_Hemingway> .

<http://dbpedia.org/resource/Ernest_Hemingway>
  <http://dbpedia.org/ontology/child>
     <http://dbpedia.org/resource/Patrick_Hemingway> .

<http://dbpedia.org/resource/The_Old_Man_and_the_Sea>
  <http://dbpedia.org/ontology/author>
     <http://dbpedia.org/resource/Ernest_Hemingway> .

<http://dbpedia.org/resource/Green_Hills_of_Africa>
  <http://dbpedia.org/ontology/author>
     <http://dbpedia.org/resource/Ernest_Hemingway> .
```

Као што се може видети, NAL и RDF имају сличне формате реченица (*субјекат - предикат - објекат*) што значи да се лако могу прилагодити један другом. Да бисмо креирали базу знања за DNARS и креирали знање које агенти могу да користе, потребно је трансформисати RDF формат `<субјекат> <предикат> <објекат>` у NAL формат `(x субјекат објекат) -> предикат`. На основу ове конверзије, RDF реченица 8.1 ће се трансформисати у следећи NAL формат:

```
(x http://dbpedia.org/resource/Ernest_Hemingway 1899-7-21)
   -> http://dbpedia.org/ontology/birthDate (1.0, 0.9)
```
(8.2)

Пошто се подразумева да реченице у Dbpedia бази знања имају високу истинитосну вредност, приликом генерисања ће се NAL реченицама аутоматски доделити подразумевана истинитосна вредност (1.0, 0.9).

У листингу 22 је приказан пример ALAS агента који има задатак да одговори на неколико питања. У наставку ће због прегледности у оквиру веровања уместо целе путање бити наведен само назив ресурса (онтологије) на који се та путања односи. Одговарајући Java код овог агента је приказан у листингу 23.

Листинг 22. Пример ALAS агента

```
 1. domain(Hemingway) agent TestAgent {
 2.   beliefs {
 3.      (x Ernest_Hemingway 1899-07-21)
 4.                                -> birthDate (1.00, 0.90),
 5.      (x Ernest_Hemingway Oak_Park,_Illinois)
 6.                                -> birthPlace (1.00, 0.90),
 7.      (x Ernest_Hemingway Gregory _Hemingway)
 8.                                -> child (1.00, 0.90),
 9.      (x Ernest_Hemingway Jack_Hemingway)
10.                                -> child (1.00, 0.90),
11.      (x Ernest_Hemingway Patrick_Hemingway)
12.                                -> child (1.00, 0.90),
13.      (x The_Old_Man_and_the_Sea Ernest_Hemingway)
14.                                -> author (1.00, 0.90),
15.      (x Green_Hills_of_Africa Ernest_Hemingway)
16.                                -> author (1.00, 0.90)
17.   }
18.
19.   Terms t1 = question(? -> (/ birthDate Ernest_Hemingway *));
20.   Terms t2 = question(? -> (/ child Ernest_Hemingway *), 3);
21.   Statements s = inference
22.      ((x The_Old_Man_and_the_Sea Ernest_Hemingway) -> author);
23. }
```

Листинг 23. Део генерисаног изворног Java кода за агента из листинга 22

```
 1. @Stateless
 2. @Remote(Agent.class)
 3. @Domain(name = "Hemingway")
 4. public class TestAgent extends XjafAgent {
 5.   private static final long serialVersionUID = 1L;
 6.
 7.   @Override
 8.   protected void onInit(AgentInitArgs args){
 9.   }
10.   @Override
11.   protected void onMessage(ACLMessage msg) {
12.   }
13.   @Override
14.   protected void doTasks(AgentInitArgs args) {
15.      List<Term> t1 = Arrays.asList(graph().answer
16.         (StatementParser.apply(
17.            "(\\ birthDate Ernest_Hemingway *) -> ?"), 1));
18.      LOG.info("The best answer: " +t1.get(0));
19.
20.      List<Term> t2 = Arrays.asList(graph().answer
21.         (StatementParser.apply(
22.            "(\\ child Ernest_Hemingway *) -> ?"), 3));
23.      LOG.info("The best answers: ");
24.      for(Term term: t2)
25.         LOG.info("" +term);
26.
```

```
27.      List<Statement> s = Arrays.asList(graph().backwardInference
28.         (StatementParser.apply(
29.            "(x The_Old_Man_and_the_Sea Ernest_Hemingway)
30.                              -> author"), 1));
31.      LOG.info("Inference: " +s1.get(0));
32.   }
33.
34.   @Beliefs
35.   public String[] initialBeliefs() {
36.      return new String[] {
37.         "(x Ernest_Hemingway 1899-07-21)
38.                               -> birthDate (1.00, 0.90)",
39.         "(x Ernest_Hemingway Oak_Park,_Illinois)
40.                               -> birthPlace (1.00, 0.90)",
41.         "(x Ernest_Hemingway Gregory _Hemingway)
42.                               -> child (1.00, 0.90)",
43.         "(x Ernest_Hemingway Jack_Hemingway)
44.                               -> child (1.00, 0.90)",
45.         "(x Ernest_Hemingway Patrick_Hemingway)
46.                               -> child (1.00, 0.90)",
47.         "(x The_Old_Man_and_the_Sea Ernest_Hemingway)
48.                               -> author (1.00, 0.90)",
49.         "(x Green_Hills_of_Africa Ernest_Hemingway)
50.                               -> author (1.00, 0.90)"
51.      };
52.   }
```

Из листинга 22 (линија 19) се види да се за добијање одговора на питање '*када је рођен Ернест Хемингвеј?*' користи ALAS конструкт `question` у следећем облику:

```
Terms t = question(? -> (/ birthDate Ernest_Hemingway *))
```
(8.3)

На основу NAL реченице о датуму рођења Ернеста Хемингвеја, агент не може закључити када је он рођен. Због тога је потребно изменити структуру реченице 8.2. То се може извести применом правила (3) које је дефинисано у поглављу 3.1. Након структурне трансформације, реченица 8.2 добија још две слике:

```
Ernest_Hemingway -> (/birthDate * 1899-7-21) (1.0, 0.9)
```
(8.4)

и

```
1899-7-21 -> (/birthDate Ernest_Hemingway *) (1.0, 0.9)
```
(8.5)

Сада је јасно да друга реченица (8.5) има исту структуру као и упитна реченица у оквиру `question` конструкта из листинга 22, па сада DNARS механизам за одлучивање на основу генерисаног Java кода (листинг 23, линија 18) даје одговор: `The best answer:1899-7-21`.

Други задатак је да агент пронађе имена деце Ернеста Хемингвеја. Питање је постављено такође применом конструкта `question` (листинг 22, линија 20). Реченице из базе знања дате у листингу 22 у оквиру конструкта `beliefs`, DNARS механизам ће реструктурирати како би се добиле нове слике на основу

којих се могу извести одговарајући закључци, у овом случају имена деце. Након реструктурирања, на основу следећих реченица:

```
Gregory_Hemingway -> (/child Ernest_Hemingway *) (1.00, 0.90)
Jack_Hemingway -> (/child Ernest_Hemingway *) (1.00, 0.90)
Patrick_Hemingway -> (/child Ernest_Hemingway *) (1.00, 0.90)
```

агент ће лако закључити ко су деца Ернеста Хемингвеја.

У листингу 22, линија 21, наведен је конструкт `inference` који служи за проверу да ли је нека реченица истинита и која је њена истинитосна вредност. У овом случају агент треба да провери да ли је Ернест Хемингвеј написао роман *Старац и море* (`The_Old_Man_and_the_Sea`). С обзиром да одговарајућа реченица постоји у бази знања, агент ће је навести заједно са истинитосном вредношћу тако да ће резултат на основу генерисаног Java кода бити:

```
Inference: (x The_Old_Man_and_the_Sea Ernest_Hemingway)
                        -> author (1.00, 0.90)
```

Када је у питању агент из листинга 24, задатак је да се дâ одговарајући опис везан за Ернеста Хемингвеја (занимање). На основу три веровања у бази агента (као што је приказано у листингу 24, линија 2), DNARS механизам ће прво покушати да пронађе реченицу са истом структуром као упитна реченица (листинг 24, линија 9). Пошто таква реченица не постоји, механизам креира све могуће слике постојећих реченица и смешта их у формирани граф особина. Креиране слике и одговарајуће реченице из базе знања агента су приказане у листингу 25.

Листинг 24. Пример ALAS агента

```
1. domain(Hemingway) agent TestAgent {
2.    beliefs {
3.       (x Ivo_Andric Nobel_Prize_in_Literature)
4.            -> award(1.00, 0.90)
5.       (x Ernest_Hemingway Nobel_Prize_in_Literature)
6.            -> award (1.00, 0.90)
7.       (x Ivo_Andric Writer) -> description (1.00, 0.90)
8.    }
9.    Terms t = question(? -> (/ description Ernest_Hemingway *));
```

Листинг 25. Све слике постојећих реченица из базе знања агента из листинга 24

```
 1. (x Ivo_Andric Nobel_Prize_in_Literature) -> award (1.00, 0.90)
 2. Ivo_Andric -> (/ award * Nobel_Prize_in_Literature) (1.00, 0.90)
 3. Nobel_Prize_in_Literature -> (/ award Ivo_Andric *) (1.00, 0.90)
 4.
 5. (x Ernest_Hemingway Nobel_Prize_in_Literature)
 6.        -> award (1.00, 0.90)
 7. Ernest_Hemingway
 8.        -> (/ award * Nobel_Prize_in_Literature) (1.00, 0.90)
 9. Nobel_Prize_in_Literature
10.        -> (/ award Ernest_Hemingway *) (1.00, 0.90)
```

```
11.
12. (x Ivo_Andric Writer) -> description (1.00, 0.90)
13. Ivo_Andric -> (/ description * Writer) (1.00, 0.90)
14. Writer -> (/ description Ivo_Andric *) (1.00, 0.90)
```

Међутим, ни једна структура креираних слика не одговара задатом упиту. Због тога, DNARS механизам примењује правила закључивања на све реченице (укључујући и слике) и на тај начин проширује граф особина са новим веровањима (закључцима). Применом правила поређења, механизам на основу реченица:

```
Ivo_Andric -> (/ award * Nobel_Prize_in_Literature) (1.00, 0.90)
```

и

```
Ernest_Hemingway -> (/ award * Nobel_Prize_in_Literature) (1.00, 0.90)
```

закључује да су Иво Андрић и Ернест Хемингвеј *слични* и формира нову реченицу:

```
Ivo_Andric ~ Ernest_Hemingway (1.00, 0.45)
```

Применом правила поређења на претходне две реченице, мења се истинитосна вредност новодобијене реченице. Истинитосна вредност, тј. учесталост и стабилност се рачунају на основу формула из табеле 3.3.
С обзиром да је релација сличности симетрична, механизам аутоматски креира и додаје у граф следећу реченицу:

```
Ernest_Hemingway ~ Ivo_Andric (1.00, 0.45)
```

Коначно, на основу ове реченице и реченице из листинга 25 (линија 13), механизам применом правила аналогије изводи нови закључак:

```
Ernest_Hemingway -> (/ description * Writer) (1.00, 0.41)
```

Реструктурирањем последње реченице (применом правила (3) из поглавља 3.1) добија се реченица која одговара структури постављеног питања на основу које систем лако закључује да је и Ернест Хемингвеј писац (Writer).

```
Writer -> (/ description Ernest_Hemingway *) (1.00, 0.41)
```

Овај пример показује да је DNARS веома погодан када систем нема довољно знања. Без обзира на оскудно знање (што се може закључити и на основу истинитосне вредности резултата), DNARS механизам је успео да донесе одговарајући закључак чија је сигурност (тачност) представљена у оквиру истинитосне вредности. С обзиром да је стабилност резултата мања од 0.5, може се закључити да не постоји довољно знања и сматра се да је добијени закључак несигуран.

9 Закључак

Интелигентни агенти играју важну улогу у софтверском инжењерству у погледу њиховог интензивног развоја у претраживању и обради информација, као и у сложеним софтверским производима, алатима и системима. Интензиван развој вештачке интелигенције у великој мери доприноси унапређењу карактеристика интелигентних агената. Међутим, да би се могућности агената максимално искористиле, потребно је омогућити њихову сарадњу и међусобну комуникацију. Развијени су мултиагентски системи који садрже групе агената који паралелно раде имитирајући понашање људских друштава како би брже и лакше решили сложене задатке.

Један од нових мултиагентских система који је настао применом савремених веб и пословних технологија је Siebog. Основна функционалност Siebog мултиагентске платформе је да обезбеди инфраструктуру за извршавање агената у веб окружењима али и да подржи мобилност и извршавање агената у дистрибуираним окружењима. Као што је описано у дисертацији, Siebog се састоји из две компоненте, клијентске – Radigost и серверске – XJAF. Интеграцијом ове две компоненте добијен је систем са оптималним могућностима и напредном подршком за извршавање интелигентних агената.

Како би агенти имали одговарајући степен интелигенције у Siebog је интегрисан дистрибуирани систем за не-аксиоматско резоновање. DNARS је архитектура за резоновање која је искоришћена како би агентима била омогућена успешна обрада и брзо доношење закључака над огромним базама знања које постају све веће. DNARS је заснован на концептима NAL формализама који се користе за управљање знањем и у оквиру DNARS-а имплементирана су четири нивоа NAL логике.

Главни недостаци Siebog платформе – недовољна интероперабилност и непостојање подршке за хетерогену мобилност агената, као и недостатак подршке за дистрибуирано не-аксиоматско резоновање довели су до потребе за развојем новог агентског, домен-оријентисаног језика који би омогућио развој интелигентних Siebog - NAL агената способних да се за потребе извршавања

одговарајућих задатака премештају са чвора на чвор у оквиру рачунарског кластера.

На почетку истраживања постављене су три хипотезе које су успешно потврђене на основу добијених резултата описаних у овој дисертацији.

Прва хипотеза:

Несметане миграције агената између клијент-сервер окружења имплементираних у различитим програмским језицима могу се обезбедити повећањем интероперабилности и обезбеђивањем хетерогене мобилности агената. Ово се може постићи развојем универзалног агентског, домен-оријентисаног језика и механизама који ће омогућити аутоматску конверзију агентског кода у имплементациони језик одредишне платформе; у раду је показано да је могуће развити такав језик који ће подржати све наведене захтеве. Скалабилност, отпорност на грешке, дељење кода, интероперабилност, вишеплатформска размена порука, хетерогена мобилност агената, расподела оптерећења, али и друге предности Siebog-а у односу на остале мултиагентске системе, захтевале су развој новог језика који ће подржати све особине Siebog-а пошто је прегледом актуелног стања у области агентских и домен-оријентисаних језика утврђено да ни један постојећи језик не може да подржи све претходно наведене захтеве. Представљен је прототип агентског, домен-оријентисаног језика ALAS а основни услови у виду обезбеђивања интероперабилности и хетерогене мобилности агената остварене су развојем таквог механизма који приликом пребацивања агента са једног чвора на други (при чему чворови могу бити имплементирани у различитим програмским језицима) омогућава аутоматску конверзију агентског кода у имплементациони језик одредишне платформе. Конверзија ALAS кода у друге програмске језике омогућује ALAS агентима да се успешно извршавају и на клијентској и на серверској страни Siebog-а. У дисертацији су описани имплементирани механизми који омогућавају успешну конверзију ALAS кода агената у програмске језике Java и JavaScript.

У раду је постављена и друга хипотеза:

Мултиагентско окружење Siebog је пројектовано тако да подржи развој интелигентних агената способних да доносе закључке и самостално решавају одговарајуће задатке. Такви агенти се могу развијати применом језика који ће моћи да подржи писање агената заснованих на дистрибуираном не-аксиоматском резоновању; која заједно са првом хипотезом обједињује све неопходне захтеве мултиагентског окружења Siebog. Ова хипотеза је потврђена на основу развијеног прототипа агентског, домен-оријентисаног језика ALAS који је дизајниран да подржи развој интелигентних агената заснованих на дистрибуираном не-аксиоматском резоновању што омогућава агентима извођење закључака над великим базама знања. NAL логика на којој је заснован DNARS омогућава агентима да успешно изводе закључке на основу неконзистентног или често недовољног, тј. несигурног знања.

Трећа хипотеза:

Интелигентни агенати у случају комплексних области функционисања се могу развијати и без помоћи експерата из одговарајућих домена примене. Ово се може постићи применом конструката специфичних за одговарајуће домене у којима се извршавају интелигентни агенти; је потврђена на основу развијеног ALAS језика као домен-оријентисаног, који подржава одговарајуће концепте који су уско везани за домене у којима се агенти извршавају а то је постигнуто применом одговарајућих програмских конструката. Применом ових специјализованих функција које садрже сву функционалну сложеност домена, програмерима је у великој мери олакшано писање агената пошто не морају да користе помоћ доменских експерата. На тај начин програмери се могу знатно више фокусирати на решавање конкретног проблема а притом је програмски код једноставнији и читљивији. ALAS је развијен применом textX алата који је својом флексибилношћу омогућио развој ефикасних конструката.

На основу претходно изнетих резултата може се истаћи да главни допринос дисертације представља развијен прототип агентског, домен-оријентисаног језика ALAS. У раду је показано да примена овог језика у области мултиагентских система може успешно решити проблем интероперабилности и хетерогене мобилности агената што је био један од основних циљева дисертације. Такође, подршка дистрибуираном систему за не-аксиоматско резоновање ће омогућити примену ALAS-а у окружењима заснованим на не-аксиоматској логици.

Могућност примене предложеног решења је велика. Развој нових технологија и све већа присутност вештачке интелигенције подразумевају примену интелигентних агената у скоро свим сферама људског живота. Језик предстаљен у овој дисертацији може наћи примену у писању агената у различитим доменима (мобилна телефонија, интелигентни системи за учење, веб окружења, рачунарске игре, разни системи за доношење одлука, финансије, банкарство, здравство, машинство, саобраћај итд.). Такође, описано решење може наћи ефикасну примену и у тренутно неким од најперспективнијих области као што су аутомобилска индустрија (аутономна вожња) и Интернет ствари (енг. Internet of Things - IoT). Све актуелнији, нови концепт који представља проширење Интернета ствари – Агенти ствари (енг. Agents of Things - AoT) је нова област која подразумева интеграцију интелигентих агената у IoT уређаје у оквиру које ALAS може наћи примену.

Међутим, перформансе ALAS-а се и даље могу побољшавати како би агенти били што продуктивнији. Један од првих будућих праваца развоја укључује потпуну подршку дистрибуираном систему за не-аксиоматско резоновање у виду имплементације преосталих нивоа не-аксиоматске логике. Поред основних правила за закључивање која су наведена у дисертацији, план је да се језик додатно побољша имплементацијом нових правила за закључивање као што су унија, разлика, негација, одлука, егземплификација, интерсекција, конверзија, контрапозиција итд.

Будући рад укључује и примену напредних могућности textX-а у виду текст едитора, пријављивања грешака, бојења синтаксе, валидације кода, приказа свих референци, скока на дефиницију правила, допуне кода, *codelens*-а итд.

С обзиром да прилив нових информација рапидно расте, потребно је стално пратити нове технологије које подржавају обраду огромних количина информација. У плану је да се уместо тренутно коришћене Titan граф базе података пређе на бесплатну Neo4J[19] граф базу података која по својим могућностима предњачи у односу на друге базе података.

[19] https://neo4j.com

10 Литература

Alberola, Juan M., Jose M. Such, Vicent Botti, Agustin Espinosa, and Ana Garcia-Fornes. 2013. „A scalable multiagent platform for large systems.” Computer Science and Information Systems 10(1):51-77. doi:10.2298/CSIS111029039A.

ASTRA. 2017. „ASTRA agent programming language.“ Pristupljeno: 10.07.2017. http://astralanguage.com.

Badica, Costin, Zoran Budimac, Hans-Dieter Burkhard, and Mirjana Ivanović. 2011. „Software agents: Languages, tools, platforms.“ Computer Science and Information Systems 8 (2):255-98. doi:10.2298/CSIS110214013B.

Baljak, V, M. Tudor Benea, A. El Fallah-Seghrouchni, C. Herpson, S. Honiden, T. Thuy Nga Nguyen, A. Olaru, R. Shimizu, K. Tei and S. Toriumi. 2012. „S-CLAIM: An Agent-based Programming Language for AmI, A Smart-Room Case Study.“ Procedia Computer Science 10:30-7. doi:doi.org/10.1016/j.procs.2012.06.008.

Bellifemine, Fabio Luigi, Giovanni Caire, and Dominic Greenwood. 2007. Developing Multi-Agent Systems with JADE. Edited by Liverpool University Michael Wooldridge, UK, Wiley Series in Agent Technology. Southern Gate, Chichester, West Sussex PO19 8SQ, England: John Wiley and Sons.

Bettini, Lorenzo. 2013. Implementing Domain-Specific Languages with Xtext and Xtend. Birmingham, UK: Packt Publishing Ltd.

Bergenti, Federico, Eleonora Iotti, and Agostino Poggi. 2016. „Core Features of an Agent-Oriented Domain-Specific Language for JADE Agents.” In de la Prieta F. et al. (eds) Trends in Practical Applications of Scalable Multi-Agent Systems, the PAAMS Collection. Springer 473:213–224. doi:10.1007/978-3-319-40159-1_18.

Bordini, Rafael H., Jomi Fred Hübner, and Michael Wooldridge. 2007. Programming Multi-Agent Systems in AgentSpeak using Jason, Wiley Series in Agent Technology: John Wiley & Sons Ltd.

Bordini, Rafael H., Mehdi Dastani, Jürgen Dix, and Amal El Fallah-Seghrouchni. 2009. „Multi-Agent Programming: Languages, Platforms and Applications.“ In Multiagent Systems, Artificial Societies, and Simulated Organizations. New York: Springer US.

Challenger,Moharram, Marjan Mernik, Geylani Kardas, and Tomaž Kosar. 2016. „Declarative specifications for the development of multi-agent systems.” Computer Standards & Interfaces 43:91-115, doi:10.1016/j.csi.2015.08.012

Collier, Rem W., Seán Russell, and David Lillis. 2015a. „Exploring AOP from an OOP perspective.“ In Proceedings of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control, 25-36. Pittsburgh, PA, USA: ACM.

Collier, Rem W., Seán Russell, and David Lillis. 2015b. „Reflecting on Agent Programming with AgentSpeak(L).” In PRIMA 2015: Principles and Practice of Multi-Agent Systems: 18th International Conference, Bertinoro, Italy, edited by Qingliang Chen, Paolo Torroni, Serena Villata, Jane Hsu and Andrea Omicini, 351-66. Cham: Springer International Publishing.

Cosenza, Biagio, Nikita Popov, Ben Juurlink, Paul Richmond, Mozhgan Kabiri Chimeh, Carmine Spagnuolo, Gennaro Cordasco, and Vittorio Scarano. 2018. „OpenABL: A Domain-Specific Language for Parallel and Distributed Agent-Based Simulations.” 24th International Conference on Parallel and Distributed Computing, Turin, Italy, August 27 – 31. doi:10.1007/978-3-319-96983-1_36.

Dastani, Mehdi, M. Birna van Riemsdijk, Frank Dignum, and John-Jules Ch Meyer. 2004. „A Programming Language for Cognitive Agents Goal Directed 3APL.” In Programming Multi-Agent Systems: First International Workshop, PROMAS 2003, Melbourne, Australia, July 15, 2003, Selected Revised and Invited papers, edited by Mehdi M. Dastani, Jürgen Dix and Amal El Fallah-Seghrouchni, 111-30. Berlin, Heidelberg: Springer Berlin Heidelberg.

Dastani, Mehdi. 2008. „2APL: a practical agent programming language.“ Autonomous Agents and Multi-Agent Systems (JAAMAS) 16 (3):214-48. doi:10.1007/s10458-008-9036-y.

Davies, Winton, and Peter Edwards. 1994. „Agent-K: An Integration of AOP and KQML.“ In Proceedings of the CIKM '94 Workshop on Intelligent Agents, edited by Tim Finin and Y Labrou. NIST.

Dejanović, Igor, Branko Perišić, Gordana Milosavljević. 2010. „Arpeggio: Pakrat parser interpreter.” XX naučna i biznis konferencija YUINFO, Kopaonik, Srbija, 3-6. Mart.

Dejanović, Igor, Gordana Milosavljević, and Renata Vaderna. 2016. „Arpeggio: A flexible PEG parser for Python.“ Knowledge-based systems 95:71-4. doi:10.1016/j.knosys.2015.12.004.

Dejanović, Igor, Renata Vaderna, Gordana Milosavljević, and Željko Vuković. 2016. „textX: A Python tool for Domain-Specific Languages Implementation.“ Knowledge-based systems 115 (1):1-4. doi:10.1016/j.knosys.2016.10.023.

Dejanović, Igor. 2019a. „Arpeggio.“ Pristupljeno 28.06.2017. https://github.com/textX/Arpeggio.

Dejanović, Igor. 2019b. „textX.“ Pristupljeno 28.06.2017. https://github.com/textX/textX.

Demirkol, S., M. Challenger, S. Getir, T. Kosar, G. Kardas, and M. Mernik. 2012. „SEA_L: A Domain-specific Language for Semantic Web enabled Multi-agent Systems.“ In Federated Conference on Computer Science and Information Systems (FedCSIS 2012), 1373-80. Wroclaw, Poland: IEEE.

Demirkol, Sebla, Moharram Challenger, Sinem Getir, Tomaž Kosar, Geylani Kardas, and Marjan Mernik. 2013. „A DSL for the development of software agents working within a semantic web environment.“ Computer Science and Information Systems 10 (4):1525-56. doi:10.2298/CSIS121105044D.

Dianxiang, Xu, Zheng Guoliang, and Fan Xiaocong. 1998. „A logic based language for networked agents.“Information and Software Technology 40 (8):435-42. doi:10.1016/S0950-5849(98)00061-5.

EBNF. „Extended Backus–Naur form.“ Wikipedia, Accessed 25 July 2017. https://en.wikipedia.org/wiki/Extended_Backus%E2%80%93Naur_form.

Efftinge, Sven, and Markus Völter. 2006. „oAW xText: a framework for textual DSLs.” Workshop on Modeling Symposium at Eclipse Summit. Pristupljeno 22.09.2018. http://voelter.de/data/workshops/EfftingeVoelterEclipseSummit.pdf.

Faci, Noura Zahia Guessoum, and Olivier Marin. 2006. „DimaX: A fault-tolerant multi-agent platform.” In Proceedings of the 2006 International Workshop on Software Engineering for Large-scale Multi-agent Systems, SELMAS '06, 13-20, New York, NY, USA, 2006. ACM. doi: 10.1145/1138063.1138067.

Feraud, Maxime, and Stéphane Galland. 2017. „First Comparison of SARL to Other Agent-Programming Languages and Frameworks.“ Procedia Computer Science 109:1080-5. doi:10.1016/j.procs.2017.05.389.

Finin, Tim, Richard Fritzson, Don McKay, and Robin McEntire. 1994. „KQML as an agent communication language.” In Proceedings of the third international conference on Information and knowledge management, CIKM ’94 456–463. ACM, New York, USA. doi:10.1145/191246.191322.

FIPA. 2018. „FIPA ACL Message Structure Specification.” Pristupljeno 22.08.2018. http://www.fipa.org/specs/fipa00061/SC00061G.pdf.

Fowler, Martin. 2010. Domain-Specific Languages. 1st Edition ed: Addison-Wesley Professional.

Garrido, Piedad, Javier Barrachina, Francisco J. Martinez, and Francisco J. Seron. 2017. „Smart Tourist Information Points by Combining Agents, Semantics and AI Techniques.” Computer Science and Information Systems 14(1):1–23. doi:10.2298/CSIS150410029G.

Georgeff, Michael P., and Amy L. Lansky. 1987. „Reactive reasoning and planning.” In Proceedings of the sixth National conference on Artificial intelligence, 677-82. Seattle, Washington: AAAI Press.

Georgeff, Michael, Barney Pell, Martha Pollack, Milind Tambe, and Michael Wooldridge. 1999. „The Belief-Desire-Intention Model of Agency.” Intelligent Agents V: Agents Theories, Architectures, and Languages: 5th International Workshop, ATAL’98 Paris, France:1-10. doi:10.1007/3-540-49057-4_1.

German, Ernesto, and Leonid Sheremetov. 2007. „Specifying Interaction Space Components in a FIPA-ACL Interaction Framework.” Languages, Methodologies and Development Tools for Multi-Agent Systems, First International Workshop, LADS 2007, Durham, UK, September 4-6. Revised Selected Papers 5118:191-208. doi:10.1007/978-3-540-85058-8_12.

GitHub. 2016a. „TextX support for Emacs.” Pristupljeno 20.01.2019. https://github.com/textX-tools/textx-mode.

GitHub. 2016b. „Vim editor syntax support for textX.” Pristupljeno 25.02.2019. https://github.com/textX/textx.vim.

GitHub. 2017. „A Visual Studio Code extension for visualization of code/model written in a DSL created using textX.” Pristupljeno 25.02.2019. https://github.com/textX-tools/viewX-vscode.

GitHub. 2018. „textX vs code.” Pristupljeno 20.02.2019. https://github.com/textX/textX-vscode.

GraphViz. 2018. „Graphviz - Graph Visualization Software.” Pristupljeno 25.09.2018. http://www.graphviz.org.

Green, Shaw, Leon Hurst, Brenda Nangle, Pádraig Cunningham, Fergal Somers and Richard Evans. 1996. „Software Agents: A review”. Technical Report, Trinty College Dublin, Ireland.

Gremlin. 2019. „Gremlin-Scala for Apache Tinkerpop 3.“ Pristupljeno 25.09.2017. https://github.com/mpollmeier/gremlin-scala.

Guerra-Hernandez, Alejandro, Jose Martin Castro-Manzano, and A. El Fallah-Seghrouchni. 2009. „CTL AgentSpeak(L): A specification language for agent programs.” Journal of Algorithms Cognition, Informatics and Logic, 60:31-40. doi:10.1016/j.jalgor.2009.02.003.

Hastjarjanto, Tom, Johan Jeuring, and Sean Leather. 2013. „A DSL for describing the artificial intelligence in real-time video games.” In Proceedings of the 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change (GAS ’13), 8-14. IEEE Press, Piscataway, NJ, USA. doi:10.1109/GAS.2013.6632583.

Hindriks, Koen V. 2017. Programming Cognitive Agents in Goal.

Hoek, Wiebe van der, and Michael Wooldridge. 2013. „Logics for multiagent systems.“ In Multiagent Systems, Intelligent robotics and autonomous agents, edited by Gerhard Weiss, 761-811. MIT Press.

Hudak, Paul. 1998. „Domain-Specific Languages.” In Handbook of Programming Languages, Vol. 3: Little Languages and Tools, edited by Peter H. Salus, 39–60. Indianapolis: MacMillan.

JADE. 2017. „JAVA Agent DEvelopment Framework.“ Pristupljeno 25.08.2017. http://jade.tilab.com.

Javacc. 2017. „The Java Parser Generator.“ Pristupljeno 12.05.2017. https://javacc.org.

Jennings, Nicholas R. 2000. „On agent-based software engineering.“ Artificial Intelligence 117 (2):277-96. doi:10.1016/s0004-3702(99)00107-1.

Jinja2. 2018. „Jinja Documentation.” Pristupljeno 12.05.2017. http://jinja.pocoo.org.

Kelly, Steven, and Juha-PekkaTolvanen. 2008. Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press.

Kosanović, Milan, Igor Dejanović, and Gordana Milosavljević. 2016. „Applang – A DSL for specification of mobile applications for android platform based on textX.“ AIP Conference Proceedings 1738 (1):240003. doi:10.1063/1.4952022.

Kosar, Tomaž, Nuno Oliveira, Marjan Mernik, Maria Joao Varanda Pereira, Matej Črepinšek, Daniela da Cruz, and Pedro Rangel Henriques. 2010. „Comparing General-Purpose and Domain-Specific Languages.” An Empirical Study Computer Science and Information Systems 7(2):247–264. doi:10.2298/CSIS1002247K.

Kosar, Tomaž, Sudev Bohra, and Marjan Mernik. 2016. „Domain-Specific Languages: A Systematic Mapping Study.“ Information and Software Technology 71:77-91. doi:10.1016/j.infsof.2015.11.001.

Kulesza, Uirá, Alessandro Garcia, Carlos Lucena, and Paulo Alencar. 2005. „A generative approach for multi-agent system development.“ In Software Engineering for Multi-Agent Systems III, edited by Choren Ricardo, Garcia Alessandro, Lucena Carlos and Romanovsky Alexander, 52-69. Springer-Verlag Berlin, Heidelberg.

Laarabi, Mohamed Haitam, Claudio Roncoli, Roberto Sacile, Azedine Boulmakoul, and Emmanuel Garbolino. 2013. „An overview of a multiagent-based simulation system for dynamic management of risk related to Dangerous Goods Transport.” In 2013 IEEE International Systems Conference (SysCon) 830-835. doi: 10.1109/SysCon.2013.6549980

Luck, M., R. Ashri, and M. D. Inverno. 2004. Agent-Based Software Development. Norwood, MA: Artech House.

Lujak, M., H. Billhardt, J. Dunkel, A. Fernández, R. Hermoso, and S. Ossowski. 2017. „A Distributed Architecture for Real-Time Evacuation Guidance in Large Smart Buildings.” Computer Science and Information Systems 14 (1): 257–282. doi:10.2298/CSIS161014002L.

Mernik, Marjan, Jan Heering, and Anthony M. Sloane. 2005. „When and How to Develop Domain-Specific Languages.” ACM Computing Surveys 37(4):316-344. doi:10.1145/1118890.1118892.

Metzger, Mieczyslaw and Grzegorz Polakow. 2011. „A survey on applications of agent technology in industrial process control”, Industrial Informatics, IEEE Transactions on 7(4):570–581. doi:10.1109/TII.2011.2166781.

Minotti, Mattia, Andrea Santi, and Alessandro Ricci, 2010. „Developing web client applications with JaCaWeb.” In Proceedings of the 11th WOA 2010 Workshop, Dagli Oggetti Agli Agenti (621), CEUR-WS.org.

Mitrović, Dejan, Mirjana Ivanovic, Zoran Budimac, and Milan Vidaković. 2011. An overview of agent mobility in heterogeneous environments. Paper presented at the Proceedings of the workshop on applications of software agents, Novi Sad, Serbia, July 3-5.

Mitrović, Dejan, Mirjana Ivanović, and Milan Vidaković. 2011. Introducing ALAS: A Novel Agent-Oriented Programming Language. Paper presented at the Symposium on Computer Languages, Implementations, and Tools (SCLIT 2011), Halkidiki, Greece, September 19-25

Mitrović, Dejan, Mirjana Ivanović, Milan Vidaković, and Zoran Budimac. 2014b. „Extensible Java EE-based Agent Framework in Clustered Environments.“ Lecture Notes in Computer Science 8732:202-15. doi:10.1007/978-3-319-11584-9_14.

Mitrović, Dejan, Mirjana Ivanović, Milan Vidaković, and Zoran Budimac. 2016a. „The Siebog Multiagent Middleware.“ Knowledge-based systems 103:56-9. doi:10.1016/j.knosys.2016.03.017.

Mitrović, Dejan, Mirjana Ivanović, Milan Vidaković, Dejan Sredojević, and Dušan Okanović. 2014c. „Integracija agentskog jezika ALAS u Java agentsko okruženje XJAF.” XX naučna i biznis konferencija, Kopaonik, Srbija, 9-13. Mart.

Mitrović, Dejan, Mirjana Ivanović, Milan Vidaković, Zoran Budimac, and Jovana Vidaković. 2016b. „Siebog: An Enterprise-Scale Multiagent Middleware.“ INFORMATION TECHNOLOGY AND CONTROL 45:164-74. doi:10.5755/j01.itc.45.2.12621.

Mitrović, Dejan, Mirjana Ivanović, Zoran Budimac, and Milan Vidaković. 2012. „Supporting heterogeneous agent mobility with ALAS.“ Computer Science and Information Systems 9 (3):1203-29. doi:10.2298/CSIS120102025M.

Mitrović, Dejan, Mirjana Ivanović, Zoran Budimac, and Milan Vidaković. 2014a. „Radigost: Interoperable web-based multi-agent platform.“ Journal of Systems and Software 90:167-78. doi:10.1016/j.jss.2013.12.029.

Mitrović, Dejan. 2015. „Intelligent multiagent systems based on distributed non-axiomatic reasoning.“ Ph.D. thesis, University of Novi Sad.

Müller, V. C. 2013. Philosophy and Theory of Artificial Intelligence. Berlin, Heidelberg: Springer.

Mzahm, Anas M., Mohd Sharifuddin Ahmad, and Alicia Y. C. Tang. 2014. „Enhancing the Internet of Things (IoT) via the Concept of Agent of Things (AoT).” Journal of Network and Innovative Computing 2:101-110.

Nabeth, Thierry, Albert Angehrn, Liana Razmerita, and Claudia Roda. 2005. „InCA: a Cognitive Multi-Agents Architecture for Designing Intelligent & Adaptive Learning Systems.“ Computer Science and Information Systems 2(2):99-114. doi:10.2298/CSIS0502099N.

Nwana, Hyacinth. 1996. „Software agents: An overview.” The knowledge engineering review 11:205-244.

Okanović, Dušan, Milan Vidaković, Željko Vuković, Dejan Mitrović, and Mirjana Ivanović. 2014. „Editor for agent-oriented programming language ALAS.“

Paper presented at the International Conference on Information Society and Techology 2014, Kopaonik, Serbia, March 9-13.

Pokahr, Alexander, Lars Braubach, Christopher Haubeck, and Jan Ladiges. 2014. „Programming BDI Agents with Pure Java.“ In Proceedings of the 12th German Conference on Multiagent System Technologies (MATES 2014), 216-33. Stuttgart, Germany: Springer-Verlag New York, Inc.

Qusay, H. Mahmoud. 2000. „Software Agents: Characteristics and Classification.” School of Computing Science, Simon Fraser University, 1-12.

Ranković, Ivan. 2018. „Istorijski Razvoj Veštačke Inteligencije.” Pristupljeno 10.02.2018. https://raf.edu.rs/citaliste/istorija/3706-istorijski-razvoj-vestacke-inteligencije.

Rao, Anand S. 1996. „AgentSpeak(L): BDI agents speak out in a logical computable language.“ In Agents Breaking Away (MAAMAW 1996), edited by W. Van de Velde and J. W. Perram, 42–55. Springer, Berlin, Heidelberg.

Rao, Anand S., and Michael P. Georgeff. 1995. „BDI Agents: From Theory to Practice.” In Proceedings of the First International Conference on Multiagent Systems (ICMAS-95), 312-319

Rao, Anand S., and Michael P. Georgeff. 1998. „Decision Procedures for BDI Logics.“ Journal of Logic and Computation 8 (3):293–343. doi:10.1093/logcom/8.3.293.

Robinson, Ian, Jim Webber, and Emil Eifrem. 2013. Graph databases. 2nd ed: O’Reilly Media, Inc.

Rodriguez, Sebastian, Nicolas Gaud, and Stéphane Galland. 2014. „SARL: A General-Purpose Agent-Oriented Programming Language.“ In 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT), 103-10. Warsaw, Poland: IEEE Computer Society Press. doi: 10.1109/WI-IAT.2014.156.

Sadalage, Pramod J., and Martin Fowler. 2012. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence: Addison-Wesley Professional.

Savić, Ivan. 2018. „Graf baze podataka.” Zbornik radova Fakulteta tehničkih nauka, Novi Sad 85-88. doi:10.24867/01BE22Savic.

Schreiber, Guus, and Yves Raimond. „RDF 1.1 Primer.“ W3C, Pristupljeno 20.08.2017. https://www.w3.org/TR/rdf11-primer.

Serenko, Alexander, and Brian Detlor. 2004. „Intelligent agents as innovations.“ AI & SOCIETY 18 (4):364-81. doi:10.1007/s00146-004-0310-5.

Shoham, Yoav, and Kevin Leyton-Brown. 2009. Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations. 1st edition ed: Cambridge University Press.

Shoham, Yoav. 1990. „Agent-Oriented Programming.“ Technical Report STAN-CS-90-1335: Stanford University: Computer Science Department.

Shoham, Yoav. 1993. „Agent Oriented Programming.“ Journal of Artificial Intelligence 60 (1):51-92.

Simić, Miloš, Željko Bal, Renata Vaderna, and Igor Dejanović. 2015. „PyTabs: A DSL for simplified music notation.“ In The 5th International Conference on Information Society and Technology (ICIST 2015), edited by M. Zdravković, Trajanović, M., Konjović, Z., 439-43. Kopaonik, Serbia.

Smith, Robin. 2017. „Aristotle's Logic.“ Stanford Encyclopedia of Philosophy, Pristupljeno 15.07.2017. https://plato.stanford.edu/entries/aristotle-logic.

Sommers, Frederic Tamler, and George Englebretsen. 2000. An Invitation to Formal Reasoning: The Logic of Terms: Ashgate.

Soon, Gan Kim, Chin Kim On, Patricia Anthony, Abdul Razak Hamdan. 2018. „A Review on Agent Communication Language.” In: Alfred R., Lim Y., Ibrahim A., Anthony P. (eds) Computational Science and Technology. Lecture Notes in Electrical Engineering 481:481-491. Springer, Singapore. doi:10.1007/978-981-13-2622-6_47

Sredojević, Dejan, Dušan Okanović, Milan Vidaković, Dejan Mitrović, and Mirjana Ivanović. 2015. „Domain specific agent-oriented programming language based on the Xtext framework.” In 5th International Conference on Information Society and Technology (ICIST 2015), edited by M. Zdravković, Trajanović, M., Konjović, Z., 495-501. Kopaonik, Serbia.

Sredojević, Dejan, Milan Vidaković, and Dušan Okanović. 2015. „Integration of agent domain-specific language ALAS into extensible Java-based framework XJAF.” Paper presented at the 34th International Conference on Organizational Science Development internationalization and cooperation, Portorož, March 25-27.

Sredojević, Dejan, Milan Vidaković, and Mirjana Ivanović. 2018. „ALAS: Agent-Oriented Domain-Specific Language for the Development of Intelligent Distributed Non-Axiomatic Reasoning Agents.” Enterprise Information Systems, 12(8-9):1058-1082, doi:10.1080/17517575.2018.1482567.

Sredojević, Dejan, Milan Vidaković, Dušan Okanović, Dejan Mitrović, and Mirjana Ivanović. 2016. „Conversion of the agent-oriented domain-specific language ALAS into JavaScript.” Paper presented at the 5th Symposium on Computer Languages, Implementations and Tools, Rhodes, Greece, September 23-29.

Sredojević, Dejan, Milan Vidaković, Mirjana Ivanović, and Dejan Mitrović. 2017. „Extension of Agent-oriented Domain-specific language ALAS as a support to Distributed Non-Axiomatic Reasoning.” Paper presented at the International Conference on Information Society and Techology (ICIST 2017), Kopaonik, Serbia, March 12-15.

Suna, Alexandru, and Amal El Fallah-Seghrouchni. 2007. „Programming mobile intelligent agents: an operational semantics.“ Web Intelligence and Agent Systems: An international journal 5 (1):47-67.

Šestak, Martina. 2014. „Graf baze podataka.” Pristupljeno 15.05.2019. https://bib.irb.hr/datoteka/764460.1-Martina_estak_Graf_baze_podataka.pdf.

Šestak, Martina. 2016. „Usporedba jezika za graf baze podataka.” Pristupljeno 15.05.2019. https://bib.irb.hr/datoteka/829680.Diplomski_rad_Martina_estak.pdf.

Thomas, S. Rebecca. 1995. „The PLACA agent programming language.“ In Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents, 355-70. Amsterdam, The Netherlands: Springer-Verlag New York, Inc.

Titan. 2015. „Distributed Graph Database.” Pristupljeno 20.08.2017. http://titan.thinkaurelius.com.

Tiwari, Shashank. 2011. Professional NoSQL: Wrox, John Wiley & Sons, Inc.

Vaderna, Renata, Željko Vuković, Dušan Okanović, and Igor Dejanovic. 2015. „A Domain-Specific Language for Service Level Agreement Specification.“ In The 7th International Conference on Information Technology, 693-7. Amman, Jordan.

Vidaković, Milan, and Zora Konjović. 2002. „EJB based intelligent agents framework.“ In The 6th IASTED International Conference on Software Engineering and Applications (SEA 2002), 343-348.

Vidaković, Milan, Goran Sladić, and Zora Konjović. 2003. „Security management in J2EE based intelligent agent framework.” In Proceedings of the 7th IASTED International Conference on Software Engineering and Applications (SEA 2003) 128–133.

Vidaković, Milan, Mirjana Ivanović, Dejan Mitrović, and Zoran Budimac. 2013. „Extensible Java EE-Based Agent Framework - Past, Present, Future.“ In Multiagent Systems and Applications, edited by Maria Ganzha and Lakhmi C. Jain, 45:55-88. Springer Berlin Heidelberg.

Vidaković, Milan. 2003. „Extensible java based agent framework.” Ph.D. thesis, Faculty of Technical Sciences, University of Novi Sad, Serbia.

Voelter, Markus. 2013. DSL Engineering: Designing, Implementing and Using Domain-Specific Languages. CreateSpace Independent Publishing Platform.

Wang, Pei, and Seemal Awan. 2011. „Reasoning in non-axiomatic logic: a case study in medical diagnosis.“ In Proceedings of the 4th international conference on Artificial general intelligence, 297-302. Mountain View, CA: Springer-Verlag.

Wang, Pei. 1996. „Non-Axiomatic Reasoning System - Exploring the Essence of Intelligence.“ Ph.D. thesis, Indiana University.

Wang, Pei. 2006. Rigid flexibility. The logic of intelligence. Vol. 34. Dordrecht, The Netherlands: Springer.

Wang, Pei. 2013. Non-axiomatic logic: a model of intelligent reasoning. World Scientific Publishing Co Pte Ltd.

Weiss, Gerhard. 2000. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. Cambridge: MIT Press.

Weiss, Gerhard. 2013. Multiagent systems. 2nd ed. Cambridge: MIT Press.

Wiebe van der, Hoek, and Michael Wooldridge. 2013. „Logics for Multiagent Systems.” In: Multiagent Systems, edited by Gerhard Weiss, 761–811. Cambridge: MIT Press.

Wooldridge, Michael, and Nicholas R. Jennings. 1995. „Intelligent Agents: Theory and Practice.” Knowledge Engineering Review 10 (2): 115–152. doi:10.1017/S0269888900008122.

Wooldridge, Michael. 1997. Agent-based software engineering. IEE Proc Software Engineering 144:26-37.

Wooldridge, Michael. 2000. Reasoning about rational agents: MIT press.

Wooldridge, Michael. 2009. An Introduction to MultiAgent Systems. 2nd edition ed: Wiley.

Списак слика

Списак табела

Списак листинга

Списак скраћеница

ACL	Агентски комуникациони језик (енг. Agent Communication Language)
AOP	Агентско програмирање (енг. Agent-Oriented Programming)
AOPL	Агентски програмски језици (енг. Agent-Oriented Programming Languages)
BDI	Веровање-жеље-намере (енг. Belief–Desire–Intention)
DNARS	Дистрибуирани систем за не-аксиоматско резоновање (енг. Distributed Non-Axiomatic Reasoning System)
DSL	Домен-оријентисани језик (енг. Domain-Specific Language)
FIPA	Foundation for Intelligent Physical Agents
GPL	Језик опште намене (енг. General-Purpose Language)
IDE	Интегрисано развојно окружење (енг. Integrated Development Environment)
KQML	Knowledge Query and Manipulation Language
MAS	Мултиагентски систем (енг. Multiagent System)
NAL	Не-аксиоматска логика (енг. Non-Axiomatic Logic)
OOP	Објектно-оријентисано програмирање (енг. Object-Oriented Programming)
RDF	Resource Description Framework
URI	Uniform Resource Identifier

Биографија аутора

Дејан Средојевић је рођен 12.09.1986. године у Лозници. Основно школовање започео је 1993. године у основној школи "Боривоје Ж. Милојевић" у Брштици (општина Крупањ) а завршио 2001. године у Крупњу. Након основне школе уписује средњу електротехничку школу "Никола Тесла" у Крупњу и завршава је 2005. године. За време основног и средњег образовања учествује на такмичењима из математике, хемије и електронике и добитник је више диплома за постигнуте резултате. Добитник је Вукове дипломе.

2005. године уписује Факултет техничких наука у Новом Саду, одсек електротехника и рачунарство, смер рачунарство и аутоматика, усмерење аутоматика и управљање системима. Основне – бечелор студије завршио је 2009. године одбранивши дипломски рад са оценом 10. Мастер студије на истом смеру уписује исте године и завршава их 2011. године са укупним просеком 9.33 одбранивши мастер рад са оценом 10.

За време студија учествује на такмичењима из информатике и 2010. године је освојио прво место из информатике на регионалном такмичењу и смотри научноистраживачког и уметничког стваралаштва талената одржаном у Лозници. Исте године, на 53-ој републичкој смотри и такмичењу научноистраживачких радова талената осваја прво место и специјалну диплому коју је добио од стране Републичког центра за таленте.

За постигнуте резултате Дејан Средојевић је у септембру 2010. године награђен Повељом са златним грбом општине Крупањ.

Од 2011. године запослен је у Високој пословној школи струковнох студија у Новом Саду где је као асистент изводио наставу на преко 10 предмета. Активно се бави научноистраживачким радом и до сада је објавио 15 радова на међународним конференцијама и један рад у истакнутом међународном часопису. Од новембра 2018. године ради као софтвер инжењер у компанији *RT-RK Automotive* у Новом Саду.

Ожењен је и има два сина, Вука и Ђорђа.