УНИВЕРЗИТЕТ У БЕОГРАДУ
ЕЛЕКТРОТЕХНИЧКИ ФАКУЛТЕТ

Зоран Б. Бабовић

Семантичка интеграција сензорских мрежа

Докторска дисертација

Београд, 2018

UNIVERSITY OF BELGRADE
SCHOOL OF ELECTRICAL ENGINEERING

Zoran B. Babovic

Semantic-based integration of sensor networks

Doctoral dissertation

Belgrade, 2018

Комисија за преглед и оцену дисертације

Ментор:

Проф. др Вељко Милутиновић
редовни проф. у пензији, Универзитет у Београду, Електротехнички факултет

Чланови Комисије:

Проф. др Јелица Протић
Редовни проф., Универзитет у Београду, Електротехнички факултет

Проф. др Саша Малков
Ванредни проф., Универзитет у Београду, Математички факултет

Датум одбране: _____

Захвалнице

Најпре бих се захвалио својим другарима и колегама са којима сам водио стручне расправе, што ме је инспирисало на креирање сопствених идеја, а у свакој од тих идеја је уткан у некој мери свако од њих.

Захваљујем се професорима са Електротехничког факултета Универзитета у Београду на докторским и основним студијама који су ми пружили квалитетну наставу и тиме оспособили да пратим најновија достигнућа у области рачунарства.

Посебно се захваљујем свом ментору професору др Вељку Милутиновићу који је најзаслужнији за формирање мог научно-истраживачког идентитета и који ми је своје огромно искуство и стручност несебично преносио у свакој ситуацији.

Срдачно се захваљујем и комисији која је прегледала овај рад и дала низ конструктивних коментара у циљу његовог побољшања.

Хвала мојој мајци на подршци и пожртвовању током целокупног мог школовања, оцу, брату и свим пријатељима и родбини на подршци да истрајем у раду на овој дисертацији.

И на крају слава и хвала Господу нашем Исусу Христу, јер сам се радом на овој дисертацији, као и током сваког другог креативног рада, уверио у истинитост Његових речи: „без мене не можете чинити ништа“ (Јн.15,5).

Наслов дисертације: Семантичка интеграција сензорских мрежа

Резиме: Развој CMOS технологије високог степена интеграције проузроковао је интензиван развој сензорских технологија, бежичних комуникација и енергетски ефикасних процесора, који заједно чине сензорске чворове, способне да опажају нашу околину, врше обраду података и да размењују податке са другим уређајима и корисницима на Интернету. Такав технолошки развој је довео до појаве визије Интернета Ствари (енг. *Internet of Things - IoT*), са циљем да се корисницима на Интернету пруже информације из реалног света који нас окружује. Као један од предуслова за реализацију замишљених IoT сервиса и производа, неопходна је хоризонтална интеграција распоређених сензорских мрежа, имплементацијом платформи које би омогућиле интеграцију хетерогених сензорских мрежа, састављених од различитих сензорских уређаја, који користе различите комуникационе протоколе и формате порука и у могућности су да опслужују већи број корисника. Предмет ове дисертације су архитектуре за интеграцију сензорских мрежа, које у општем случају пружају подршку раду IoT апликацијама са захтевима за високим перформансама. Од основних генеричких типова, издвојена је архитектура заснована на семантици података и циљ ове дисертације је анализа архитектура које омогућавају семантичку интеграцију сензорских мрежа коришћењем семантичких веб технологија ради омогућавања интероперабилности сензорских података и мрежа. У раду су најпре идентификовани основни типови генеричких архитектура и дате су њихове кључне карактеристике и начин реализације, а за архитектуре засноване на пролазном уређају и брокеру порука урађена је евалуација перформанси у раду са подацима у реалном времену креирањем симулационог окружења. Затим су архитектуре са семантичком интеграцијом сензорских мрежа класификоване идентификовањем типичних пројектанских приступа. Издвојене су две групе приступа, и то приступи оријентисани ка сензорским мрежама и апликативно оријентисани приступи, а свака група садржи даље по четири типа архитектуралних приступа. За сваки архитектурални тип дата је анализа предности и недостатака коришћеног приступа и кратак опис конкретних представника. На основу

уочених недостатака анализираних архитектура, предложена је нова архитектура која користи приступ заснован на дистрибуираним базама података. У сржи те архитектуре је дистрибуирано *RDF* складиште високих перформанси, специјализовано за рад са семантичким сензорским подацима у оперативној меморији, у циљу подршке ефикасног складиштења и претраживања семантичких сензорских података у условима интензивног генерисања нових сензорских опажања са већег броја сензора. У предложеном *RDF* складишту дат је низ пројектантских решења како би се остварили жељене перформансе, укључујући начин креирања индексних структура за чување и претрагу *RDF* тројки, предикатски засновану стратегију дистрибуције *RDF* тројки по рачунарским чворовима, алгоритам креирања плана извршавања *SPARQL* упита и начин одржавања вишедимензионалних индексних структура за чување резултата мерења, временских и просторних података сензорских опажања. Евалуација перформанси је изведена са скупом података сензорских мерења са неколико хиљада метеоролошких станица у САД за време урагана Чарли, током 2004. године, симулирањем ситуације генерисања сензорских опажања у реалном времену. Измерена времена извршавања временско-просторних-вредносних семантичких упита паралелно са интензивним додавањем нових сензорских опажања показују да је реализовано *RDF* складиште веома ефикасно и да постиже знантно боље перформансе од јавног доступног *RDF* складишта *Virtuoso*, у распону од 3 до 5 пута краће време извршавања упита и то само при најнижем оптерећењу. Са већом количином сензорских података време извршавања упита код *Virtuoso* складишта се неконтролисано повећава, док реализовано *RDF* складиште ефикасно скалира и благо повећава време извршавања упита.

Кључне речи: Сензорске мреже, интеграција, Internet of Things, семантичке веб технологије, RDF складишта

Научна област: Техничке науке - Електротехника и рачунарство

Ужа научна област: Рачунарска техника и информатика

УДК: 621.3

Disertation Title: Semantic-Based Integration of Sensor Networks

Abstract: The progress of CMOS technology very large scale integration has resulted in the intensive development of sensor technologies, wireless communications, and energy efficient processors, which together make sensor nodes capable of perceiving our environment, processing data and exchanging data with other devices and users on the Internet. Such technological development has led to the emergence of the Internet of Things (IoT) vision, with the goal of providing information of the real world that surrounds us to the Internet users. As one of the prerequisites for the realization of imaginary IoT services and products, the horizontal integration of deployed sensor networks is required, by implementing platforms that are able to serve a large number of users and would allow the integration of heterogeneous sensor networks consisted of different sensor devices, using different communication protocols and message formats. The subject of this dissertation is the architectures for the integration of sensor networks, which generally support IoT applications with high performance requirements. Among the available generic types, an architecture based on the data semantics is selected and the aim of this dissertation is the analysis of the architectures that enable semantic integration of sensor networks using Semantic Web technologies in order to achieve the interoperability of sensor data and networks. The basic generic architecture types were first identified and their key characteristics and method of implementation were given. For architectures based on the gateway and the message broker, the performance evaluation was performed in the scenario of real-time sensor messages delivery by creating a simulation environment. Then the architectures with semantic-based integration of sensor networks are classified by identifying typical design approaches. Two groups of approaches are identified, approaches oriented to sensor networks and application-oriented approaches, whereas each group contains four architectural types. For each architectural type, an analysis of the advantages and disadvantages of the used approach is given, as well as a brief description of the concrete representatives. As a result of observed shortcomings of the analyzed architectures, a new architecture is proposed which uses an approach based on distributed databases. At the core of this architecture is a distributed high-performance RDF triple store specialized for

working with semantic sensor data in RAM, in order to efficiently support storage and search of semantic sensor data in the conditions of intensive generation of new sensory observations from a large number of sensors. In the proposed RDF triple store, a number of innovative techniques are implemented in order to achieve the desired performance. Such techniques include the method of creating index structures for storing and searching RDF triples, the predicate-based distribution strategy of RDF triples to computer nodes, the algorithm of the SPARQL query execution plan, and the multidimensional indexing structures for keeping measurements, temporal, and spatial data of sensor observations. Performance evaluation was performed with a dataset of sensor data measurements from several thousands of meteorological stations in the United States during hurricane Charlie in 2004, simulating the situation of real-time sensor sensing. The measured execution time of semantic spatio-temporal-thematic queries over a different volume of sensor data shows that the implemented RDF triple store is very efficient and achieves significantly better performance than the publicly available RDF triple store Virtuoso, in the range from 3 to 5 times shorter query execution time, but only at the lowest workload. By increasing sensor data volume, the query execution time at the Virtuoso triple store increases uncontrollably, whereas the realized RDF triple store effectively scales and slightly increases the query execution time.

Keywords: Sensor networks, integration, Internet of Things, Semantic Web technologies, RDF triple store

Scientific field: Technical sciences - Electrical and computer engineering

Specific scientific field: Computing and informatics

UDC: 621.3

Садржај

Скраћенице

6LoWPAN - IPv6 over Low power Wireless Personal Area Network
AMQP - Advanced Message Queuing Protocol
API - Application Programming Interface
BFSK - Binary Frequency Shift Keying
BPSK - Binary Phase Shift Key
CMOS - Complementary Metal–Oxide–Semiconductor
CSIRO - Commonwealth Scientific and Industrial Research Organisation
DDS - Data Distribution Service
DOLCE - Descriptive Ontology for Linguistic and Cognitive Engineering
EU FP7 - European Union's Seventh Framework Programme
FEC - Forward Error Correction
FFD - Full Function Device
GML - Geography Markup Language
GPS - Global Positioning System
GSN - Global Sensor Network
GWT - Google Web Toolkit
HTML - HyperText Markup Language
HTTP - HyperText Transfer Protocol
IETF - Internet Engineering Task Force
IDL - Interface Definition Language
IoT - Internet of Things
IoT-EPI - IoT European Platform Initiative
IoE - Internet of Everything
JSON - JavaScript Object Notation
LR-WPAN - Low Rate Wireless Personal Area Networks
M2M - Machine-to-Machine
MAC - Medium Access Control
MQTT - Message Queue Telemetry Transport
nesC - network embedded systems C
NoSQL - "Not Only SQL"
O&M - Observation and Measurement
OGC SWE - Open Geospatial Consortium Sensor Web Enablement
O-QPSK - Offset Quadrature Phase Shift Key
OSI - Open Systems Interconnection
OWL - Web Ontology Language
PPDU - PHY Protocol Data Unit
PSDU - PHY Service Data Unit
RDBMS - Relational Database Management System
RDF - Resource Description Framework
RDFS - Resource Description Framework Schema
REST - Representational State Transfer
RFID - Radio-Frequency Identification
RFD - Reduced Function Device
SAS - Sensor Alerting Service
SBC - Single Board Computer
SensorML - Sensor Model Language
SNEEql - Sensor Network Engine Query Language
SNMPv3 - Simple Network Management Protocol

SOAP - Simple Object Access Protocol
SOS - Sensor Observation Service
SPARQL - Simple Protocol and RDF Query Language
SPS - Sensor Planning Service
SQL - Structured Query Language
SSN - Semantic Sensor Network
stSPARQL - Spatio-Temporal SPARQL
SWEET - Semantic Web for Earth and Environmental Terminology
TCP/IP - Transmission Control Protocol/Internet Protocol
TDD - Time Division Duplex
TinyOS - Tiny Operating System
URI - Uniform Resource Identifier
URL - Uniform Resource Locator
UWB - Ultra-wideband
W3C - World Wide Web Consortium
WPAN - Wireless Personal Area Networks
WSDL - Web Service Definition Language
WSN - Wireless Sensor Network
WWW - World Wide Web
XLINK - XML Linking Language
XML - eXtensible Markup Language
XMPP - Extensible Messaging and Presence Protocol
XEPs - XMPP Extension Protocols
XPATH - XML Path Language
XSLT - Extensible Stylesheet Language

Слике

1. Увод

Крајем 90их година 20. века напредак технологије интегрисаних електричних кола ниске потрошње, CMOS *(Complementary Metal–Oxide–Semiconductor)* комуникационих уређаја као и претварача отворио је нове могућности примене комбиновањем тих технологија. Знатно повећан број транзистора на чипу који се притом све више повећава пратећи Муров закон, повећао је могућности процесирања и количину меморије коју је могуће ставити на чип који заузима све мању површину и троши све мање енергије. Сличан напредак је у комуникационим уређајима, где су применом полупроводничке технологије креирани минијатурни радио чипови са веома ниском потрошњом. Са друге стране, интегрисани претварачи физичких величина у електричне сигнале такође ниске потрошње (у екстремним ситуацијама напајају се амбијенталном енергијом) и мале површине изведени као микро-електромеханички сензори (MEMC) могу да служе за детекцију светлости, топлоте, кретања, хемијских супстанци и слично. Паралелно са трендом минијатуризације дошло је и до смањење цена тих уређаја.

Комбиновањем описаних минијатурних процесорских, комуникационих и сензорских компонената на малој површини величине неколико квадратних центиметара, добијају се уређаји ниске потрошње и цене, које је могуће распоредити у неком простору или у близини објеката од интереса, ради добијања информација о физичким величинама и пољима у том простору. Такви уређаји се зову сензорски чворови (енг. *sensor nodes*). Због своје мале површине, такви уређаји не ремете природно стање простора и објеката. Применом процесорских и комуникационих капацитета на сензорским чворовима, измерене вредности о физичким величинама је могуће адекватно обрадити и проследити ка другим сличним уређајима у близини или у сарадњи са њима размени информације и координисати заједничке акције. Комуникација може бити остварена радио везом, оптичким путем што укључује и инфрацрвену светлост, а није искључена ни жична веза између

сензорских чворова. Умрежавањем сензорских чворова бежичном везом добијамо бежичну сензорску мрежу (енг. *Wireless Sensor Network - WSN*). Шароликост бежичних уређаја опремљених сензорском технологијом обухвата и пасивне бежичне компоненте RFID (*Radio-Frequency Identification*) тагове који се користе у сврху идентификације и праћења објеката, а такође и уређаје са већим капацитетом процесирања и комуникације попут паметних телефона и тзв. рачунарских модула (енг. *Single Board Computers - SBC*) који имају могућност извршавања сложенијих оперативних система и комуникационих протокола, а да притом и даље задржавају мале димензије и релативно малу потрошњу енергије.

Са све већим ширењем ових уређаја, постаје остварљива визија тзв. Интернета Ствари (енг. *Internet of Things - IoT*) [1, 2] која кроз широк скуп сензорских и информационо-комуникационих технологија омогућава повезивање реалног и виртуелног света кроз повезивање разнородних сензора и актуатора и пратећих уређаја на Интернет, како би широк скуп корисника добио информацију о разним физичким величинама и појавама из реалног окружења. Како су информације из реалног света често повезане са информацијама из друштвених мрежа, односно о људима који су опремљени неким од сензора, ова визија је еволуирала у општију физичку-сајбер-друштвену повезаност [3], названу Интернет Свега (енг. *Internet of Everything - IoE*) [4] са намером да се повежу људи, подаци, процеси и ствари. У позадини ове визије је да се без обзира на извор података, у сржи исте технологије користе за комуникацију, интеграцију, обраду података и закључивање односно доношење одређених одлука.

Потенцијалне примене сензорских мрежа су велике. Постоји неколико области где су сензорске мреже нашле своју примену као што је мониторинг животне средине и градова, мониторинг станишта животиња, детекција сеизмичке активности у некој области, праћење околности пре и током елементарних непогода, интелигентна контрола саобраћаја, праћење инвентара у фабрикама и магацинима, примене у пољопривреди за

прецизније одржавање одговарајућих услова за узгајање воћа и поврћа, системи за паметне куће у сврху побољшања енергетске ефикасности, здравствена заштита путем надгледања виталних параметара пацијената итд. У свакој од ових примена, захтевају се посебни режими рада сензорских уређаја, што одређује архитектуру коришћене сензорске мреже, врсту сензорских чворова и пратећих сензора, комуникационе протоколе и друге параметре.

Да би се реализовале ове могућности, пред информатичким и комуникационим технологијама се поставља низ изазова које треба ефикасно решити јер појединачни уређаји у бежичним сензорским мрежама имају јако ограничене ресурсе: имају ограничену процесорску брзину, капацитет складиштења података и комуникациону пропусну моћ. Пошто је могуће распоредити већи број уређаја који могу да комуницирају између себе, процесорска ограничења могу бити превазиђена пажљивим осмишљавањем дистрибуираних алгоритама у склопу којих би се извршавала одређена процесирања на тим умреженим сензорским чворовима.

Типична ситуација је да једна сензорска мрежа обухвата на стотине и хиљаде сензорских чворова, распоређена је од једног власника и служи за један апликативни домен, што се често означава као вертикално решење. Ако би други корисници, имали потребу за новом апликацојом, која би такође користила исте сензоре у тој области, потребно би било распоредити нову сензорску мрежу, јер сензорска опажања из постојеће мреже нису доступна за коришћење изван оквира њихове сензорске мреже односно апликације.

Поменута ограничења су довела до идеје креирања тзв. глобалне сензорске мреже, која би интегрисала све распоређене сензоре из разних сензорских мрежа, који се базирају на различитим хардверским уређајима, користе другачије комуникационе протоколе и формате података. Идеја односно визија такве глобалне интегрисане сензорске мреже назива се *Сензорска Мрежа* (енг. *Sensor Web*) [5]. У циљу добијања квалитетнијих информација из реалног

света, један од праваца истраживања је додавање контекстуалних информација уз измерене вредности сензора. Контекстуалне информације обично садрже временске, просторне и тематске мета-податке додате основним сензорским очитавањима. Интегрисане сензорске мреже које садрже ове додатне семантичке податке називају се *Семантичке Сензорске Мреже* (енг. *Semantic Sensor Web*) [6-8]. У основи те платформе користе се семантичке веб (енг. *Semantic Web*) технологије и то *Web Ontology Language* (*OWL*) [9] језик за представљање онтологија које омогућавају интероперабилност између различитих сензорских мрежа, као и *Resource Description Framework (RDF)* [10] модел за представљање описа сензора и семантичких сензорских очитавања [11, 12].

Успешном реализацијом принципа семантичких сензорских мрежа, стварају се услови да Интернет корисници кроз адекватне сервисе буду у могућности да дохватају опажања свих интегрисаних хетерогених сензора кроз слање одговарајућих семантичких упита и да на тај начин добију жељене информације о реалном свету који нас окружује. Додатно, коришћењем логичких закључивача који користе релације концепата из апликативних домена са сензорским опажањима дефинисаним у онтологијама, отварају се могућности извођења знања вишег нивоа о реалном свету.

Текуће анализе показују да можемо очекивати огроман раст распоређених међусобно повезаних сензорских уређаја у нашем окружењу до 2020-2021 године. Умерене процене иду од око 21 милијарде до 28 милијарди распоређених уређаја по прогнозама аналитичара из компанија *Gartner* [13] и *Ericsson* [14] респективно, док неке анализе прогнозирају чак 212 милијарди глобално распоређених уређаја 2020 [15]. Сходно томе, број IoT апликација енормно расте, а компанија *VisionMobile* предвиђа да ће број IoT програмера достићи 4,5 милиона до 2020. године [16], а њима су потребне ефикасне платформе на којима би развијали нове апликације.

Ефикасна обрада података генерисаних из толиког броја сензорских уређаја поставља низ пројектантских изазова. Подаци из сензорских мрежа имају веома динамичку природу, сензорски уређаји могу да буду привремено искључени, могу да се прикључују нови уређаји, квалитет сензорских очитавања може да варира услед ограничених ресурса уређаја, сензори могу бити мобилни и статички итд. Затим постоји проблем адекватног моделовања различитих сензора, њихових карактеристика и радних процеса, а посебно њихових релација са реалним објектима од интереса. Оптимални информациони модел мора да омогући интероперабилност између хетерогених сензорских мрежа и примену у широком апликативном домену [7,8].

Тренутна ситуација доста подсећа на почетке *Светски разапете мреже* (енг. *World Wide Web - WWW*) због недостатка ефикасних претраживача и платформи за интеграцију [8], односно архитектуре која би на адекватан начин интегрисала све сензоре и истовремено омогућавала испоруку сензорких података заинтересованим корисницима као и могућност да лако пронађу жељена сензорска мерења без знања о конкретним сензорима и њиховим карактеристикама. Због таквог стања, Европска Унија је инвестирала око 200 милиона евра у периоду од 2014. до 2017. године у развој и истраживање у области IoT [17], а посебно је 2016. године започета иницијатива *IoT European Platform Initiative (IoT-EPI)*[1] која обухвата 7 истраживачких и иновативних пројеката на тему развоја IoT платформи, а такође у оквиру посебног позива IoT-03-2017 - *R&I on IoT integration and platforms*, у току 2018. године започето је још 7 истраживачких пројеката на тему IoT интеграције и платформи[2].

У складу са претходном анализом текућег стања и прогноза о све већој актуелности проблема, циљ истраживања у склопу ове докторске дисертације представља анализа архитектура за интеграцију сензорских мрежа и њихових

[1] https://iot-epi.eu/
[2] https://cordis.europa.eu/programme/rcn/702035_en.html

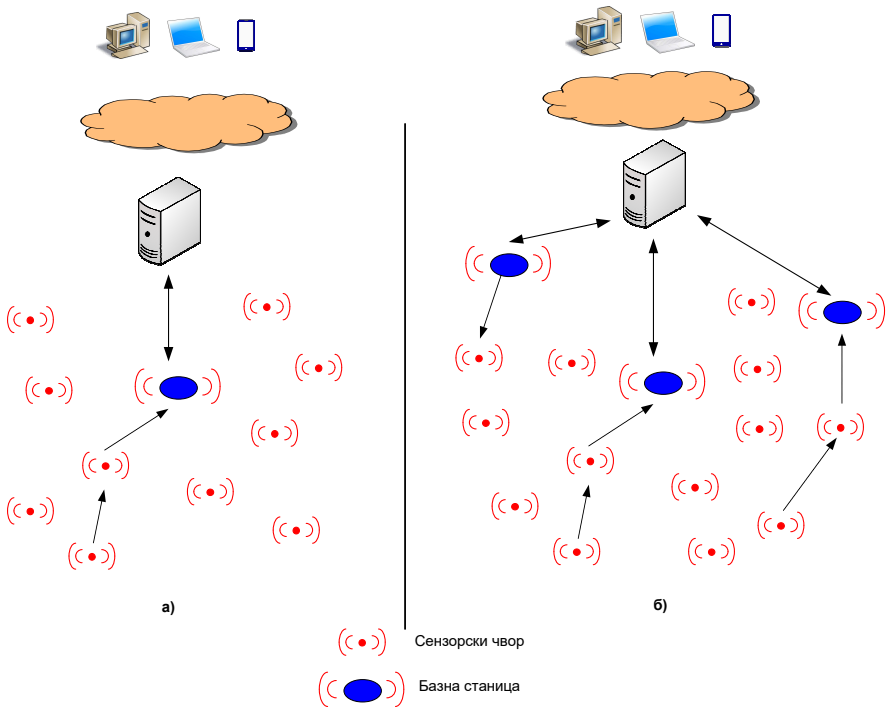
перформанси, посебно оних заснованих на семантичким технологијама. У оквиру рада се предлаже скалабилна архитектура заснована на семантици сензорских података која омогућује интеграцију хетерогених сензора и истовремено омогућава ефикасно претраживање и дохватање жељених сензорских очитавања, како најсвежијих, тако и оних из прошлости. Подразумева се да се за описивање техничких карактеристика сензора и сензорских очитавања користе семантичке веб технологије (енг. *Semantic Web*), конкретно *OWL* језик за дефинисање концепата у онтологијама, а *RDF* модел за представљање семантичких сензорских података. Предложена архитектура се заправо заснива на решењу за динамичко индексирање и ефикасно претраживање *RDF* тројки полу-структурираног скупа сензорских података у дистрибуираном окружењу. Претпоставља се окружење са великим бројем провајдера сензорских очитавања и такође великим бројем корисника. Претходна истраживања су показала да су најчешћи типови упита за сензорским очитавањима просторно-временски упити [6-8], тако да је њима посвећена посебна пажња и они су посебно оптимизовани.

Структура овог рада је следећа. У следећем поглављу, биће најпре дат преглед кључних технологија у области сензорских мрежа и семантичких веб технологија, неопходних за разумевање проблема који се обрађује. У поглављу 3, дат је преглед могућих архитектура за интеграцију сензорских мрежа које покривају распон од малог до великог степена скалабилности, а посебан нагласак је дат на анализи перформанси таквих архитектура у условима преноса порука сензорских података у реалном времену. У поглављу 4 је дата класификација и опис постојећих архитектура за семантички засновану интеграцију сензорских мрежа у светлу кључних пројектанских параметара, уз критички осврт на предности и недостатке постојећих решења. Затим у поглављу 5 је дат опис предложене архитектуре за семантичку интеграцију сензорских мрежа на бази дистрибуираног RDF складишта са подршком за чување динамички генерисаних семантичких сензорских опажања уз евалуацију перформанси. Поглавље 6 садржи закључак и анализира кључне доприносе дисертације и могући даљи правац истраживања.

2. ПРЕГЛЕД ТЕХНОЛОГИЈА СЕНЗОРСКИХ МРЕЖА И СЕМАНТИЧКИХ ВЕБ ТЕХНОЛОГИЈА

2.1. Бежичне сензорске мреже

Бежичне сензорске мреже спајају широк распон информационо-комуникационих технологија као што су хардвер, умрежавање, системски софтвер и програмске методологије [18, 19, 20, 21, 22]. Бежична сензорска мрежа (енг. *Wireless Sensor Network - WSN*) се дефинише као мрежа уређаја који су у стању да опажају околину и да размењују прикупљене информације о околини комуницирајући бежичним путем. Уређаји који чине мрежу се зову сензорски чворови и могу бити густо распоређени у неком простору, тако да се суседни чворови налазе врло близу. Сензорски чворови комуницирају размењујући поруке, при чему се користи прослеђивање порука преко више чворова (енг. *multi-hop*), што доводи до у општем случају бољег искоришћења енергије него код једноставног слања порука у једном скоку (енг. *single-hop*) између два сензорска чвора (видети слику 1).

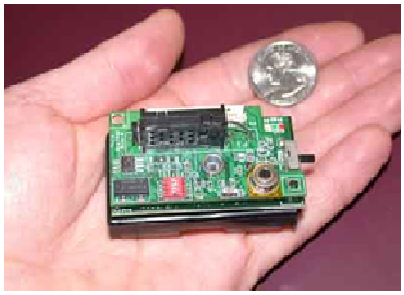


Слика 1 - Организација бежичне сензорске мреже: а) са једном базном станицом, б) са више базних станица

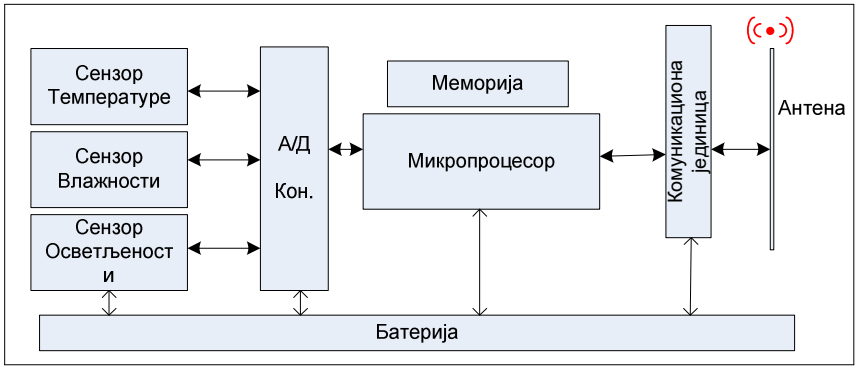
Подаци се са изворишта прослеђују ка сензорском чвору посебне улоге који се зове базна станица (енг. *sink*), који је обично повезан преко посебног пролазног уређаја (енг. *gateway*) са другим мрежама, а најчешће Интернетом, али који може и самостално да обрађује добијене податке од других чворова. Сензорски чворови могу (али не морају) бити истог типа, статични или мобилни, са знањем о својој локацији или не. Са децентрализованом организацијом добија се робустан и скалабилан систем који због постојања редундантности чворова превазилази робустност појединачних чворова који су склони отказима.

Како су описани уређаји мале површине и мале потрошње енергије, најчешће се напајају путем батерија. Такви уређаји могу бити распоређени у јако неприступачним областима, тако да је углавном тешко остварљива замена батерије. Са друге стране, типични захтеви су да сензорска мрежа буде оперативна у што је могуће дужем временском периоду. Тиме се од пројекатаната бежичне сензорске мреже захтева да остваре потребну функционалност уз минималну потрошњу енергије како би продужили рад целе мреже. Најлакши начин да се оствари минимална потрошња енергије је искључење већине компонената на једном сензорском чвору највећи део времена, а да се само у дефинисаним тренуцима те компоненте укључе ради извршења потребних операција. С обзиром да мрежу чине мањи или већи број уређаја који комуницирају између себе, потребно је развити адекватне протоколе за комуникацију како би се координисао рад више сензорских чворова. Такође пошто је комуникациона компонента мале потрошње то повлачи за собом и мали домет, тако да је у случају покривања простора веће површине неопходна колаборација већег броја чворова кроз прослеђивање порука у више корака, да би се измерена вредност у једном делу простора пренела до уређаја који је захтевао ту вредност, а није у домету сензорског чвора који је измерио вредност. Прослеђивање порука између сензорских чворова такође може да узме у обзир капацитет батерија и оптимизује се у складу са минималном потрошњом.

Бежичну сензорску мрежу чине сензорски чворови. Типичан сензорски чвор је приказан на слици 2. Он се састоји од четири компоненте и то: процесорске јединице која у себи садржи микропроцесор/микроконтролер са меморијом; сензорске јединице коју чини један или више сензора и пратећи аналогно-дигитални (А/Д) конвертор; комуникационе јединице са пријемником и предајником података; и јединице напајања. У зависности од конкретне примене, могу бити додате и друге компоненте као што су систем за глобално и локацијско позиционирање, генератор напајања, систем за померање сензорског чвора и други.



а)



б)

Слика 2 – а) Изглед сензорског чвора и
б) дијаграм организације једног сензорског чвора

Типичан режим рада сензорског чвора укључује неколико операција: сензор претвара мерену физичку величину или поље у аналогни сигнал, који се путем А/Д конвертора претвара у дигиталну вредност и прослеђује процесорској јединици; микропроцесор са пратећом меморијом извршава процесирање неопходно за обављање задатих активности у сарадњи са другим сензорским чворовима; комуникациона јединица повезује сензорски чвор са осталим сензорским чворовима преносом пакета података бежичним

путем. Паралелно, јединица напајања обезбеђује извор енергије за рад осталих подсистема на сензорском чвору.

Извор напајања - Пошто су сензорски чворови углавном неприступачни, време рада сензорске мреже директно зависи од времена трајања јединице напајања у сензорским чворовима. Захтеви за малом димензијом сензорског чвора, имплицирају да је јединица напајања ограничавајући ресурс и зато су опремљени лимитираним извором напајања (<0.5 Ah, 1.2V). У случају отказа неколико чворова услед нестанка напајања, може доћи до значајне промене топологије мреже што захтева њену реорганизацију и поновно рутирање пакета. Због тога управљање потрошњом енергије и њено чување игра значајну улогу тако да истраживачи развијају протоколе и алгоритме за сензорске мреже који су свесни потрошње енергије. Потрошња напајања се може поделити у три домена: сензорско опажање, комуникација и процесирање података. У сваком од ових домена може се остварити енергетска ефикасност: одабиром хардверских компоненти са малом потрошњом, на комуникационом нивоу путем одговарајућих протокола и технике рутирања, све до апликативног нивоа где се користе посебни алгоритми обраде података.

Сензорска јединица: Сензори који се уграђују у сензорске чворове су извор података о средини у којој се налазе [23]. Они се базирају на особини неких материјала да им се електричне карактеристике мењају ако су изложени различитим утицајима околине. Такве промене су предвидљиве у извесном распону што омогућава производњу сензора са познатим карактеристикама у одређеним условима. Рецимо, термистор је отпорник променљиве отпорности која се мења са променом температуре. Фотоћелије и детектори магле функционишу слично, с тим да је материјал постављен тако да му се отпорност мења под утицајем фотона који га погоде, односно под утицајем влаге. Сензори генеришу аналогни, тј. континуални електрични сигнал (напонски или струјно одређен), тако да је неопходна употреба А/Д конвертора да би се рецимо напонски сигнал претворио у дигитални запис који је даље могуће обрадити и складиштити помоћу микроконтролера.

Развијене су доста софистициране структуре за мерење и детекцију разних физичких величина. Такве структуре такође троше неколико миливата за свој рад и неопходно је да буду укључене само делић времена. Такође су развијена екстремно ефикасна A/D конверторска кола, тако да је сензорска јединица у сензорском чвору сличне потрошње као и микроконтролер.

Нарочиту пажњу данас привлаче микро-електромеханички системи (MEMC) који могу да се користе за опажање разноврсних физичких величина, хемијских концентрација и утицаја средине, јефтино и ефикасно. У процесу производње, транзистори се могу распоредити на силицијуму у виду минијатурних механичких облика, тако да при утицају гравитационе силе или убрзања, долази до промене у карактеристици материјала, што се може регистровати, појачати и дигитализовати. Први комерцијални MEMC сензор, акцелерометар, користи се у аутомобилској индустрији за активацију ваздушних јастука. Прецизност MEMC акцелерометра је довољна за већину примена уз цену од неколико долара, за разлику од високопрецизних пиезо-електричних акцелерометара који коштају стотине долара. Са масовном производњом MEMC сензора, може се очекивати побољшање њихове прецизности уз очување ниске цене коштања.

Процесорска јединица: Иако сензорски чворови имају све моћније процесорске и меморијске капацитете, процесорска јединица је и даље ограничени ресурс. У сензорске чворове се типично уграђују једноставни микроконтролери са примарним циљем да буду што енергетски ефикаснији, пре него да садрже већи скуп функционалности. Распон фреквенције уграђених микронтролера иде од 4MHz до око 200MHz, а имају потрошњу реда једног миливата при раду на око 10MHz. Током припремног режима рада када је већина компоненти на чипу искључена, потрошња се додатно смањује на око једног микровата. Ако узмемо да је микроконтролер активан у 1% времена, излази да је његова просечна потрошња свега неколико микровата [19]. На пример, за један прототип сензорског чвора [24] користи се 4 MHz Atmel 90LS8535 8-битни микроконтролер Харвардске архитектуре са 16-битним адресним

простором и 32 8-битна регистра опште намене. Садржи 8KB програмске флеш меморије и 512 бајтова SRAM за податке. Специјално развијени оперативни систем за *WSN*, *TinyOS*, који се користи на овом сензорском чвору, заузима 3500 бајтова, тако да остаје расположиво 4500 бајтова. Други пример процесорске јединице је такође прототип μAMPS са 59–206MHz SA-1110 микропроцесором који извршава вишенитни оперативни систем μ-OS. Потрошња напајања услед обраде података дата је у литератури [18] и може се представити формулом:

$$P\text{p} = CVdd^2f + VddIe^{Vdd/n'Vt} \quad (1)$$

где је *C* укупна прекидачка капацитивност, *Vdd* је напон напајања, а *f* је фреквенција прекидања. Други члан представља изгубљену снагу услед струје цурења. Ова формула може да сугерише да је сензорском чвору потребна извесна енергија да пређе из једног у други радни режим тако да често искључење односно укључење процесора не доноси увек уштеду енергије због потрошње енергије у фази прелазног режима. Највећи потрошач енергије на микроконролерима ниске потрошње је заправо меморија, без обзира на то што такви микроконтролери имају лимитиране складишне капацитете. Типично, то је мање од 10KB RAM-а за податке и мање од 100KB ROM-а за програмску меморију, а ако је потребна додатна меморија, она се додаје као засебан чип и то је обично флеш меморија већег капацитета реда величине 1MB. Иако је ово јако мали капацитет у поређењу са стандардним капацитетима код PC рачунара, уграђена меморија заузима већину површине на чипу и троши добар део расположиве енергије.

Комуникациона јединица: Комуникациона јединица на сензорском чвору је типично базирана на радио-фреквенцијској (*RF*) компоненти. RF комуникација захтева модулацију, филтрирање и демодулацију са мултиплексираним колима, што их чини комплексним и скупим. Потребна енергија за радио комуникацију се рапидно повећава са повећавањем удаљености, при чему препреке у виду људи и физичких објеката додатно слабе сигнал. Пошто су антене сензорских чворова близу земље, губитак везе

између два сензорска чвора може бити врло често и слабљење је сразмерно четвртом степену удаљености између тих чворова. Да би се покриле веће удаљености коришћењем комуникационих компоненти малог домета, бежична сензорска мрежа је тако организована да се пренос информација врши преносом у више корака (енг. *multi-hop*) односно прослеђивањем порука преко више сензорских чворова. Као што је раније поменуто, преносом у више корака се постиже боље искоришћење енергије него код једноставног слања порука у једном кораку. Ово се може доказати на следећи начин. Наиме, нека је у *N-hop* мрежи укупна раздаљина приликом преноса *Nr*, где је *r* удаљеност за један скок. Минимална улазна снага на пријемном чвору за дату грешку преноса је $P_{receive}$, а снага на предајном чвору је P_{send}. Модел RF слабљења при земљи је дат са формулом:

$$Preceive \propto \frac{Psend}{r^{\alpha}} \Rightarrow Psend \propto r^{\alpha} Preceive \tag{2}$$

где је *r* удаљеност за дати пренос, а α је коефицијент RF слабљења са вредношћу у распону од 2 до 5, у зависности од ефекта интерференције и рефлексије. Одавде се изводи да је мањи утрошак предајне снаге у случају са преносом преко N корака него у једном покушају у случају једнаке укупне удаљености преноса *Nr*:

$$\eta_{rf} = \frac{Psend(Nr)}{N \cdot Psend(r)} = \frac{(Nr)^{\alpha} Preceive}{N \cdot r^{\alpha} Preceive} = N^{\alpha-1} \tag{3}$$

Ова формула даје да се са већим бројем чворова добија већа уштеда у предајној снази, али треба приметити да је овде занемарен утрошак снаге пратећих RF компонената као и ефекат кашњења, што се при пројектовању мора узети у обзир како би се нашао оптимални баланс у погледу цене и енергетске ефикасности.

Следећи могући начин комуникације сензорских чворова је коришћењем инфрацрвеног опсега, пошто је та комуникација отпорна на сметње од електричних уређаја и није потребно лиценцирати коришћени фреквентни

опсег. Предајник није скуп и пуно данашњих уређаја попут лаптопова, мобилних телефона већ имају уграђен инфрацрвени интерфејс. Главна препрека коришћењу инфрацрвених уређаја за везу између сензорских чворова је захтев за оптичком видљивошћу између предајника и пријемника што је често неостварљиво у сензорским мрежама. Такође постоје примери оптичке комуникације између сензорских чворова поготово код сензорских чворова ултра ниске потрошње, али при тој комуникацији је такође присутан захтев оптичке видљивости између предајника и пријемника.

Пошто се у WSN преносе мали пакети података, ниска је брзина података, тако да веома често долази до преузимања фреквенције због кратких домета комуникације, а и због малог процента времена када је укључена комуникациона јединица.

Комуникациони подсистем на сензорском чвору у *WSN* троши око 20 миливата уз типичан домет десетине метара. Комуникација представља операцију са највећом потрошњом енергије на сензорском чвору, при чему се за слање једног бита информације потроши енергије исто као за 100 до око 10000, а типично 1000 програмских инструкција [19]. Из тог разлога се где год је то могуће врши обрада сирових података на сензорским чворовима, како би се слали агрегатни или само неопходни подаци.

2.2. Комуникациона инфраструктура сензорских мрежа

У комуникационој архитектури *WSN* код сензорских чворова и базних станица, користи се стек протокол базиран на *OSI* (*Open Systems Interconnection*) моделу [21]. Стек протокол комбинује знање о потрошњи енергије и рутирању порука, интегрише податке са мрежним протоколом, комуницира кроз бежични медијум ефикасно трошећи снагу и промовише кооперацију између сензорских чворова. Протокол стек се састоји од *физичког слоја*, *слоја везе података*, *мрежног слоја*, *транспортног слоја* и *апликативног слоја*.

Физички слој регулише технике за потребе једноставне али робустне модулације што укључује генерисање носеће фреквенције, детекцију сигнала, модулацију, и енкрипцију података предаје и пријема сигнала. Избор одговарајуће модулације сигнала је критичан за поуздану комуникацију у сензорским мрежама. Најчешће су у употреби бинарна и М-арна модулација, а такође и *UWB* (*Ultra-wideband*) или импулсни радио (*IR*), посебно због ниске потрошње и једноставне имплементације пријемника. *UWB* се базира на преносу путем основног спектра и не захтева међуфреквенције и носеће фреквенције. Користи се пулсна позициона модулација. Главна предност *UWB* је отпорност на рефлексију.

Слој везе података је задужен за мултиплексирање тока података, детекцију оквира података, приступ медијуму и контролу грешака. Он обезбеђује поуздану комуникацију од-тачке-до-тачке, односно од-тачке-до-више тачака. Овај слој укључује протокол за контролу приступа медијуму (енг. *Medium Access Control - MAC*) односно за координацију приступа и пренос преко заједничког медијума између сензорских чворова и успостављање комуникационе везе за пренос података. У пракси доминирају два приступа *MAC* протокола за *WSN*: протокол на бази резервација (фиксна алокација) и протоколи на бази утркивања (енг. *contention-based*).

Мрежни слој у бежичним сензорским мрежама обухвата посебне протоколе за рутирање пакета између базних станица и сензорских чворова, што некада укључује прослеђивање порука у више корака. Протоколи треба да задовоље низ специфичности и ограничења тих мрежа као што је немогућност глобалног адресирања сензорских чворова, тј. додељивање глобалних идентификатора или адреса; затим ток сензорских података је углавном усмерен од сензорских чворова из разних региона ка базним станицама; генерисани саобраћај поседује доста редундантности пошто више истих типова сензора мере неку физичку величину или појаву блиско постављени. Таква редундантност може бити искоришћена од стране протокола за рутирање да би се уштедела енергија или искористио пропусни опсег.

Постоји неколико принципа и техника на којима су базирани протоколи за рутирање, а то су: енергетска ефикасност, рутирање на бази атрибута, агрегација података, локацијско рутирање, комуникација преко вишеструких путања, рутирање са испуњењем жељеног квалитета сервиса и други.

Транспортни слој је одговоран за омогућавање приступа сензорској мрежи од стране Интернета или других мрежа. Задатак овог слоја је да обезбеди поузданост и избегне загушење протока података услед постојања више протокола усмерених од стране базне станице ка кориснику и обрнуто. Међутим, развој транспортног слоја у бежичним сензорским мрежама представаља прави изазов, због ограничења сензорских чворова по питању напајања и меморије, као и скупе операције потврде пријема. *TCP* као транспортни протокол није погодан за употребу у сензорским мрежама, јер се у њима не ради глобално адресирање од краја-до-краја комуникације, већ на бази атрибута. Такође, енергетски је ефикаснији приступ скок-по-скок, него од тачке-до-тачке. Обично се од Интернет корисника до базне станице користи или *TCP* или *UDP*, а од базне станице до сензорских чворова искључиво *UDP* налик протоколи.

Апликативни слој обухвата низ протокола за испоруку података из сензорских мрежа до крајњих апликација. У следећем поглављу биће дата дубља анализа апликативних протокола који се базирају на приступу са *повлачењем података (pull-based)*, *гурањем података (push-based)* и *прослеђивање порука на бази објави-претплати* приступа.

2.3. Комуникациони стандарди

Разне организације и индустријски конзорцијуми су креирали стандарде за *WSN*, са дефинисаним функцијама и протоколима потребним за рад сензорских чворова у различитим мрежама. Кључни параметар у предложеним стандардима је ниска потрошња енергије. Предложени стандарди не адресирају све слојеве у *OSI* моделу, тако да се у конкретним

применама користе различити стандарди на разним нивоима комуникационе архитектуре.

2.3.1. IEEE 802.15.4

IEEE 802.15.4 [25, 26] је стандард дизајниран за персоналне бежичне мреже мале брзине преноса (енг. *Low Rate Wireless Personal Area Networks - LR-WPAN*), мале комплексности, ниске цене и мале потрошње енергије, са посебним фокусом на бежичне сензорске мреже. Намењен је за коришћење у апликацијама које захтевају малу комплексност бежичне комуникације на кратким растојањима у области индустријске примене, пољопривреде, саобраћаја, паметних зграда, медицинског надзора итд. Стандард адресира физички слој и слој линка података, а претпоставља тек повремен пренос мале количине података што имплицира мали однос рад-пауза. Додатно, структура пакета је тако пројектована да се додаје минимална количина додатних података у односу на садржај пакета података.

IEEE 802.15.4 дозвољава креирање две врсте топологије мреже: *топологију звезде* и тзв. *peer-to-peer топологију*. Дефинисана су два типа уређаја: потпуно функционални уређај (енг. *Full Function Device - FFD*) и уређај са умањеним функцијама (енг. *Reduced Function Device - RFD*). У зависности од топологије, *FFD* могу имати улогу координатора или мрежног уређаја, док *RFD* имају улогу мрежних уређаја. Код *топологије звезде*, комуникација се обавља између мрежних уређаја и једног централног контролера који се зове *PAN* координатор (*FFD* уређај). Ова топологија се предвиђа у случају када је потребно покрити малу површину и са малим кашњењима у преносу. У случају *peer-to-peer топологије*, сваки мрежни уређај може да комуницира са било којим другим мрежним уређајем преносом у више корака. Постојање преноса у више корака, подразумева чување табела за рутирање. Користи се када је потребно покрити веће области и када кашњење није критичан параметар.

У физичком слоју IEEE 802.15.4 подржава две опције које се разликују у фреквенцијском опсегу, а обе су реализоване на бази технике директне секвенце широког спектра. Опција ниског опсега обухвата фреквенције од 868MHz и 915MHz уз коришћење *бинарне фазно померене модулације (Binary Phase Shift Key - BPSK)*. За примене у Европи је предвиђена фреквенција од 868MHz са једним каналом и брзином преноса од 20Kb/s, а за Северну Америку је предвиђена фреквенција 915MHz ISM опсега која нуди 10 канала са брзином преноса од 40kb/s. Опција високог опсега обухвата рад на фреквенцији у опсегу око 2.4GHz ISM са могућношћу рада широм света, уз коришћење *квадратурне фазно померене модулације са офсетом* (*Offset Quadrature Phase Shift Key - O-QPSK*). Овај опсег обухвата фреквенције од 2.4GHz до 2.483GHz и нуди 16 канала са празнином од 5MHz, уз брзину преноса од 250kb/s. Структура пакета (или у терминологији стандарда јединица података физичког протокола – *PHY Protocol Data Unit - PPDU*) у обе опције је иста и садржи следећа поља: преамбулу (4 бајта), почетак делимитера пакета (1 бајт), дужина пакета (1 бајт) и поље садржаја односно јединица података физичког сервиса (*PHY Service Data Unit - PSDU*) промењљиве дужине од 2 до 127 бајтова.

2.3.2. ZigBee

ZigBee стандард[3] дефинише мрежни и апликативни слој изнад IEEE 802.15.4 стандарда за персоналне бежичне мреже мале брзине преноса (енг. *LR-PAN*). *ZigBee* уређаји троше јако мало енергије и могу да раде помоћу мале батерије неколико година. Стандард је креиран од стране *ZigBee* алијансе и главни допринос је могућност креирања меш мрежне топологије умрежавањем од стотине до хиљаде уређаја. Меш топологија омогућава реконфигурисање око блокираних путања између чворова обилажањем у више корака преко других чворова. У стандарду се дефинише три типа уређаја: *ZigBee координатор* који је заправо PAN координатор из IEEE 802.15.4 стандарда, затим *ZigBee рутер* и *ZigBee крајњи уређај*.

[3] http://www.zigbee.org/

Такође, *ZigBee* стандард дефинише *топологију стабла* као посебан случај *peer-to-peer топологије* дефинисане у IEEE 802.15.4. Ова топологија је организована у виду хијерархијског стабла, при чему чворови на неком нивоу шаљу податке ка чворовима на нижем нивоу, све док се не стигне до *ZigBee* координатора који је у корену стабла. Постоји само један *ZigBee* координатор, који је обично базна станица, а у случају организације мреже са више базних станица, присутно је више *ZigBee* координатора, што се своди на унију више раздвојених стабала. *ZigBee* координатор иницира формирање мреже, чува информације и може да изврши спрегу неколико мрежа. *ZigBee* рутери (морају бити *FFD* уређаји) повезују групе уређаја заједно и омогућују пренос у више корака преко уређаја. *ZigBee* крајњи уређаји (могу бити *RFD* или *FFD* уређаји) садрже сензоре, актуаторе и контролере, врше прикупљање података и комуницирају са рутерима или координатором.

2.3.3. 6LoWPAN

IETF (*Internet Engineering Task Force*) организација је 2007. године предложила отворени стандард *6LoWPAN (IPv6-based Low power Wireless Personal Area Networks*) [27] који омогућава пренос IPv6 пакета преко IEEE 802.15.4 мреже у условима ниске потрошње, мале брзине преноса и мале цене мреже. Употребом овог протокола, уређаји ниске потрошње могу директно да комуницирају са IP уређајима преко IP базираних протокола и да користе све предности IP комуникације и управљања. *6LoWPAN* стандард обезбеђује *адаптациони слој*, *нови формат пакета* и *управљање адресама*. *Адаптациони слој* се користи за прилагођавање величине IPv6 пакета на знатно мању величину оквира у IEEE 802.15.4. То се остварује компресијом 60 бајтова великог IPv6 заглавља у заглавље величине 7 бајтова и фрагментацијом 1280 бајтова дугачког IPv6 пакета у мање пакете величине 127 који одговарају IEEE 802.15.4 оквиру. Механизам за управљање адресама је задужен за динамичко креирање 16-битних кратких адреса уређаја који се користе у комуникацији, чиме је омогућено хијерархијско рутирање. За управљање *LR-WPAN* мрежом, *6LoWPAN* користи *SNMPv3 (Simple Network Management Protocol*) протокол како би се омогућило управљање густо постављеним јефтиним уређајима.

Главна разлика између *6LoWPAN* и *ZigBee* стандарда је могућност рада са IP уређајима директно у случају *6LoWPAN* стандарда, док *ZigBee* уређаји захтевају постојање IEEE 802.15.4/IP пролазног уређаја како би имали ту могућност. Са друге стране, ако у апликацији нема потребе за комуникацијом са IP уређајима или је величина пакета мала, боље перформансе постиже мрежа са *ZigBee* уређајима.

2.3.4. Bluetooth

Bluetooth [28, 29] је технологија за бежичну комуникацију на кратким растојањима са намером да замени жични пренос у случају *WPAN (Wireless Personal Area Networks)*. Главне одлике *Bluetooth* протокола је робустност, мала потрошња и ниска цена. Многе особине протокола су опционе, тако да постоје варијације у постојећим имплементацијама. Физички слој *Bluetooth* стандарда функционише у области нелиценцираног ISM опсега на фреквенцији око 2,4GHz, тачније од (2400, 2483,5)MHz. На располагању је 79 канала са размаком од 1MHz. Систем користи скоковити фреквенцијски предајник са брзином мењања фреквенције од 1600 скокова/s, зарад избегавања сметњи и нестајања сигнала. Користи се *бинарна фреквенцијски померена модулација* (*Binary Frequency Shift Keying - BFSK*) због смањења комплексности предајника, техника корекције грешака прослеђивањем (*Forward Error Correction - FEC*), уз брзину преноса од 1MB/s. Чворови су организовани у тзв. пикомреже, које су управљане од једног главног чвора и имају до седам подређених чворова. Физички канал је издељен у временске слотове трајања 625μs. *Bluetooth* пружа ефекат потпуне-двосмерне комуникације (енг. *full-duplex*) кроз коришћење временски подељене дуплексне (*Time Division Duplex - TDD*) шеме.

2.3.5. WirelessHART

WirelessHART стандард [30] је настао 2007. године као проширење популарног *HART* стандарда за жичну комуникацију, намењен за индустријске потребе и то за процесе мерења и управљачких апликација. Базира се на IEEE 802.4.15 2.4GHz стандарду. Пројектован је да буде поуздан, безбедан и енергетски

ефикасан. Подржава решеткасту тополгију мреже и звезда топологију као и њихову комбинацију, скоковито мењање канала и временски синхронизоване поруке. Комуникација унутар мреже је безбедна са енкрипцијом, верификацијом, аутентификацијом и управљањем кључевима.

2.4. Оперативни системи за уређаје бежичних сензорских мрежа

Оперативни систем за платформе бежичних сензорских мрежа треба да омогући управљање ресурсима и обезбеди ефикасне програмске апстракције и интерфејсе за широк скуп потенцијалних апликација сакривајући специфичности хардвера на коме се извршава. Дизајн одговарајућег оперативног система зависи од карактеристика платформи бежичних сензорских мрежа, а то су пре свега јако ограничени ресурси, велика динамичност и немогућност приступа уређајима након распоређивања. Ефикасан развој апликација је могућ уз одговарајуће алате за развој који помажу ефикасно тестирање и исправљање грешака, тако да је подршка ефикасних алата у тесној вези са оперативним системом. Због тога се дизајн оперативног система огледа у ефикасном решавању изазова на нивоу сензорских чворова што треба да омогући портабилност апликација и ефикасност управљања ресурсима, затим на мрежном нивоу ефикасност коришћења комуникационог пропусног опсега и омогућавање скалабилности апликација и на крају кроз подршку развојног окружења путем ефикасних алата. У наставку ће бити дат кратак опис неких од најпопуларнијих оперативних система за уређаје *WSN*.

2.4.1. TinyOS

TinyOS [31] је бесплатан оперативни систем отвореног кода за бежичне сензорске мреже заснован на компонетном моделу послова уз омогућено конкурентно извршавање послова по моделу изврши-до-завршетка уз могуће истискивање. *TinyOS* је најпре развијен на Универзитету Беркли у Калифорнији, а временом је заједница прерасла у конзорцијум *TinyOS* Алијансе.

Основна архитектура *TinyOS* оперативног система је монолитне структуре, са компонентним моделом у позадини, а са статичком сликом меморије у време извршавања, тј. нема динамичке алокације меморије. Извршни модел је вођен догађајима, а конкурентност се заснива на извршавању команди, догађаја и задатака. Једна компонента декларише које команде подржава, затим догађаје које сигнализира и скуп задатака које треба извршити. Свакој компоненти се додељује статички алоциран оквир меморије фиксне дужине у чијем контексту се све функције обављају и чувају сви подаци [24]. Постоји неколико предефинисаних системских компоненти које имају улогу драјвера за уређаје (за комуникацију и приступ сензорима). Декларацијом компоненти и њихових веза креира се конфигурација на нивоу апликације са хијерархијом компонената, при чему хардвер представља најнижи ниво компонената. Ток извршавања почиње од компоненти вишег нивоа које позивају функције односно издају команде компонентама нижег нивоа, а након завршетка извршавања компоненте нижег нивоа врше асинхрону нотификацију компонената вишег нивоа путем сигнализације догађаја. Команде и догађаји могу део свог извршавања да издвоје у целине које се зову задаци (енг. *tasks*) и они се распоређују за извршавање од стране *TinyOS* распоређивача по *FIFO* принципу. Могуће је истискивање задатака у редоследу извршавања од стране догађаја, тако да догађаји који се дуго извршавају требају бити организовани у виду више задатака и на тај начин представљају операције у више фаза. Атомске секције служе за заштиту од конфиликта услед утркивања током истискивања задатака.

Промена извршавања *TinyOS* програма може бити изведена само рекомпајлирањем и затим репрограмирањем сензорског чвора са новом сликом меморије, при чему се губе дотадашње статусне информације. *TinyOS* омогућава управљање потрошњом напајања кроз адекватан програмски интерфејс којим се омогућава управљање режимом рада и процесора и комуникационе јединице. О ефикасној потрошњи напајања води рачуна и

TinyOS распоређивач послова, тако што активира режим мировања уколико нема послова за извршавање у реду чекања.

Бежична комуникација је подржана кроз *MAC* слој који се заснива на IEEE 802.15.4 физичком слоју и формату оквира. За системски развој и развој апликација се користи програмски језик *nesC* (назив изведен од *network embedded systems C)*, који је дијалекат програмског језика *C*, са стриктним форсирањем раздвајања конструкције и композиције програмских компонената. Уз *TinyOS* је развијено симулационо окружење *TOSSIM* [31], које омогућава симулацију апликација у хомогеном мрежном окружењу. Добра особина је да исти програмски код може бити коришћен током симулације и у реалним платформама. Симулација се одвија на нивоу бита, тако да је могуће симулирати протоколе на ниском нивоу.

2.4.2. Contiki

Contiki [32] је бесплатан оперативни систем отвореног кода развијен за мрежне системе са ограниченом меморијом, укључујући и бежичне сензорске мреже. Настао је на Шведском институту за рачунарске науке, где се и даље координише његов развој.

Архитектура *Contiki* оперативног система је модуларна на нивоу микрокернел дизајна. Срж овог ОС чине сервиси, који се састоје од интерфејса и имплементације. Примери сервиса су комуникација, драјвери за уређаје, обрада сензорских података и слично. Извршни модел је примарно вођен догађајима, али постоји подршка за коришћење програмских нити преко апликативне библиотеке. Догађаји могу бити синхрони, у ком случају се извршавају без одлагања, или асинхрони, који могу да се изврше у неком каснијем тренутку. Добра особина *Contiki* ОС-а је да се имплементације сервиса могу заменити у току извршавања, пошто су апликације повезане са сервисима преко библиотека за везу. У том случају, није потребно репрограмирати комплетан систем, већ само жељену имплементацију сервиса, при чему треба водитити рачуна о алокацији меморије због зависности

програмског кода од места у меморији и немогућности повећања величине кода.

Није експлицитно подржано управљање напајањем на вишем нивоу, већ је апликацијама доступан приступ неким интерним структурама, попут реда са догађајима. На тај начин, апликација може да активира режим мировања, уколико нема догађаја који чекају на обраду. *Contiki* подржава флексибилан мрежни стек протокол, од *MAC* слоја до *IEEE 802.15.4* и *UDP* имплементација, иако се фаворизују IP базирани протоколи тако да се користи као примарни пример имплементације *6LoWPAN* протокола. За системски развој и развој апликација користи се *C* програмски језик. Постоји и симулационо окружење као кориснички процес на *BSD Unix* оперативном систему, где је сваки симулирани сензорски чвор представљен као посебан кориснички процес.

2.4.3. FreeRTOS

FreeRTOS[4] је оперативни систем за рад у реалном времену намењен за уграђене системе (енг. *embedded systems*), без неког специфичног адресирања *WSN*. Написан је у језику *C*, компактан је и једноставан. Користи монолитни приступ, са омогућеним распоређивањем послова са и без истискивања. Сваки посао се извршава унутар сопственог контекста, не зависећи од других послова унутар система или од распоређивача послова. Омогућене су тзв. лаке ко-рутине за активности које се кратко извршавају. Комуникација и синхронизација између послова се остварује путем редова, семафора и региона са међусобним искључивањем.

За жичну комуникацију се користе варијација TCP/IP имплементације протокола са различитим ограничењима, а нема посебне подршке за бежичну комуникацију. Такође, нема посебне подршке за управљање напајањем. За системски развој и развој апликација се користи програмски језик *C* као и асемблер. Расположива су симулациона окружења на *Windows* и *Linux* оперативном систему, при чему се симулира само један уграђени систем.

[4] https://www.freertos.org/

2.5. Семантичке веб технологије

Иницијатива која је имала за циљ да прошири веб са додатним семантичким подацима, односно значењем информација на вебу зове се *Semantic Web* [35]. Идеја је да се значење информација на вебу формално дефинише кроз онтологије, тако да њиховим коришћењем машине могу ефикасно да интерпретирају податке и повезују их са другим сродним подацима. *Semantic Web* иницијатива је подржана од стране Конзорцијума за светску разапету мрежу (енг. *World Wide Web Consortium - W3C*), а такође је од стране истог конзорцијума иницран развој низа технологија које омогућавају реализацију *Semantic Web* идеје. Овде ће бити дат кратак преглед неких кључних технологија.

2.5.1. RDF

Једна од кључних технологија семантичког веба је *Resource Description Framewrok* (*RDF*) [36] и служи за описивање ресурса на вебу и њихових релација, при чему ресурси могу бити разног типа као сто су веб страна, особе, ствари, неки апстрактни појмови и друго. *RDF* је модел података графовске структуре, без шеме података, са сопственим описом, односно ознаке унутар графа описују податке у том графу. *RDF* модел се често назива RDF граф, у коме је основни елемент тројка у форми (*субјекат, предикат, објекат*). Свака тројка у моделу формира грану <*предикат*> у графу, тако што се повезује чвор <*субјекат*> са чвором <*објекат*>. Конкретно, чињеница да је Иво Андрић писац књиге На Дрини ћуприја може се представити тројком:

<“*Ivo Andrić*” *jePisac* “*Na Drini ćuprija*“>.

Формална дефиниција *RDF* тројке је следећа:

Дефиниција 1: *Дат је скуп URI* референци ***U***, скуп празних чворова ***B*** и скуп литерала ***L***. *Тројка са особином (s, p, o)*$\in$ (**U** $\cup$ **B**) $\times$ **U** $\times$ (**U**$\cup$**B**$\cup$**L**) *назива се RDF*

тројка. Елементи скупа $\boldsymbol{U} \cup \boldsymbol{B} \cup \boldsymbol{L}$ се називају *RDF* термини. Скуп *RDF* тројки се назива *RDF* граф.

Због своје флексибилне структуре, *RDF* може служити за представљање неструктурираних података, који се често генеришу на вебу, или унутар социјалних мрежа, јер у тим случајевима шема података није позната унапред услед разноврсности типа и особина тих података. Са друге стране, *RDF* може такође служити за представљање структурираних као и полуструктурираних података, које потичу из релационих база података или *XML*-а.

RDF омогућава да се различити извори података повежу тако што би се само додало неколико тројки које специфицирају односе између концепата из тих *RDF* графова. То би било јако тешко изводљиво у системима за управљање релационим базама података (енг. *Relational Data Base Management Systems - RDBMS*), пошто би требало извршити промену шеме базе података и такође успоставити адекватно референцирање.

Постоји неколико предложених формата којима се специфицира синтакса серијализованих *RDF* података [37]:

- **N-Triples** [38] је формат препоручен од стране WWW конзорцијума, једноставан за парсирање и представља поједностављену верзију општијег и комплекснијег формата *Notation 3 (N3)* [39]. *RDF* тројке се пишу као размаком раздвојени *RDF* термини, са завршном тачком. Заграде (<>) означавају *URI (Uniform Resource Identifier)*, а наводници (“”) литерале, док идентификатори празних чворова почињу са ‘_:’.
- **RDF/XML** дефинише XML синтаксу за представљање *RDF* тројки [40]. Формат најпре садржи дефиницију простора имена, затим `rdf:about` конструкцију за субјекат и низ предиката са одговарајућим објектима.
- **Terse RDF Triple Language (Turtle)** [41] је креиран да би побољшао изражајност *N-Triples* формата. Такође, *Turtle* синтакса се користи код дефинисања графовских упита у језику SPARQL.

Пример *Turtle RDF*-а:

```
@prefix ff: <http://www.superliga.org#>
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntx-ns#>
ff:CrvenaZvezda rdf:type ff:FudbalskiKlub .
```

2.5.2. Онтологије и језици за опис онтологија – OWL и RDFS

Додатна семантика података представљених *RDF* моделом података је садржана у онтологијама, које дефинишу концепте унутар неког домена и њихове релације. Онтологије практично специфицирају речник података, односно термине који се користе и релације између њих. Термини су заправо класе објеката, а релације хијерархије тих класа. Осим ових особина, онтологије могу укључивати информације о особинама класа, ограничењима над вредностима атрибута, логичке релације између објеката и друго. Унутар иницијативе семантичког веба, предложено је неколико језика за представљање онтологија, а најпопуларнији су *RDFS (Resource Description Framework Schema)* [42] и *Web Ontology Language (OWL)* [43] који је препоручен од стране WWW конзорцијума.

RDFS [42] је креиран над *RDF*-ом и додаје неке основне конструкције са циљем да се апликацијама пружи интерпретација *RDF* података. Основна конструкција се односи на дефинисање типова субјеката, кроз дефинисање класа и подкласа и на тај начин се омогућава креирање хијерархије класа. Пример за основне конструкције:

```
ff:FudbalskiTim rdf:type rdfs:Class .
ff:Igrac rdf:type rdfs:Class .
ff:Napadac rdf:type rdfs:Class .
ff:OdbrambeniIgrac rdf:type rdfs:Class .

ff:Napadac rdfs:subClassOf bb:Igrac .
ff:OdbrambeniIgrac :subClassOf bb:Igrac .
```

Додатна проширења се односе на дефинисање домена и опсега за предикате, чиме се дефинише који се типови класа могу додати као субјекти или објекти

за одређене предикате. *RDFS* тројке у том случају имају вредност предиката *rdfs:domain* и *rdfs:range*.

OWL [43] је препоручен од стране WWW конзорцијума и иако је знатно моћнији, потпуно је компатбилан са *RDFS* језиком. *OWL* додатно дефинише извесне особине које могу имати предикати. На тај начин се омогућује извођење закључака над подацима. То практично значи да се не мора експлицитно навести особина за сваку инстанцу неке класе, већ је довољно дефинисати особину за дату класу, а остале информације ће бити изведене.

RDF омогућује да се концепти специфицирају и повезују унутар графа података. Тиме се узрокује да се *RDF* тројке и *RDFS/OWL* тројке могу мешати и могу се поделити у *шематске тројке* и *тројке података*. *Шематске тројке* се изражавају језицима за опис шема односно са *RDFS* и *OWL* и дају информације о шеми. *Тројке података* описују податке о инстанцама и представљају се *RDF* изразима.

2.5.3. Упити над RDF подацима и SPARQL

RDF складиште (енг. *RDF Triple Store*) је задужено за управљање *RDF* подацима и представља систем за смештање, дохватање, претрагу, додавање, ажурирање и брисање *RDF* тројки у неком перзистентном медијуму. Претрага *RDF* података се врши користећи концепт *Узорака Основног Графа* (енг. *Basic Graph Patterns – BGP*) који је низ повезаних упита креираних за *RDF* модел података.

Као пример узорак тројке `?s f:isWriterOf ?o` селектује све тројке из *RDF* графа које имају вредност предиката `f:isWriterOf`. Укупно постоји 8 могућих типова узорака *RDF* тројки који се добијају пермутовањем константи и променљивих на местима субјеката, предиката и објеката узорака тројки и то су: *spo, sp?, ?po, s?o, ?p?, s??, ??o* и *???*.

Уско грло приликом извршавања упита за велике скупове *RDF* података су операције спајања и уније, због великог броја тројки које могу бити резултат појединачног узорка тројки, а један упит може у себи садржати вишеструка спајања и уније. Зато је од изузетне важности да се умањи потреба за операцијама спајања и уније, као и њихова цена у имплементацији упита, како би се добиле задовољавајуће перформансе и скалабилан приступ жељеним *RDF* подацима. Посебним техникама чувања статистике дистрибуције *RDF* тројки, односно субјеката, објеката и предиката, врши се процена селективности појединих узорака тројки у поступку извршавања упита.

Тренутни стандард за *RDF* упитни језик препоручен од стране WWW конзорцијума од јануара 2008. године је *SPARQL Protocol and RDF Query Languge (SPARQL)* [44] језик, а последња ревизија је *SPARQL 1.1.* из 2013. године. У *SPARQL*-у су дефинисане четири врсте упита: *SELECT, CONSTRUCT, ASK* и *DESCRIBE* форма, с тим да се *SELECT* форма најчешће користи. Основна *SELECT* форма *SPARQL* упита је узорак тројки, која представља тројку са нула или више неповезаних променљивих на местима субјекта, предиката и објекта. Примери *SPARQL* упита са субјектом, предикатом и објектом као неповезаним променљивама су следећи:

```
?tim ff:igraIgrac Pancev .
ff:CrvenaZvezda ?s Pancev .
ff:CrvenaZvezda ff:igraIgrac ?igrac .
```

Резултат извршавања горњих узорака тројки су тројке са следећим особинама: сви тимови за које је играо Панчев, затим сви односи које тим Црвена Звезда има са Панчевом и на крају сви играчи који играју за клуб Црвена Звезда.

Знатно кориснији су *SPARQL* упити који су узорци графа. Они се састоје од скупа узорака тројки, у којима се свака променљива која се појављује у два или више узорака тројки мора везати за исти ресурс. Скуп узорака тројки се пише унутар витичастих заграда {}. Пример оваквог типа *SPARQL* упита је:

```
SELECT ?trener ?klub
WHERE {
   ?trener ff:trenira ?klub .
```

```
   ?klub rdf:type ff:FudbalskiKlub .
   ?klub ff:region Beograd .
}
```

Резултат овог *SPARQL* упита су тренери свих фудбалских клубова из околине Београда.

Основна форма *CONSTRUCT SPARQL* упита је два или више граф узорка који конструишу нови граф. Такође, свака променљива која се налази у више узорака тројки мора да се веже за исти ресурс. *ASK* форма тестира да ли узорак тројки има решење, а *DESCRIBE* форма враћа *RDF* подграф који описује узорак тројки.

2.5.4. Linked Data

Linked Data[5] (Повезани подаци) [45] је релативно нова парадигма која се односи на начин објављивања и дељења података на вебу њиховим представљањем *RDF* моделом података, идентификовањем преко *URI* референци и приступањем преко *HTTP* протокола. Кључни принципи *Linked Data* приступа су следећи:

- Коришћење *URI* идентификатора за именовање ствари и концепата.
- Коришћење *HTTP URI* како би имена могла да се претражују.
- Пружање информација преко стандардних технологија (*RDF*, *SPARQL*) о траженим концептима преко *URI* идентификатора.
- Откривање нових ствари путем веза ка другим *URI* идентификаторима.

Применом ових принципа, дошло је до ширења доступних скупова података на вебу из различитих области живота, при чему количина података унутар одређених скупова неретко достиже неколико стотина милиона тројки, а у неким случајева и пар милијарди. Један од примера је *Linked Open Data* (повезани отворени подаци)[46] иницијатива у којој су подаци из различитих домена као што су географске локације, људи, компаније, књиге, филмови,

[5] http://linkeddata.org/

научни подаци (гени, протеини, лекови), статистички подаци и слично, међусобно повезани тако да представљају један огроман облак (енг. *cloud*) података[6].

Примењеујући принципе *Linked Data* на сензорски домен, дошло је до креирања *Linked Sensor Data* (повезани сензорски подаци) [47, 48] приступа у којем се сензори и сензорска очитавања идентификују са *URI* идентификаторима, а који даље могу да се повезују са другим јавно доступним повезаним подацима, рецимо са местима на којима се сензори налазе. Такав приступ подразумева да се сензорски подаци чувају у *RDF* складиштима и могу да се претражују преко *SPARQL* упита.

Анализирајући природу токова сензорских података, истраживачи су разматрали могућност како токове сензорских података представити коришћењем принципа *Linked Data*. Зато су у референци [49], аутори дефинисали принципе који омогућавају тзв. *Linked Stream Data*:

- Представљање сензора са *URI* идентификаторима.
- Представљање времена са *URI* идентификаторима.
- Представљање просторних локација са *URI* идентификаторима.
- Представљање просторно-временских информација са *URI* идентификаторима.

Описани принципи се примењују у системима за континуалну обраду упита над сензорским токовима.

[6] http://lod-cloud.net/

3. ГЕНЕРИЧКЕ АРХИТЕКТУРЕ ЗА ИНТЕГРАЦИЈУ СЕНЗОРСКИХ МРЕЖА И ЊИХОВЕ ПЕРФОРМАНСЕ

У овом поглављу ће бити разматране основне архитектуре које омогућавају хоризонталну интеграцију сензорских мрежа као и перформансе таквих архитектура у зависности од параметара као што су протоколи апликативног слоја и технике кодовања порука.

3.1. IoT апликације за рад у реалном времену

У овој анализи корисничке апликације су реализоване као веб апликације, иако су у општем случају могуће и стандардне десктоп и мобилне апликације. Разлог за фаворизовање стандардних веб *HTML* апликација је тај што оне имају бројне предности у односу на десктоп-базиране апликације, а то су боља преносивост, лакше распоређивање (енг. *deployment*), аутоматско ажурирање и одржавање за све клијенте. Међутим, стандардни *HTML* није омогућавао подршку апликацијама за рад у реалном времену, што је изразита карактеристика сензорских односно *IoT* апликација. Насупрот томе, програмски оквири (енг. *frameworks*) базирани на уграђеним додацима (енг. *plug-in*) на платформама прегледача веба превазишли су ограничења стандардне *HTML* спецификације већ дуги низ година. Ова подршка је најпре била омогућена преко *Java Applet* додатка[7], технологије која омогућује изршавање *Java* десктоп апликација у оквиру прегледача веба.

Много већу популарност су постигле платформе у виду додатака за прегледаче веба, као што су *Adobe Flash*[8] и *Microsoft Silverlight*[9], које пружају веб програмске библиотеке обогаћене компонентама за исцртавање графике и подршком за размену порука у реалном времену и пренос мултимедијалних садржаја. Међутим, однедавно подршка и интересовање за платформе

[7] https://docs.oracle.com/javase/tutorial/deployment/applet/index.html
[8] https://www.adobe.com/products/flashplayer.html
[9] https://www.microsoft.com/silverlight/

додатака прегледача веба полако нестаје јер креатори апликација не желе да буду зависни од било ког додатка прегледачима, а неке мобилне платформе укључујући *iOS* уопште не подржавају ове додатке.

Током времена, неке *HTTP* засноване технике као што је *Comet* [50] побољшали су подршку за асинхрони пренос података на стандардној *HTML* платформи, али потпуни развој богатих апликација у реалном времену омогућено је увођењем *HTML5* технологија као што су *WebSocket* [51] и *Canvas* графички елемент [52]. Како су имплементације *WebSocket* постале стабилне и зреле, појавио се велики број веб клијената за протоколе за размену порука и у варијанти отвореног кода али и као заштићеног власништва. Тиме су се стекли услови за интензиван развој сензорских односно *IoT* апликација у виду веб апликација.

3.2. IoT протоколи апликативног слоја

На нивоу *IoT* апликативног слоја, могу се идентификовати три модела интеракције (комуникације) између снабедевача података, у овом случају платформе која интегрише сензорске мреже и која представља изворе односно произвођаче сензорских података и корисника, односно потрошача сензорских података, у виду апликација из разних домена. То су интеракција *по моделу повлачења података* (енг. *pull-based)* или *синхрона испорука података, интеракција по моделу гурања података (*енг. *push-based)* или *асинхрона испорука података* и *интеракција по моделу размене порука по објави-претплати (*енг. *publish-subscribe) принципу*. У наставку ће бити дати описи кључних метода и техника за сва три идентификована модела.

3.2.1. Интеракција по моделу повлачења података (енг. pull-based) или синхрона испорука података

Овај модел претпоставља да клијенти издају захтев или упит за одређеним сензорским подацима пружаоцу сервиса, а пружаоц сервиса одговара са одговарајућим подацима. Овај модел је типичан за сервисно-оријентисану архитектуру. Постоје следећи приступи.

3.2.1.1. REST (Representational State Transfer)

REST је архитектурални стил у којем клијент шаље стандардни *HTTP* захтев, бирајући једну од метода као што су *GET, POST, PUT* и *DELETE*, а сервер одговара са одговарајућим подацима. *REST* означава интеракцију између клијента и сервера засновану на ресурсима којима се приступа користећи *HTTP* протокол, адресира преко *URI*, а садржај се представља преко *HTML, XML*, или *JSON* формата.

3.2.1.2. Стандардни веб сервис

Стандардни веб сервис има за циљ да обезбеди комуникациону интероперабилност између различитих софтверских платформи где се интерфејс описује преко *Web Services Description Language (WSDL)* језика [53] за опис сервиса, а поруке се размењују кроз *Simple Object Access Protocol (SOAP*) [54], протоколу који се заснива на *HTTP* транспорту и *XML* формату података. Такође, стандардни веб сервис се може користити из стандардне *HTML-JavaScript* платформе[10], мада је то мање популаран начин због слабијих перформанси, а алтернатива је приступ преко посредничког *HTTP* сервера, са којег ће бити извршен захтев ка стандардном веб сервису, а враћени резултати ће бити представљени у *HTML* или *JSON* формату.

3.2.1.3. CoAP (Constrained Application Protocol)

CoAP [55] је усмерен на *M2M* (*Machine-to-machine*) тј. машина-са-машином комуникацију са циљем омогућавања модела интеракције захтев-одговор као што је *REST* на уређајима и окружењу са ограниченим ресурсима. *CoAP* се лако преводи у знатно захтевнији у погледу ресурса *HTTP* протокол чиме се омогућава интеграција бежичних сензорских мрежа, на пример, са Интернетом преко посредничке компоненте (енг. *proxy*), која се најчешће налази унутар пролазне компоненте (енг. *gateway*). *CoAP* је заснован на *UDP* транспорту и подржава поуздану комуникацију ка једном учеснику (енг. *unicast*), као и комуникацију ка више учесника (енг. *multicast*) са тзв. најбољим

[10] https://plugins.jquery.com/soap/

покушајем (енг. *best-effort*), тако да су крајње тачке комуникације дефинисане као *IP* адреса и *UDP* порт. *CoAP* уноси веома мали додатни садржај у поруке: садржи заглавље од 4 бајта, затим опционо заглавље од једног или два бајта и затим садржај. *CoAP* поруке се морају сместити у једном IP датаграму, што у случају IEEE 802.15.4 базираних протокола (нпр. *6LoWPAN*) производи 127 бајтова дуге поруке. Као проширење стандардног REST стила, *CoAP* омогућава клијентима да издају захтев за праћење специфичног ресурса на серверу додавањем параметра "*Observe*" уз GET захтев, што резултира примањем асинхроних обавештења о ресурсу са сервера.

3.2.1.4. Метода позивања удаљеног објекта

Овај метод се заснива на стандардном *HTTP* захтеву и представља заправо метод прилагођен програмерској перспективи и доступан је у различитим програмским апликативним интерфејсима (енг. *Application Programming Interface - API*) библиотека у веб окружењу. Идеја је да се на страни клијента обезбеди посреднички објекат (*proxy*) са одређеним методама који представља удаљени објекат на серверу. Позивање методе посредничког објекта се преноси на удаљени објекат путем *HTTP* захтева и одговарајућим кодовањем аргумената. Најчешћи приступ је такав да клијент нуди објекат/функцију за повратни позив која се позива када је операција комплетирана. Овај приступ је погодан ако клијент и сервер имају компатибилне програмске платформе.

3.2.2. Интеракција по моделу гурања података (енг. *push-based*) или асинхрона испорука података

Ово је модел интеракције клијента и сервера који дозвољава серверу да изврши слање података клијентима одмах по њиховој доступности, на пример по доласку на сервер. Постоји неколико техника:

3.2.2.1. Long-Polling

Клијент шаље *HTTP* захтев и чека податке од веб сервера. Сервер задржава конекцију све док нови подаци не буду на располагању, дође до истека времена или клијент прекине везу. Након доласка података на сервер, сервер

шаље податке клијенту и након тога клијент покреће нови *HTTP* захтев. Стога, сервер може да шаље податке клијентима у било којем тренутку јер увек постоје захтеви на чекању код сервера за дате клијенте. Добра страна коришћења *long-polling* технике је употреба стандардног *HTTP* протокола који није блокиран од стране заштитног зида (енг. *firewall*). Такође, то је робустан метод и ради и са активираним прокси сервером. Недостатак је заузимање конекције по клијенту, чак и ако се подаци не преносе.

3.2.2.2. HTTP Streaming

Код ове технике, веб сервер не завршава поруку са одговором односно везу као и обично, већ је држи отвореном и само додаје нове податке у поруку одговора. Ова техника се може реализовати кроз *XHR multipart streaming* и *XHR iFrame streaming* [56]. *XHR multipart streaming* користи *HTTP* тип садржаја '*multipart*', који омогућава серверу слање података у више делова. *XHR iFrame streaming* омогућава да се подаци шаљу у више <*script*> тагова. Да би се спречиле огромне величине одговора на клијентској страни, веза се мора периодично прекинути и затим послати нови *HTTP* захтев.

3.2.2.3. Сокет везе

Сокет је технологија базирана на *TCP* транспорту за остваривање двосмерне мрежне комуникације преко једне конекције. *HTML5* спецификација је представила *WebSocket* протокол [51], који омогућава комуникацију преко сокета из прегледача веба. Успостављање *WebSocket* конекције се покреће надоградњом *HTTP* захтева. Након успостављања везе, нема потребе за размену заглавља између повезаних страна, тако да се додају минимални контролни подаци.

3.2.3. Интеракција по моделу размене порука по објави-претплати механизму

Овај модел интеракције клијента и сервера у суштини користи комуникационе примитиве из претходно описаних приступа. Овај модел омогућава клијенту (претплатнику) да се претплати на податке који су

повезани са одређеном темом и да прима податке о тој теми које објављују други клијенти (објављивачи). Брокер (или диспечер) је серверска компонента одговорна за усмеравање порука између објављивача и претплатника, обезбеђујући уједно квалитет услуга, перзистенцију порука и сличне функционалности. Неколико протокола за размену порука се користе као *IoT* апликативни протоколи и у наставку ће бити дат њихов опис, а у табели 1 је приказано њихово поређење.

3.2.3.1. MQTT (Message Queue Telemetry Transport)

MQTT [57] је дизајнирао *IBM* 1999. године за лагану (енг. *lightweight*) *M2M* комуникацију са циљем омогућавања објави-претплати протокола за размену порука са што минималнијим захтевом за пропусним опсегом, величином кода, потрошњом енергије и минималним додатним подацима у порукама. Теме у *MQTT* протоколу имају хијерархијска имена која су раздвојена косом цртом (/), на пример *wsn1/sensors/temperature/temp1*. Клијентима је дозвољено да користе "џокер" знакове приликом претплаћивања на жељене теме како би што лакше покрили више тема. Постоје три нивоа квалитета услуге у *MQTT*: испорука података без захтевања потврде пријема, са захтевањем потврде пријема и испорука „тачно једном" која подразумева четири корака у току синхронизације испоруке података. Постоји варијација овог протокола под називом *MQTT for Sensor Networks (MQTT-SN)* који је намењен за употребу на уређајима који раде на не-*TCP/IP* мрежама као што је *ZigBee*.

3.2.3.2. AMQP (Advanced Message Queuing Protocol)

AMQP [58] је бинарни протокол отвореног стандарда за размену порука високих перфомансi, првенствено дизајниран за пословно окружење, али се користи и у другим апликативним областима. *AMQP* 1.0 [59] је тренутна верзија и то је протокол на нивоу жичаног преноса који дефинише формат порука са заједничким типовима података, при чему се могу обезбедити додатни мета-подаци ради ефикасне интерпретације података, чиме се постиже интероперабилност између различитих произвођача. Протокол обезбеђује поуздану комуникацију са три начина испоруке порука: највише

једном, најмање једном и тачно једном испоруком. За разлику од *AMQP* верзије 1.0, *AMQP* 0.9.1 верзија претпоставља модел у којем се поруке објављују ка *размењивачима (*енг. *exchanges)*, и према обавезујућим правилима *везивања (*енг. *binding)* поруке се прослеђују у *редове (*енг. *quees)* и даље испоручују клијентима који су претплаћени на те *редове*. У зависности од типа *размењивача*, постоје четири могућа начина усмеравања (рутирања) поруке између објављивача и потрошача: директна размена (поруке се усмеравају само једном *реду*), размена на више излаза (поруке се усмеравају ка сваком *реду* који је везан за *размењивач*), размена по темама (поруке се усмеравају према кључевима и џокер знаковима), и размена по заглављима (поруке се усмеравају према њиховим атрибутима у заглављима)

3.2.3.3. XMPP (Extensible Messaging and Presence Protocol)

XMPP [60] је скуп технологија за размену порука у реалном времену и заснива се у основи на *XML* стриминг технологији. Протокол је 1999. године развила *Jabber* заједница отвореног кода за апликације за четовање (енг. *instant messaging*). Спецификација протокола садржи језгро спецификације стандардизоване од стране *Internet Engineering Task Force (IETF)* [61] и преко 300 екстензија објављених кроз проширења протокола односно *XMPP Extension Protocols (XEPs)* који покривају различите намене као што је модел размене порука објави/претплати (*XEP-0060: Publish-Subscribe*), размена сензорских података (*XEP-0323: Internet of Things - Sensor Data*), вишекориснички чат итд. У *XMPP* протоколу, клијенти размењују *XML* поруке назване *станца* (енг. *stanzas*). Постоје три основне врсте *станца*: *message*, *presence* и *iq (info/query)*. *Message* станца је корени елемент и обухвата садржај корисничке поруке (у елементу потомку *body*), као и информације о пошиљаоцу (*from*), примаоцу (*to*), типу поруке и идентификатор (*id*). У објави-претплати моделу размене порука чвор (енг. *node*) представља тему. Протокол је дошао у фокус IoT заједнице [62, 63], после појављивања лаких имплементација за уређаје са ограниченим ресурсима, тј. *µXMPP* [64] и *XMPP client for mbed* [65].

3.2.3.4. DDS (Data Distribution Service)

DDS [66] је отворени стандард за комуникациони протокол средњег слоја. *DDS* предлаже архитектуру без сервера за интероперабилно дељење података са високим перфомансама користећи објави-претплати модел централизован око података (енг. *Data-Centric Publish-Subscribe*). Овај модел претпоставља типизоване интерфејсе дозвољавајући *DDS* учесницима да дефинишу теме са одређеним типовима података, који одговарају типовима података објеката које апликације желе да објављују или примају. Апликације користе компоненту *DataWriter* датог типа података за објављивање објеката података на одређену тему преко компоненте *Publisher*, док се компонента *DataReader* датог типа података користи за пријем објеката података преко компоненте *Subscriber*. Динамичко откривање *DDS* учесника се реализује упаривањем њихових објава и претплата на основу тема истог имена и типа података. За дефинисање типова података се користи *Interface Definition Language (IDL)*. Квалитет сервиса се може одредити на нивоу *Publishers/Subscribers* компонената, као и на нивоу *DataWriters/DataReaders*. *DDS* протокол специфицира употребу мултикаст *UDP* унутар *LAN*, и *TCP* транспорт за комуникацију преко *WAN* окружења. Иначе, *DDS* спецификација дефинише програмске интерфејсе у *C++* и *Java* програмским језицима.

Табела 1- Поређење протокола IoT апликативног слоја за пренос порука

Протокол	Иницијална намена	Стандард	Транспорт	Објави/Претплати Модел	Поузданост испоруке порука		
					Са потврдом	Без потврде	Тачно-једном
COAP	REST on Constrained Devices	IETF RFC 7252	UDP	Observing feature	+	+	-
MQTT	Lightweight M2M	OASIS Standard	TCP	Hierarchical topics	+	+	+
AMQP	Enterprise apps	ISO and IEC	TCP	4 Exchange types: Direct, Fan Out, Topic, Header	?	+	+
XMPP	Instant Messaging	IETF RFC 6120, 6121	TCP	Node in pub/sub plug-in	-	-	-
DDS	High Performance Apps	Object Management Group (OMG)	UDP, TCP	Typed topics	---------- 23 QoS полиса ---------		

3.3. Кодовање порука

У клијент-сервер комуникацији, пренети подаци се кодују у одређеним форматима порука који производе различите величине порука и захтевају одређена времена за операцију кодовања/декодовања, док комплексне шеме омогућавају серијализацију целог графа објеката или пружају мета-податке и на тај начин омогућавају интероперабилност. Генерално, формате порука можемо класификовати у *текстуалне* и *бинарне формате порука*. У оба приступа, по пријему поруке, клијент парсира податке и претвара их у одговарајућу интерну структуру програмске платформе. Након тога, подаци су спремни за даљу обраду. Следи опис најчешће коришћених формата порука.

3.3.1. Текстуални формати порука

Текстуални формати порука чувају податке у људима читљивом формату. Најпопуларнији представници су *Extensible Markup Language (XML)* [67], *JavaScript Object Notation (JSON)*[11] и *Comma-Separated Values* (CSV)[12]. *XML* је универзални формат за размену података. Код њега су одвојени подаци и мета-подаци који се могу односити на граматичке дефиниције (шема) и који се користе за проверу валидности *XML* документа. *JSON* садржи податке изражене у облику парова име-вредност у типичној програмерској нотацији, али независно од било ког програмског језика, иако је иницијално настао за *JavaScript*. *CSV* је једноставан формат код кога су подаци одвојени зарезима и примену је нашао у апликацијама за унакрсна израчунавања, односно код табеларног начина приказивања података. У случају примене семантичких технологија за сензорске односно IoT апликације, за представљање семантичких сензорских података користе се серијализовани формати RDF модела [37-41]. Опис популарних формата за серијализацију RDF модела је дат у одељку 2.5.1.

[11] http://json.org

[12] https://tools.ietf.org/html/rfc4180

3.3.2. Бинарни формати порука

Бинарни формати порука су дизајнирани са специфичним циљем смањивања величине поруке и/или ефикасног кодирања графа објеката, али не обезбеђују добру интероперабилност. Многи програмски језици обично имају имплементирану подразумевану серијализацију објеката бинарним кодовањем, нпр. *Java* серијализација објеката[13], али постоје и напредније технике као што је *Kryo* програмска библиотека[14], која се користи у многим апликацијама високих перформанси. Неки текстуални формати порука, као што су *XML* и *JSON*, имају побољшану бинарну верзију. На пример, *Efficient XML Interchange (EXI)* [68] формат је дизајниран тако да претвори *XML* поруке у бинарне и на тај начин смањи величину поруке и пропусни опсег у окружењу са ограниченим ресурсима што је посебно важно за IoT домен. У 2011. години, *W3C* је усвојио *EXI* формат као препоруку [69]. Бинарни *JSON* формати као што су *BSON*[15], *BJSON*[16], и *UBJSON*[17], имају за циљ примену у апликацијама високих перформанси и омогућавају једноставно парсирање и рад са бинарним порукама ослањајући се на једноставност *JSON* организације података без шеме. Избегавајући обраду текста, ови формати смањују величину поруке и побољшавају обраду порука у поређењу са стандардним *JSON*-ом.

Бинарни формати порука се такође користе у комуникацији специфичној за неку апликацију када се подаци ређају према интерној спецификацији која тежи да омогући лакшу конверзију и боље перформансе. У таквим ситуацијама, произвођачи пружају сопствену имплементацију бинарног кодовања порука. *Protocol Buffers (PBF)* [70] је компактни бинарни формат развијен у компанији *Google*, заснован између осталих техника, на кодовању целих бројева променљивом величином записа. Бинарни формати порука

[13] https://docs.oracle.com/javase/7/docs/platform/serialization/spec/serialTOC.html
[14] https://github.com/EsotericSoftware/kryo
[15] http://bsonspec.org/
[16] http://bjson.org/
[17] http://ubjson.org/

отвореног кода *Colfer*[18] и *Protostuff*[19] заснивају се на идејама формата *PBF* и додатно побољшавају перформансе у погледу брзине обраде и величине порука [71]. Истраживачки напори везани за семантичке токове података испитивају укључивање техника компресије на *RDF* токовима података како би постигли и уштеду простора и боље време обраде. На пример, *RDSZ* [72] и рад *Joshi*-ја и других [73] су примери таквих приступа. Техника *HDT* [74] креира бинарну поруку од *RDF* података користећи особине *RDF* графовске структуре кроз одвојено кодовање концепата и тројки из *RDF* графа.

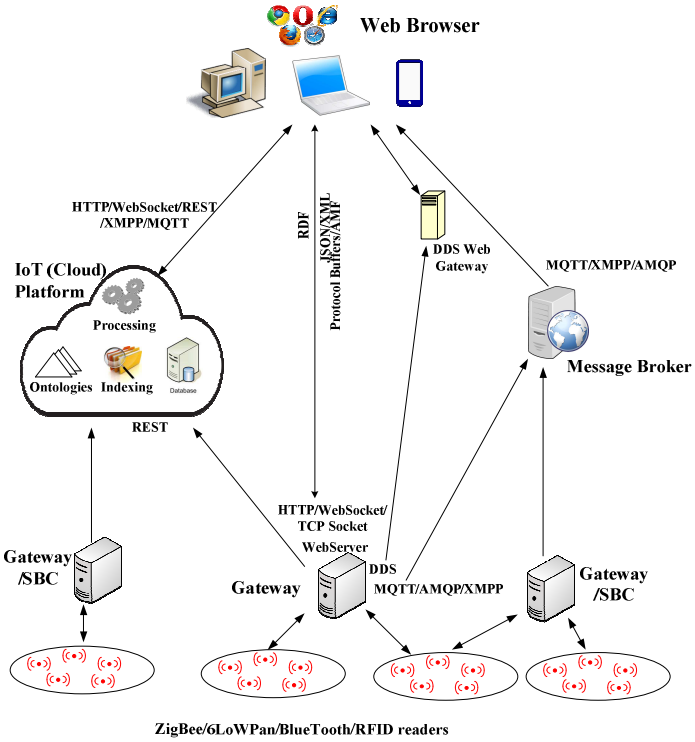
3.4. Генеричка архитектура апликација са хоризонталном интеграцијом сензорских мрежа

У циљу анализе ефикасности појединих архитектура за интеграцију сензорских мрежа, креирана је генеричка архитектура (слика 3) која покрива широк спектар сценарија примене из различитих домена укључујући здравствену заштиту, паметне куће, надзор животне средине, паметне градове, транспорт и логистику итд.

Генерално, генеричке архитектуре можемо поделити на основу логичких или физичких компонената или функционалних улога. Код физичке поделе, могу се издвојити три типа организације интеграције сензорских мрежа и апликација које се базирају око:

- Пролазног уређаја (енг. *gateway*)
- Брокера порука
- IoT платформи

[18] https://github.com/pascaldekloe/colfer
[19] https://github.com/protostuff/protostuff



Слика 3 – Генеричка архитектура апликација
са хоризонталном интеграцијом сензорских мрежа

Логичка подела се врши на основу функционалности које се извршавају у процесу прикупљања, обраде и испоруке сензорских података. У том контексту, генеричка архитектура је подељена у три слоја и то: *слој података или перцепције*, *слој обраде* и *апликативни слој*.

Слој података садржи хетерогене уређаје који су извори сензорских опажања појава и физичких величина из стварног света. Примарни уређаји су сензорски чворови опремљени одговарајућим сензорима и/или актуаторима. Распон тих уређаја иде од уређаја са ограниченим ресурсима укључујући извор енергије, рачунских и комуникационих капацитета, до моћнијих компактних рачунарских модула (енг. *Single Board Computer – SBC*) и паметних телефона или чак крајње једноставних *RFID* тагова, активних или пасивних. Сензорски чворови у оквиру *WSN* су повезани бежичном везом формирајући топологију у облику *звезде* или *peer-to-peer* организацију, користећи IEEE 812.15.4 засноване протоколе као што су *ZigBee* и *6LoWPAN*, или *Bluetooth*, градећи тако бежичну сензорску и актуаторску мрежу (енг. *WSAN*) (одељак 2.1). Ови чворови могу да извршавају лаган (енг. *lightweight*) софтвер за

опслуживање захтева за подацима као што су *CoAP* [55] и *XMPP* [62-65] у циљу омогућавања комуникације од краја-до-краја и интеракције базиране на повлачењу података од корисника који пребивају у *WAN* окружењу. Међутим, таква комуникација се мора усмеравати преко базне станице или пролазне компоненте (енг. *gateway*). У случају сензора повезаних на *SBC* уређаје и мобилне телефоне [75], сензорска очитавања се могу даље прослеђивати ка *WAN* мрежи преко *WiFi*, *GPRS*, или *LTE* конекције или евентуално преко жичне везе и неког од одговарајућих протокола. У *IoT пројектантском прегледу* (енг. *IoT Developer Survey*) [76] израђеном током 2016. године од стране *Eclipse IoT Working Group* у партнерству са *IEEE IoT* и *AGILE-IoT* истраживачким пројектом, којом приликом је анкетирано 528 пројектаната IoT апликација, показује се да пројектанти у 70,9% случајева своја решења базирају на *TCP/IP* протоколу и у 73,1% случајева на *Linux* оперативном систему, док се протоколи за сензорске чворове са ограниченим ресурсима *ZigBee* и *6LoWPAN* користе у 25,4%, односно у 16,2% случајева. Ови подаци наводе на закључак да су однедавно SBC уређаји знатно заступљенији у IoT архитектурама од сензорских чворова са веома ограниченим ресурсима.

3.4.1. Архитектура са пролазним уређајем

Пролазни уређај (енг. *gateway*) је типично хардверски знатно моћнији уређај у погледу расположиве енергије, процесне моћи и комуникационих могућности. То је обично стандардно опремљени десктоп рачунар, али исто тако то некада може бити *SBC* уређај попут *Raspberry Pie*[20] који извршава уграђени *Linux* оперативни систем. У зависности од изабране организације, пролазни уређај може да обавља различите функције као што је транслација протокола између *IPv4/v6* мрежа и *6LoWPAN* сензорских мрежа, управљање уређајима, сакупљање података од сензорских чворова и накнадног објављивања прикупљених података ка IoT платформи или брокеру порука, а може извршавати компоненту средњег слоја (енг. *middleware*) у циљу сакривања специфичности сензорских мрежа иза њега. Пролазни уређај такође може да опслужује захтеве за сензорским подацима од стране

[20] https://www.raspberrypi.org/

заинтересованих корисника излагањем функционалности кроз изабране сервисне интерфејсе, у складу са описом из поглавља 3.2. Интерактивни модел повлачења података понуђен преко *REST* приступа, стандардног *HTTP* протокола, или веб сервисног интерфејса, се обично користи за захтевање најсвежијих сензорских података или за слање упита клијената.

Са друге стране, пролазни уређај може опслуживати модел интеракције гурања података преко *TCP* или *WebSocket* конекција или неком од *HTTP*-базираних техника (одељак 3.2.2). То се може искористити у случају испоруке података вођеном догађајима или за испоруку токова сензорских података клијентима. Ове услуге могу бити хостоване употребом стандардног апликативног сервера или извршавањем самосталних комуникационих програмских библиотека и платформи као што су на пример *Netty*[21] или *Node.js*[22]. Овај приступ је нарочито погодан када се користи сопствено прилагођено решење за интеграцију сензорских мрежа мањег степена скалабилности, када се може користити и специфично кодовање порука.

3.4.2. Архитектура са брокером порука

Ако није неопходна нека специфична обрада сензорских података, ефикасна скалабилна архитектура се може реализовати употребом једног или више брокера порука који дистрибуирају објављене сензорске податке заинтересованим корисницима. Брокери имплементирају неки од протокола за размену порука као што су *MQTT*, *XMPP*, или *AMQP*, док архитектура заснована на *DDS* протоколу не захтева посредника у преносу порука, иако постоје посебне компоненте за откривање и преусмеравање да би се пронашли и директно повезали одговарајући објављивачи и претплатници порука. Ако клијенти користе веб апликације и *DDS* протокол, неопходна је посредничка компонента која би извршила конверзију на *WebSocket* протокол. У ситуацији када се не очекује велики број корисника, чак и пролазни уређај може покретати брокер порука и испоручивати сензорске податке

[21] http://netty.io/
[22] https://nodejs.org

заинтересованим клијентима који пребивају у *LAN/WAN* окружењу. Као што је описано у одељку 3.2.3, неки протоколи за пренос порука омогућавају пројектантима система да дефинишу различите нивое квалитета сервиса, што одређује да ли је гарантована испорука порука. Штавише, брокер порука може сачувати поруке како би се оне касније испоручиле претплатницима који нису били повезани када су те поруке биле објављене.

3.4.3. Архитектура са IoT платформом

Централна компонента у процесном слоју је IoT платформа која укључује широку класу решења која могу извршавати више различитих функционалности укључујући функције средњег слоја, обраду и складиштење сензорских података и друге. IoT платформа може бити интегрисана са облаком (енг. *cloud*) [22, 77, 78, 79] и изложити своје карактеристике по платформа-као-сервис моделу (енг. *Platform as a Service - PaaS*). Платформе у облаку нуде скоро неограничене складишне и рачунарске ресурсе и на тај начин могу подржати решења која обрађују огромну количину IoT података. Типично, IoT платформа може обављати низ функционалности попут складиштења сензорских података како би касније били спојени са новим подацима, вршења фузије података који су пореклом из различитих сензорских извора, примене алгоритама машинског учења и проналажења скривеног знања (енг. *data mining*) како би се класификовали прикупљени подаци, да би се предвидела нова сензорска опажања или за откривање аномалија код ново-пристиглих података. У оквиру IoT платформе, могу се комбиновати разна решења за обраду огромне количине података и алати за рад са порукама у реалном времену попут *Apache Spark*[23], *Apache Kafka*[24], *Apache Storm*[25] и *Spark Streaming*[26]. У референци [77] аутори су извршили преглед таквих решења и њихових потенцијалних употреба унутар IoT платформи.

[23] http://spark.apache.org
[24] http://kafka.apache.org/
[25] http://storm.apache.org/
[26] http://spark.apache.org/streaming/

Решења базирана на семантичким технологијама трансформишу сирове сензорске податке у формате вишег апстракног нивоа, поштујући одређена синтаксна правила или додавајући сировим подацима мета-податке тако креирајући семантичку представу сензорских опажања по *RDF* моделу. Овакве семантички базиране архитектуре нуде интелигентне сервисе излагањем *SPARQL* крајњих тачака. Хоризонтална интеграција сензорских мрежа се реализује представљањем хетерогених сензорских података преко концепата дефинисаних у онтологијама сензорског домена. Централна онтологија покрива сензорских домен за коју *W3C* предлаже стандардизовану Онтологију семантичких сензорских мрежа (енг. *Semantic Sensor Network Ontology*) [12]. У поглављу 4 детаљно ће бити анализиране семантички-базиране архитектуре за интеграцију сензорских мрежа.

IoT платформе могу такође интегрисати систем за непрекидну обраду упита (енг. *continuous query processing engine*), који обавља обраду у реалном времену над примљеним сензорским подацима, и тако произвести сложене, агрегиране или изведене податке које је потребно проследити корисницима уколико су испуњени одређени услови. IoT платформа излаже своје сервисе кроз различите комуникационе протоколе и стандарде. У студијама више аутора [22, 77, 79] дата је упоредна анализа расположивих IoT платформи. Те платформе укључују у себе неке од популарних брокера порука за IoT апликативне протоколе за пренос порука. На пример, *Xively*[27] има подршку за *REST* и *MQTT* протокол, док *Nimbits*[28] подржава *REST* и *XMPP* протокол. Анализа перформанси приликом испоруке сензорских података код IoT платформи је доста сложена због употреба различитих компонената у целокупном процесу проласка података од извора до потрошача сензорских података. На овом месту биће дат приказ скупа стандарда за сервисе интегрисаних сензорских мрежа креираних од стране *Open Geospatial Consortium Sensor Web Enablement (OGC SWE)* радне групе, као једне од

[27] https://xively.com/

[28] https://www.nimbits.com/

најранијих и најутицајнијих спецификација, веома често имплементираних барем у једном делу од стране разних IoT платформи.

3.4.3.1. OGC SWE скуп стандарда за сервисе интегрисаних сензорских мрежа

Најпознатији скуп стандарда у области интеграције сензорских мрежа је *OGC SWE* спецификација [33][34], започета још 2003. године са циљем да се омогуће принципи идеје *сензорског веба (Sensor Web)*, концепта повезаних сензорских мрежа и сензора контролисаних од различитих власника и са другачијим карактеристикама. Стога је предложено неколико спецификација кодовања података, модела и сервисних интерфејса унутар *SWE*. Спецификације су континуално побољшаване и током 2011. године се појавила *SWE* верзија 2.0 [34], а и даље се побољшава. Иначе, *OGC* је међународни конзорцијум индустријских, академских и државних организација које учествују у развоју отворених стандарда за геопросторне и локацијске сервисе.

OGC SWE спецификација полази од идеје *Sensor Web* са глобално интегрисаним сензорима и фокусира се на следеће функционалности: откривање сензорских система, сензорских опажања (енг. *sensor observations*) и процеса опажања који задовољавају потребе корисника, достављање информација о сензорским техничким могућностима и квалитету мерења, пружање корисницима могућност издавања захтева за мерење сензора, претплате на упозорења за одређене сензорске вредности који задовољавају дефинисане критеријуме, дохватање сензорских опажања у реалном времену и могућност процесирања сензорских опажања коришћењем сензорских параметара. Унутар *SWE* спецификације, поред физичких сензора, сензорски ресурси могу да укључе архивиране сензорске податке, резултат симулације или процесирања алгоритама. Да би се остварile описане функционалности, *SWE* садржи неколико спецификација:

Sensor Model Language (SensorML) спецификација садржи стандардне моделе и *XML* шему за опис сензорских система и процеса као што су процедура

сензорског мерења и процесирање након мерења. *SensorML* описи се користе за откривање сензорских система, успостављање других сервиса као што су *SOS*, *SAS* и *SPS*, захтевање процесирања сензорских опажања и описа целокупног трајања сензорског опажања од настајања до процесирања.

Observations & Measurements (O&M) спецификација се састоји од стандардног модела и *XML* шеме за кодовање измерених сензорских вредности. Главни концепт у *O&M* спецификацији је *Observation* (опажање) и представља чин опажања особине својства од интереса (енг. *Feature of Interest*), на пример брзине ветра, који резултује производњом неке вредности. Својство представља неки идентификован објекат који се може опажати. Вредност је резултат неке процедуре која је описана са *SensorML* записом. Структура *Observation* кодована у формату *XML* садржи поља која се односе на следеће ресурсе: својство од интереса, коришћена процедура која је произвела резултантну вредност, опажана особина, резултат и квалитет резултујуће вредности, временске и друге параметре који описују догађај опажања. Просторне особине опажања су дате преко локацијске особине својства од интереса. Да би се омогућила већа интероперабилност, препоручује се коришћење *SWE Common* типова података. У *O&M* верзији 2.0 концептуални модел је постао ISO стандард.

Sensor Observation Service (SOS) укључује спецификацију стандардних веб сервисних интерфејса за захтевање, филтрирање и дохватање сензорских опажања и информација о сензорским системима. Ова спецификација користи *SensorML* описе за моделовање сензора и сензорских система, а *O&M* за моделовање одговора сензорских опажања. Типичне операције унутар сервиса су *GetObservation* за дохватање сензорских опажања, *DescribeSensor* за добијање описа сензора и *GetCapabilities* за дохватање сервисних мета података. Побољшања у *SOS* 2.0 верзији се односе на обавезно и стриктно коришћење *O&M* 2.0 спецификације за кодовање одговора и увођење обавезних временских и просторних оператора.

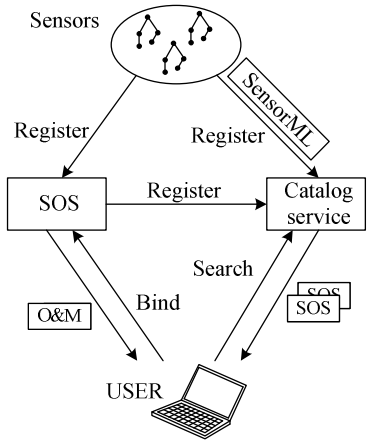
Sensor Planning Service (SPS) дефинише стандардни веб сервисни интерфејс за захтевање кориснички задатих операција сензорских очитавања и задатака за сензоре као и слање колекције захтева ка сензорима и конфигурабилним процесима. Интерфејс омогућује комплетан процес контроле, планирања и праћење статуса сензорских задатака. Типичне операције су *Submit* за слање задатка, *GetStatus* за добијање статуса послатог задатка и *GetFeasibility* за проверу да ли је задатак изводљив за неки сензор. У верзији *SPS* 2.0 извршена је промена модела и синхронизација са осталим спецификацијама унутар *SWE 2.0*, а модел обавештавања се ослања на објави/претплати механизам.

Sensor Alerting Service (SAS) укључује стандардни веб сервисни интерфејс за објављивање и претплаћивање на сензорска упозорења. *SAS* сервис извршава алгоритам упоређивања примљених сензорских мерења са узорцима и ако су задовољени услови за упозорење, дистрибуира обавештење о упозорењу ка претплаћеним корисницима. Корисници су у могућности да дефинишу сопствене критеријуме за детекцију одређених догађаја. Примери упозорења су детекција вредности испод или изнад граничне вредности, детекција кретања, низак ниво батерије у неком сензорском чвору и слично. У SWE 2.0 спецификацији, *SAS* је замењен са новим сервисом *Sensor Event Service* са циљем да се ослања на стандардне протоколе и формате асинхроног обавештавања корисника. Такође, повећана је интероперабилност са другим *SWE* сервисима јер се користи *O&M* стандард за кодовање одговора.

Web Notification Service (WNS) садржи спецификацију за стандардни веб сервисни интерфејс за асинхрону испоруку порука или упозорења од других *SWE* сервиса. *WNS* обезбеђује два начина асинхроног обавештавања: са и без чекања одговора корисника коме је послато обавештење. Међутим, у новијој спецификацији, овај сервис се више не користи.

Постоји неколико сценарија коришћења *SWE* сервиса од стране корисника у сврху добијања жељених сензорских опажања. Каталог сервис (енг. *catalog service*) пружа корисницима функције откривања (енг. *discovery function*)

сензора који су се регистровали преко *SensorML* описа, или је *SOS* регистровао те сензоре у каталог сервису. Интеракција почиње када корисник затражи од каталог сервиса одређена сензорска опажања (слика 4). Каталог сервис враћа кориснику инстанце *SOS* сервиса који су у могућности да одговоре на такав захтев. Корисник се конектује на неки од примљених *SOS* сервиса и добија сензорска опажања кодована у *O&M* формату. У случају када каталог сервис не пронађе адекватни *SOS* сервис, корисник може да затражи одговарајући *SPS* сервис ради захтева одговарајућем сензору да изврши мерење и произведе сензорско опажање потребно кориснику. *SPS* у том случају може обавестити корисника о расположивој вредности или одмах или са неким кашњењем асинхроно преко *WNS*.



Слика 4 – Интеракција корисника и сервиса код OGC SWE
(адаптирана слика из референце [33])

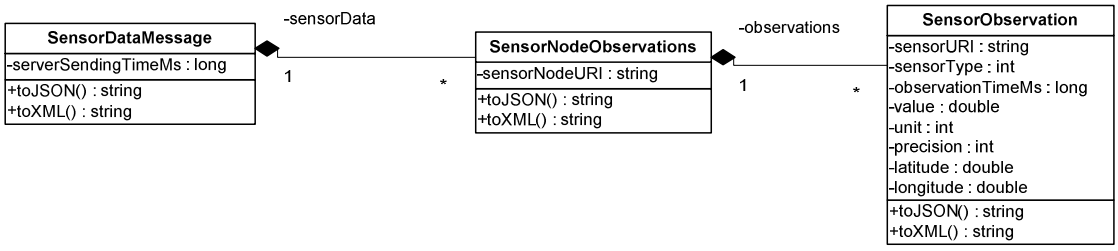
Последњи сценарио се односи на случај када је кориснику потребно сензорско опажање које задовољава неки услов. У том случају, корисник добија информацију од каталог сервиса о одговарајућем *SAS* сервису и корисник се претплаћује на тај *SAS*. *SAS* непрестано добија сензорска опажања, врши њихову проверу са задатим условом и обавештава корисника када је задовољен услов. Обавештења се шаљу кориснику или директно или преко *WNS* сервиса.

Главни недостатак *OGC SWE* спецификације је ограничена интероперабилност јер садржи само синтаксна правила кодовања података, али не и семантичке податке о сензорима и њиховим опажањима. Без обзира

што спецификација не адресира семантички аспект сензорских података, концептуални модел *OGC SWE* спецификације је знатно утицао на низ будућих информационих модела са укљученом семантиком података као и уопштено на област интеграције сензорских мрежа.

3.5. Евалуација перформанси генеричких IoT архитектура

Из претходног поглавља је јасно да постоји низ параметара који могу да утичу на перформансе одређене архитектуре за интеграцију сензорских мрежа. Због тога је извршена евалуација перформанси одређених генеричких архитектура анализом утицаја избора комуникационог протокола и кодовања порука. Стога су изабране две тестне апликације које за извршавање користе две генеричке архитектуре. Прва тест апликација се извршава на архитектури са пролазним уређајем који покреће апликативни сервер, а врши се мерење кашњења пропагације сензорских података од извора сензорских опажања до корисничких апликација, док у другој тест апликацији која се извршава на архитектури заснованој на брокеру порука, врши се мерење кашњења преноса као и брзина протока порука са сензорским подацима.



Слика 5 - Сензорски модел података

Апликативни сценарио коришћења је преузет из система даљинског грејања [80], у којем су постављени сензори на разним локацијама дистрибутивне мреже и они пружају информације о температури, притиску и протоку воде у цевима, потрошњи енергије итд. Слични обрасци коришћења могу се наћи у IoT апликацијама из других домена. Сва сензорска опажања се прикупљају на пролазном уређају одакле се даље испоручују у складу са изабраним архитектуралним приступом.

Сензорски модел података је у одређеној мери настао под утицајем модела предложеног спецификацијом *OGC SWE O&M* [33] (слика 5). Сензорско опажање садржи следеће податке: *URI* сензора, тип сензора, измерену вредност, јединицу за вредност, време опажања и опционо географску ширину и дужину локације сензора. Порука са сензорским подацима садржи сензорска опажања са једног или више сензорских чворова.

```
<sensorData>
  <sensorNodeObservations>
      <sensorNodeURI>sensor.net/sn0</sensorNodeURI>
      <sensorObservation>
            <sensorURI>sensor.net/sn0/flowSensor2</sensorURI>
            <sensorType>5</sensorType>
            <observationTimeMs>1470852496994</observationTimeMs>
            <val>53.34045686695581</val>
            <unit>5</unit>
      </sensorObservation>
       ...
      <sensorObservation>
       ...
      </sensorObservation>
  </sensorNodeObservations>
  <serverSendingTimeMs>1470852496995</serverSendingTimeMs>
</sensorData>
```

а) XML Sensor Data Message

```
{"sensorData":
  [{"sensorNodeURI":"sensor.net/sn0",
    "observations":
            [{"sensorURI":"sensor.net/sn0/flowSensor2",
              "sensorType":5,
              "observationTimeMs":1470852496994,
              "val":53.34045686695581,
              "unit":5},
              ...
              {"sensorURI":...
              }]
  }],
  "serverSendingTimeMs":1470852496995
}
```

б) JSON Sensor Data Message

```
message SensorDataMessageProtoBuf {
        required uint64 server_sending_time = 1;
        repeated SensorNodeObservationsProtoBuf sensor_data = 2;
}
message SensorNodeObservationsProtoBuf {
        required string sensor_node_uri = 1;
        required uint64 server_sending_time = 2;
        repeated SensorObservationProtoBuf observations = 3;
}
```

в) Google Protocol Buffers Data Definition

```
message SensorObservationProtoBuf {
        required string sensor_uri = 1;
        required int32 sensor_type = 2;
        required uint64 observation_time_ms = 3;

        double value = 4;
        optional int32 unit = 5;
        optional int32 precision = 6;

        optional double latitude = 7;
        optional double longitude = 8;
}
```

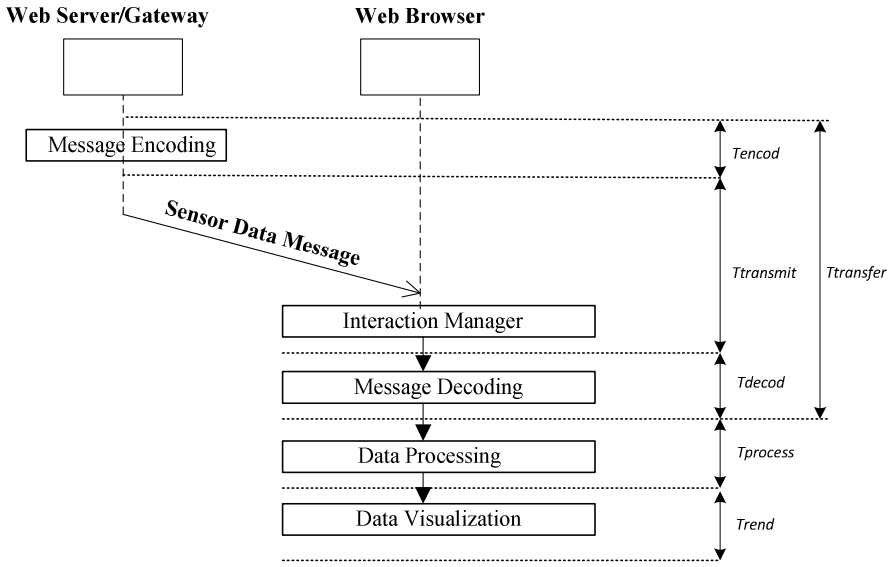
Слика 6 – Сензорске поруке: а) XML формат, б) JSON формат и в) Google Protocol Buffers дефиниције порука

У складу са усвојеним моделом сензорских података, креиране су *XML* и *JSON* поруке (слика 6а и 6б), и конфигурисане су структуре података за генерисање *Protocol Buffers* формата бинарних порука (слика 6в). Сензорска опажања су генерисана брзином од 1Hz. Тестови су обухватали испоруку 100 порука, што значи да једна итерација теста траје 3,33 минута. Постоје три параметра који се могу изабрати у апликацији: комуникациони протокол, формат кодовања порука и број надгледаних сензорских чворова који сразмерно повећава величину сензорске поруке.

3.5.1. Анализа перформанси архитектуре са пролазним уређајем

За процену перформанси, анализира се укупно протекло време од тренутка када су подаци доступни на серверу, односно на пролазном уређају, до

тренутка када корисник може да види визуелну интерпретацију сензорских података. Пре издвајања компоненти времена кашњења, биће дат кратак опис неколико важних блокова клијентске апликације реализоване као веб апликације на HTML5 платформи, као што је приказано на дијаграму секвенце на слици 7.



Слика 7 - Дијаграм секвенце са релевантним временима
за мерење кашњења преноса порука

Менаџер интеракције је одговоран за пријем порука са сензорским подацима преко изабраног протокола. Обично је његова функционалност делегирана одговарајућој програмској библиотеци (енг. *framework*) која реализује комуникациони протокол ослањајући се на *WebSocket* или *HTTP* базиране технике, нпр. *long-polling*.

Декодер порука - У зависности од употребљеног формата порука, позива се одговарајућа функција ради декодовања примљене поруке. Неке од коришћених функција су већ уграђене у прегледач веб, а за специфичне формате порука користе се функције из програмских библиотека отвореног кода.

Обрада података – Након декодовања, сензорски подаци се даље обрађују применом адекватних алгоритама у складу са сценаријом употребе или се

пореде са претходно пристиглим подацима. Треба добро проценити време извршавања ове обраде, како се не би нарушио одзив апликације и створио утисак корисника да је дошло до грешке, нарочито у условима извршавања са једном програмском нити, што је случај код *HTML-JavaScript* платформе.

Визуализација података - У овом тесту сензорски подаци су представљени једноставним вертикалним тракама, при чему висина траке представља однос последње измерене вредности и максималне измерене вредности сензорског опажања.

Укупно време кашњења је изражено у једначини (1) као:

$$Ttotal = Ttransfer + Tprocess + Trender \quad (4)$$

$$Ttransfer = Ttransmit + Tdecod \quad (5)$$

Тумачење једначине (4) је следеће: *Ttransfer* означава време потребно за пренос поруке са подацима са сервера на клијент, укључујући кодовање и декодовање поруке. *Tprocess* представља време потребно за обраду података на страни клијента, а *Trender* је време потребно за исцртавање графичких елемената на клијенту.

Једначина (5) описује компоненте *Ttransfer*: *Ttransmit* укључује време потребно за кодовање поруке на серверу као и време за које се подаци пренесу од сервера до клијента. У расположивим програмским библиотекама, време потребно за кодовање поруке некада није могуће посебно измерити, јер је саставни део стек протокола, тако да није издвојено у *Ttransfer*, већ се садржи у *Ttransmit*. *Tdecod* представља време потребно за декодовање примљених порука и конверзију у интерни објекат података на клијенту.

Избор одређеног протокола или формата порука има утицај на одређени део укупног времена кашњења *Ttotal*. На пример, избор протокола утиче на *Ttransmit*, док избор формата порука утиче на све компоненте *Ttransfer*. Осим тога, различите комбинације протокола и формата порука могу дати

неочекиване вредности *Ttransfer*. И коначно, конкретна имплементација веб платформе има доминантан утицај на сваки посебан временски период, а тиме следствено и на укупно време *Ttotal*. У овом тестирању, игнорисано је време *Tprocess* на клијенту након декодовања поруке, јер у великој мери зависи од конкретних примењених алгоритама, али у реалним ситуацијама не сме бити занемарено приликом анализе кашњења.

3.5.1.1. Имплементација и тестна подешавања

Клијентска веб апликација је развијена на стандардној HTML5 платформи коришћењем програмске библиотеке *Google Web Toolkit (GWT)*[29]. GWT је алат за развој софтвера са отвореним кодом, који омогућава програмерима да напишу код у *Java* програмском језику, који се затим преводи на *JavaScript* језик и извршава у веб прегледачу. Овај приступ је нарочито погодан ако се исти објекти користе на серверу и клијенту, због аутоматске конверзије. За комуникацију по моделу гурања, што укључује *WebSocket* и *long-polling* протоколе, коришћена је имплементација преко програмске библиотеке *Atmosphere* 2.3.0[30], јер скрива некомпатибилности прегледача веба и апликативних сервера. Серверски код је имплементиран у *Java* програмском језику и извршава се унутар *Apache Tomcat* апликативног сервера (верзија 8.0.21)[31].

Тестирање је вршено са укупно пет формата порука и то два текстуална и три бинарна формата порука. Два текстуална формата порука су стандардни *XML* и *JSON*. Бинарни формати порука су следећи. Први је *GWT* серијализатор, који је подразумевани бинарни формат за серијализацију објеката преко GWT механизма позива удаљених процедура (енг. *Remote Procedure Calls - RPC*). Други је *Google Protocol Buffers* (*PBF*) [70], већ описан у одељку 3.3.2 као ефикасни компактни бинарни формат. У недостатку званичне *PBF* имплементације за *JavaScript* или *GWT*, коришћена је *JavaScript*

[29] http://www.gwtproject.org/
[30] https://github.com/Atmosphere/atmosphere
[31] http://tomcat.apache.org/

имплементација отвореног кода *dcodeIO*[32]. Трећи бинарни формат је специјално конструисани формат за ово тестирање, *PBF* стринг формат, са циљем да се елиминише уношење прекомерног садржаја од стране *GWT* алата због стриминга бинарних порука коришћењем *UTF-8* стрингова. Због тога је вршено препакивање *PBF* бинарних порука у *UTF-8* стрингове који се састоје од 16-битних карактера, комбиновањем два суседна бајта из почетне *PBF* поруке у један *UTF-8* кодни карактер уз избегавање тзв. неисправних *UTF-8* кодова, који садрже сурогат кодове од D800 до DFFF.

Хардверска конфигурација тестног окружења је била следећа: серверски рачунар који има улогу пролазне компоненте, покреће *Intel* i5-3320M *CPU* и ради на 2.60GHz са 4GB RAM-а, а опремљен је са *Gigabit Ethernet* мрежном картицом. Оперативни систем (ОС) је *Windows* 7, из разлога да би се омогућило коректно поређење у каснијем тесту са веб *платформом Microsoft Silverlight*, а та платформа на серверској страни захтева апликативни сервер *Microsoft Internet Information Server* (*IIS*) који се извршава само на *Windows* ОС-у. Клијентске апликације су се извршавале на *Google Chrome* прегледачу веба, унутар *Windows 7* ОС-а, а у *LAN* околини сервера. За синхронизацију часовника клијентског и серверског рачунара разматрана су два синхронизациона протокола: *Precise Time Protocol (PTP)* [81] и *Network Time Protocol (NTP)* [82]. Иако обећава ефикасну синхронизацију у *LAN* окружењу, *PTP* је елиминисан због недостатка одговарајуће имплементације за *Windows* ОС. Такође је елиминисан и *NTP* протокол јер је часовник на серверу одступао у распону од неколико милисекунди до око 80 милисекунди у односу на најближе *NTP* временске сервере (Мађарска, Аустрија, Немачка). На крају, за синхронизацију часовника клијента и сервера коришћен је алат *Domain Time II*[33] развијен од стране компаније *Greyware Automation Products*, који је заправо модификована имплементација *PTP* протокола. Користећи овај алат, одступање часовника клијента у односу на сервер је било мање од 0,5ms током свих тестирања.

[32] https://github.com/dcodeIO/ProtoBuf.js

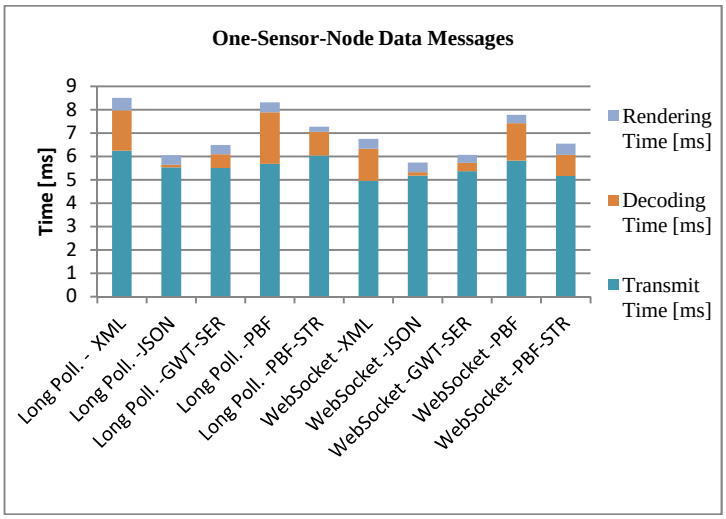
[33] https://www.greyware.com/software/domaintime/

3.5.1.2. Резултати и дискусија

Резултати теста за *HTML5* веб апликацију су представљени у табели 2 и табели 3, као и на дијаграмима на слици 8 и слици 9. Одмах је уочљиво да *PBF* стринг формат производи знатно краће поруке од било ког другог формата порука. Насупрот томе, време преноса таквих порука је веће од формата порука *JSON* и *GWT* серијализатора. Објашњење за ово је да иако је препаковање основне *PBF* поруке у и из стринга једноставна операција, она троши извесно време на страни сервера и клијената. Треба приметити да унутрашња конверзија *Atmosphere-GWT PBF* поруке, производи за фактор око 3,6 дужу поруку од оригиналне *PBF* поруке, због конверзије *UTF-8* стринга који се користи за пренос преко *HTTP* везе. Одатле потиче мотивација за ручно паковање *PBF* поруке што резултира краћим стрингом за фактор више од 7.

Табела 2 – Резултати тестова за поруке са једног сензорског чвора

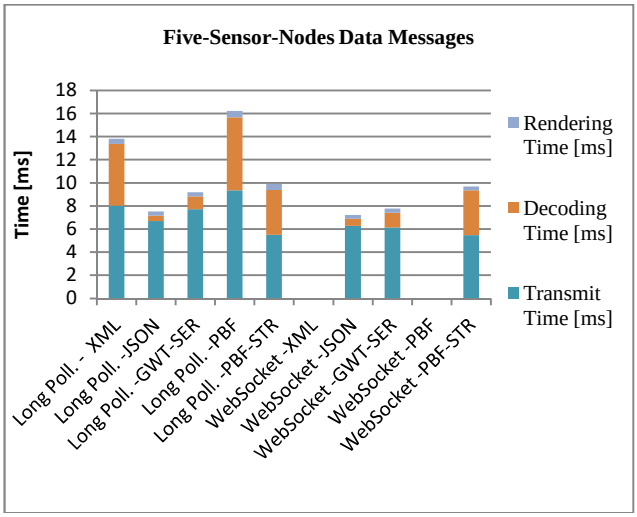
Протокол	Long Poll.	Long Poll.	Long Poll.	Long Poll.	Long Poll.	Web Sock.	Web Sock.	Web Sock.	Web Sock.	Web Sock.
Кодовање	XML	JSON	GWT -SER	PBF	PBF-STR	XML	JSON	GWT -SER	PBF	PBF-STR
Сензорских чворова	1	1	1	1	1	1	1	1	1	1
Величина корисног дела поруке [знак]	2322	1478	1153	1966	272	2322	1478	1153	1966	272
Ttotal [ms]	8.51	**6.05**	6.52	8.32	7.28	6.76	**5.74**	6.09	7.78	6.55
Ttransfer [ms]	7.98	**5.65**	6.13	7.89	7.05	6.33	**5.33**	5.75	7.42	6.07
Ttransmit [ms]	6.25	**5.54**	5.51	5.68	6.04	4.96	**5.18**	5.37	5.82	5.16
Tdecod [ms]	1.73	**0.11**	0.59	2.21	1.01	1.37	**0.15**	0.36	1.60	0.91
Trender [ms]	*0.53*	*0.40*	*0.39*	*0.43*	***0.23***	*0.43*	*0.41*	*0.34*	*0.36*	*0.48*



Слика 8 – Резултати тестова за поруке са једног сензорског чвора

Табела 3 - Резултати тестова за поруке са пет сензорских чворова

Протокол	Long Poll.	Long Poll.	Long Poll.	Long Poll.	Long Poll.	Web Sock.	Web Sock.	Web Sock
Кодовање	XML	JSON	GWT -SER	PBF	PBF-STR	JSON	GWT -SER	PBF-STR
Сензорских чворова	5	5	5	5	5	5	5	5
Величина корисног дела поруке [знак]	11281	7163	4485	9578	1343	7163	4485	1343
Ttotal [ms]	13.81	**7.51**	9.26	16.22	9.91	**7.22**	7.83	9.69
Ttransfer [ms]	13.36	**7.17**	8.90	15.68	9.39	**6.89**	7.45	9.34
Ttransmit [ms]	8.03	**6.68**	7.73	9.36	5.49	**6.28**	6.13	5.47
Tdecod [ms]	5.33	**0.49**	1.10	6.32	3.90	**0.61**	1.27	3.87
Trender [ms]	*0.45*	*0.34*	*0.36*	*0.54*	*0.52*	***0.33***	*0.38*	*0.35*



Слика 9 - Резултати тестова за поруке са пет сензорских чворова

Додатни проблем са *PBF* порукама је да све коришћене програмске библиотеке, а то су *GWT*, *Atmosphere* и *dcodeIO*, користе сопствене структуре за представљање великих целих бројева и низова бајтова у *JavaScript* програмском окружењу, што уноси прилична увећања услед припреме података за одговарајући формат *dcodeIO PBF* бафера.

Резултати тестова показују да *JSON* поруке имају најмању вредност кашњења у *HTML5* тесту, без обзира на употребљени протокол комуникације. Главни разлог за то је оптимизација прегледача веба за декодовање *JSON* порука, јер се та операција не интепретира, већ директно извршава унутар извршног окружења.

У случају других порука, извршно окружење прегледача веба интерпретира одговарајући *JavaScript* програмски код за декодовање. Стога се може закључити да у окружењу са брзом комуникацијом, изузетно кратко кодовање порука не пружа никакву посебну корист ако операција кодовања/декодовања није добро оптимизована. Даље, *WebSocket* има мање кашњење од *long-polling* протокола, нарочито због честе размене заглавља. Разлика у преносу ових протокола за исте формате порука је врло мала, обично мање од 1ms. Такође, графичко исцртавање типично траје мање од 1ms у тестовима, тако да то није параметар од утицаја на перформансе.

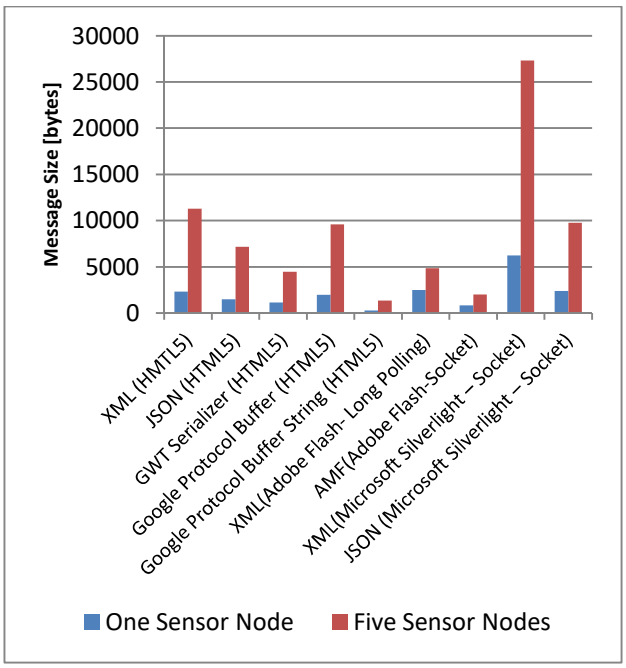
3.5.1.3. Поређење резултата кодовања порука

Овај тест је проширен тестирањем на веб платформама базираним на уграђеним додацима прегледача веба и то *Microsoft Silverlight* и *Adobe Flash*. Намера је била да се покаже како коришћењем истих комуникационих протокола и формата порука, перформансе могу знатно варирати од платформе до платформе.

Вредности величина порука произведених од свих начина кодовања порука на тестираним платформама приказане су у табели 4 и на слици 10.

Табела 4 – Величина порука за све испитиване типове формата порука

Формат порука (Платформа)	Величина корисног дела поруке – један сензорски чвор [знак]	Величина корисног дела поруке – пет сензорских чворова [знак]
XML (HMTL5)	2322	11281
JSON (HTML5)	1478	7163
GWT Serializer (HTML5)	1153	4485
Google Protocol Buffer (HTML5)	1966	9578
Google Protocol Buffer String (HTML5)	**272**	**1343**
XML (Adobe Flash- Long- Polling)	2484	4838
AMF (Adobe Flash-Socket)	820	1997
XML (Microsoft Silverlight – Socket)	6227	27301
JSON (Microsoft Silverlight – Socket)	2403	9774



Слика 10 - Величина порука за све испитиване типове формата порука

PBF стринг креира најкраћу поруку међу свим кодовањима порука. У случају порука са 5 сензорских чворова, *Adobe AMF* кодовање производи 50% дужу поруку, 1997 знакова у односу на 1343 знакова, али то је знатно мања дужина поруке од било ког другог кодовања порука. За поруке са једним сензорским чвором, разлика између *PBF* стринга и *AMF* поруке је чак три пута, 272 наспрам 820 знакова, а у односу на најдужу поруку *XML* на *Silverlight* платформи, разлика је скоро 23 пута, 272 према 6227 знакова. Даље, у случају порука са 5 сензорских чворова, величина *XML* поруке на *Adobe Flash*-у је много мања од *XML* поруке на *HTML5* платформи, 4838 знакова у односу на 11281 знакова, док у случају поруке са једног сензорског чвора, *XML* поруке на обе платформе имају сличне величине. Ово имплицира да је увећање од заглавља у *Adobe* поруци значајно када је користан садржај поруке мали, али заглавље остаје компактно за веће садржаје поруке. Међутим, увећање од заглавља поруке је знатно веће код *Silverlight* платформе него за друге платформе пошто *XML* и *JSON* поруке имају много веће величине него на друге две платформе. Разлог лежи у имплементацији комуникационог протокола по узору на стандардни веб сервис, што подразумева коришћење *SOAP* порука и заглавља који имају велику иницијалну величину.

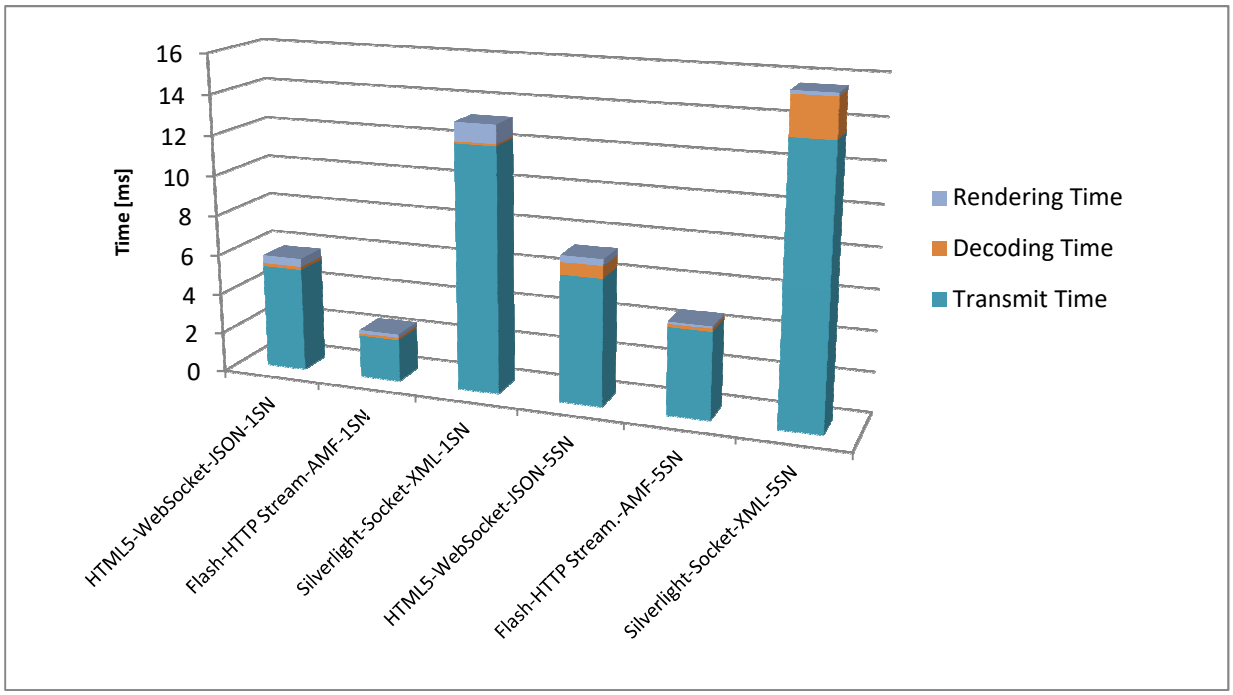
3.5.1.4. Поређење перформанси са имплементацијама на другим веб платформама

Најбољи резултати тестирања по платформама приказани су у табели 5 и на слици 11. На основу извршених тестова, најефикаснија комбинација је *Adobe Flash* апликација заснована на *HTTP streaming* протоколу и *AMF* кодовању порука. Ова комбинација је постигла најкраће кашњење и стога нуди комуникацију са највећом брзином испорука порука: за поруке са једним сензорским чвором, просечно укупно кашњење је 2,41ms, док код порука са пет сензорских чворова, кашњење је 4,56ms. Апликација на *HTML5* платформи постиже кашњење од 5,74 ms и 7,22ms за поруке са једним и пет сензорских чворова. Најефикаснија комбинација у случају *HTML5* апликације је *WebSocket* протокол заједно са *JSON* форматом порука, пре свега због оптимизације прегледача веба за декодовање *JSON* порука. Резултати теста за *Silverlight* апликације су значајно слабији него за друге две платформе (13,16ms и 15,55ms) и као што је већ продискутовано, кривицу сноси унутрашња архитектура *Windows Communication Foundation* библиотеке заснована на *SOAP* порукама, која уноси знатно увећање величине порука.

Генерално, у тестираном окружењу, комуникациони протоколи имају мањи утицај на кашњење од начина кодовања порука за одређену платформу. Ови резултати се могу објаснити чињеницом да због велике брзине комуникације у *LAN*-у, постоји мала разлика у времену преноса за одређени распон величина порука. Међутим, може се приметити да имплементација кодовања и декодовања порука значајно варира међу платформама, упркос истој комплексности процесирања. Најбоља илустрација за ово је време декодовања порука за *XML* и *JSON* поруке. Док веб прегледачи имају најоптимизованију *JSON* имплементацију, *Silverlight* декодује *XML* поруке најбрже. То доводи до закључка да свака од три тестиране платформе има најмање кашњење са омиљеним форматима порукама.

Табела 5 – Најбољи остварени резултати кашњења преноса порука на веб платформама за пренос порука са једног и пет сензорских чворова

Платформа	HTML5	Adobe Flash	Microsoft Silverlight	HTML5	Adobe Flash	Microsoft Silverlight
Протокол	WebSocket	HTTP Streaming	Socket	WebSocket	HTTP Streaming	Socket
Кодовање	JSON	AMF	XML	JSON	AMF	XML
Број сензорских чворова	1	1	1	5	5	5
Корисна величина поруке	**1478**	1639	6227	7163	3994	27301
Ttotal [ms]	5.74	**2.41**	13.16	7.22	**4.56**	15.55
Ttransfer [ms]	5.33	**2.23**	12.23	6.89	**4.45**	15.38
Ttransmit [ms]	5.18	**2.11**	12.13	6.28	**4.29**	13.49
Tdecod [ms]	0.15	0.12	**0.1**	0.61	**0.16**	1.89



Слика 11 - Најбољи остварени резултати кашњења преноса порука на веб платформама за пренос порука са једног и пет сензорских чворова

Са екстремно кратким кодирањем порука преко *PBF* стринг формата, нису постигнути добри резултати у *HTML5* апликацији, углавном због субоптималне имплементације која је резултат комбиновања неколико различитих програмских библиотека. То може да важи и са другим кодовањима порука, јер обично веома кратке величине порука захтевају знатне процесне ресурсе односно време, што може да превагне у односу на

брзину преноса и да се не испуне очекивања перфомански. Сокет протокол је ефикаснији од *HTTP* long-polling протокола на свим плаформама, али не претерано значајно. Интересантно да је најмање кашњење у тестирању остварено са *HTTP* стриминг протоколом на *Adobe Flash* платформи. Објашњење лежи у чињеници да је у питању једносмерна комуникација са релативно малим увећањем садржаја услед заглавља.

3.5.2. Анализа перфомански архитектуре са брокером порука

Клијентска апликација је у овом тесту била идентична претходној, осим што су се за комуникацију користили протоколи за размену порука *MQTT*, *AMQP*, *XMPP* и *DDS* имплементирани у *JavaScript* програмском језику. У овој тест апликацији, није било потребе за синхронизацијом часовника између рачунара који покреће произвођача сензорских података односно пролазног уређаја и потрошача тих података, односно клијента, јер оба актера се извршавају на истом рачунару, док су се брокери порука извршавали на другом серверском рачунару. Мерење кашњења је у овом случају заправо мерење времена обиласка поруке. Сензорска опажања су такође генерисана у интервалу од 1s. Поруке садрже податке са једног сензорског чвора, а кодоване су у *JSON* формату.

3.5.2.1. Имплементација

Пошто је циљ теста тестирање перформанси архитектуре услед промена IoT апликативног протокола за размену порука, изабране су слично као у претходном тесту, клијентске веб апликације са *JavaScript* имплементацијом протокола, док је *WebSocket* изабран као основни комуникациони протокол. На страни објављивача сензорских података, изабрана је *Java* имплементација протокола како би се осигурала најбоља могућа преносивост, односно покривеност што већег броја система.

3.5.2.2. Анализа измерених резултата

Током првог теста мерено је кашњење преноса сензорских порука од објављивача до претплатника, а то је време *Ttransfer* описано у претходном

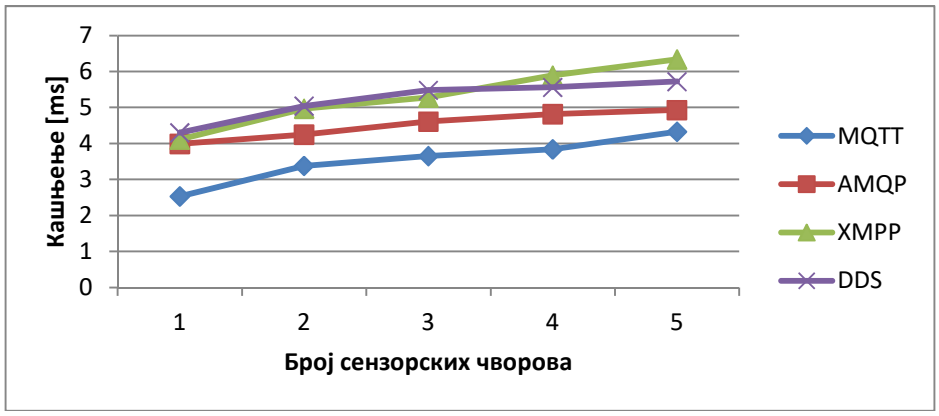
поглављу и односи се на временски тренутак након декодовања поруке на страни клијента. Измерена времена кашњења приказана су у табели 6 и на слици 12. Најмање кашњење је настало код *MQTT* протокола, а затим следи *AMQP*, док су разлике између *XMPP* и *DDS* протокола занемарљиве. У случају теста за поруке са једним сензорским чвором, *MQTT* постиже кашњење од само 2,53ms. Кашњење пропорционално расте са повећањем величине поруке, тако да су резултати мерења прилично предвидљиви. *MQTT* као веома лаган протокол, уноси мало прекорачење у фази синхронизације протокола (енг. *handshaking*), док код *AMQP* долази до изражаја оптимизовано бинарно кодовање порука што доводи до ефикасног рада са сензорским подацима. *XMPP* очигледно уноси веће прекорачење у односу на претходна два протокола, због процеса убацивања сензорских порука у *XML* станца структуре дефинисане у овом случају у проширењу стандарда *XEP-0060: Publish-Subscribe*. Веће кашњење код *DDS* протокола је последица коришћења посредне компоненте ради конверзије протокола за веб платформу, односно за *WebSocket*, тако да иако је ефикасан типизовани канал за комуникацију, овде то не долази до изражаја.

Табела 6 – Измерена кашњења за протоколе за пренос порука IoT апликативног слоја

Протокол	Број сензорских чворова	1 сензорски чвор	2 сензорска чвора	3 сензорска чвора	4 сензорска чвора	5 сензорска чвора
	Корисна величина поруке [знак]	1475-1477	2896-2901	4316-4321	5734-5746	7157-7169
MQTT	*Ttransfer* [ms]	**2.53**	**3.38**	**3.65**	**3.84**	**4.33**
AMQP	*Ttransfer* [ms]	3.99	4.25	4.61	4.82	4.93
XMPP	*Ttransfer* [ms]	4.11	4.97	5.28	5.89	6.34
DDS	*Ttransfer* [ms]	4.3	5.04	5.48	5.56	5.72

Сва измерена времена су реда величине неколико милисекунди, док је резолуција тајмера 1ms, па иако су резултати просечне вредности релативно великог броја пренесених порука, у питању су апроксимативне вредности. Да би се смањила грешка због резолуције тајмера, али и да се измери максималан пропусни капацитет сваког протокола, спроведен је још један тест у којем се

мерио број пренетих порука које су обишле пун круг, са следећим сценаријем: произвођач сензорских података објављује поруку и након пријема, претплатник одмах пошаље ту поруку произвођачу.



Слика 12 - Измерена кашњења за протоколе за пренос порука IoT апликативног слоја

Процес се наставља тако што произвођач одмах поново шаље исту поруку претплатнику итд. Резултати мерења садрже број порука које су обишле комплетан круг од произвођача до претплатника и назад, у временском интервалу од 100s. Другим речима, добијено је:

$$Ttransmission\,[\mathrm{s}] = 100/2*\text{број послатих порука} \qquad (6)$$

Време преноса *Ttransmission* је различито од времена *Transmit* анализираног у одељку 3.5.1, јер у овом мерењу време кодовања поруке није укључено, већ на компоненте овог времена искључиво утичу имплементација протокола за пренос.

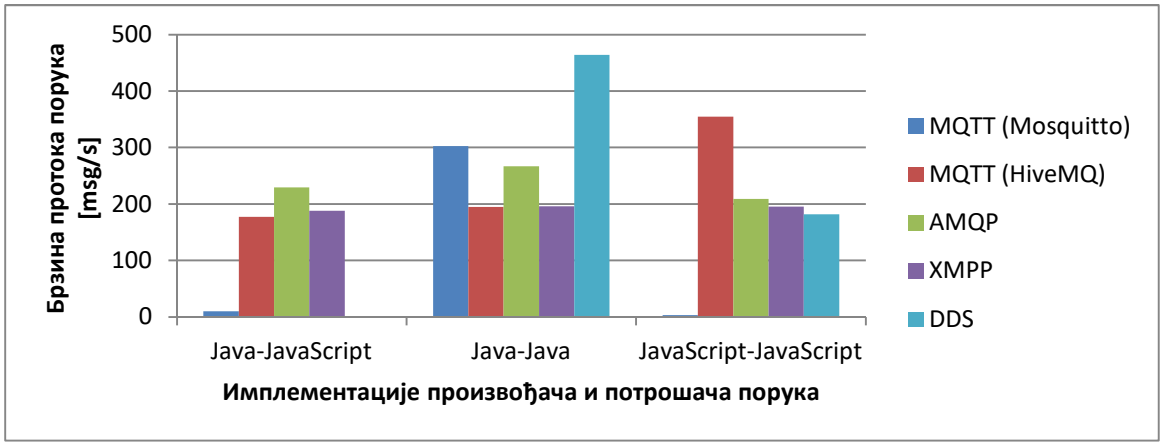
Првобитна намера је била да се изведе нови тест у истој конфигурацији као у претходном случају приликом мерења кашњења, са *Java* клијентом код произвођача, а *JavaScript* имплементацијом на страни клијента. Међутим, тест са *DDS* протоколом није успео, јер његов *Java* клијент није примао поруке од *JavaScript* клијента. Међутим, тест за *DDS* протокол у конфигурацији са оба *Java* клијента, као и са оба *JavaScript* клијента је функционисао, тако да је иста конфигурација коришћена и за три остала протокола како би се адекватно

упоредили резултати брзине протока порука и резултати су приказани у табели 7 и на слици 13.

Табела 7 – Брзине протока преноса порука протокола IoT апликативног слоја

Протокол	Java – JavaScript клијенти		Java – Java клијенти		JavaScript – JavaScript клијенти	
	Брзина протока порука [пор/s]	*Ttransmission* [ms]	Брзина протока порука [пор/s]	*Ttransmission* [ms]	Брзина протока порука [пор/s]	*Ttransmission* [ms]
MQTT (Mosquitto)	9.85	50,76	302.48	1,65	3,33	150,15
MQTT (HiveMQ)	177.37	2,82	194.60	2,57	**354,64**	**1,41**
AMQP*	**229.38**	**2,18**	266.97	1,87	208,97	2,39
XMPP	187.85	2,66	196.04	2,55	195,63	2,56
DDS	X	X	**463.94**	**1,08**	181,88	2,75

Ови тестови показују веома различите резултате. У случају *AMQP* и *XMPP* протокола, вредности брзине протока порука су усаглашене са тестом кашњења: *AMQP* постиже благо већу брзину протока порука од *XMPP* протокола, 229,38 пор/s према 187,87 пор/s. Међутим, мора се нагласити да резултати за *AMQP* протокол значајно варирају са понављањем тестова у више наврата, а најбољи резултати су измерени одмах по покретању брокера порука.



Слика 13 - Брзине протока порука IoT протокола апликативног слоја

За *MQTT* протокол, брзина протока порука за случај тест конфигурације *Java-Java* клијената је очекивано и знатно је већа него за *AMQP* и *XMPP* протоколе, са 302,48 пор/s у односу на 266,97 пор/s и 196,04 пор/s респективно. Међутим, у примарном тест подешавању са *Java* <-> *JavaScript* клијентима као и за

JavaScript <-> *JavaScript* клијенте, резултати за *MQTT* протоколи су веома слаби, испод 10 порука у секунди, тј. 9,85 пор/s и 3,33 пор/s. Разлог за ову аномалију лежи у унутрашњој архитектури најпопуларнијег *MQTT* брокера порука *Mosquitto* (тестирање је извршено са верзијама до 1.4.8.) која утиче на перформансе прослеђивања порука преко *WebSocket*-а (користи се имплементација *libwebsocket* библиотеке верзија 2.0.2.), и за исправно функционисање је неопходна модификација имплементације главне обраде улазно/излазних догађаја, што се може очекивати у некој од будућих верзија. Тест је поновљен коришћењем другог брокера порука за *MQTT* протокол *HiveMQ* и резултати су такође приказани у табели 7 и на слици 13. Са *HiveMQ* брокером порука, брзина протока порука код *JavaScript* <-> *JavaScript* клијената, је највећа од свих тестираних протокола, достигавши 354,60 пор/s. Међутим, у тесту са *Java*-*Java* клијентима, *HiveMQ* тест знатно заостаје у односу на *Mosquitto* тест, односно постиже 35,7 % мању брзину протока, 194,60 пор/s у односу на 302,48 пор/s код *Mosquitto*. Такође, тест кашњења је са *HiveMQ* брокером показао знатно знатно веће кашњење преноса порука у односу на тест са *Mosquitto* брокером. Други неочекивани резултат је више од 50% већа брзина протока порука за *DDS* протокол него за други најбољи протокол у конфигурацији са *Java*-*Java* клијентима, *MQTT*, и то 463,94 пор/s према 302,48 пор/s. Објашњење лежи у чињеници да се код *DDS* протокола успоставља директна *TCP* веза између клијената, без икаквог посредовања компонената за откривање или преусмеравање учесника у протоколу. На другој страни, брзина протока са *JavaScript* <-> *JavaScript DDS* клијентима је нижа него за све остале протоколе, 181,88 пор/s. Као и код теста мерења кашњења, ово је последица конверзије протокола за *WebSocket* које врши *Java* посредни сервер.

3.5.2.3. Резиме и закључак

Треба уочити да најбоље време за пренос порука код тестирања са једним сензорским чвором за архитектуру са пролазним уређајем и апликативним сервером и *HTML5* веб апликацијом износи 5,33ms (табела 5), што је знатно веће од било ког времена преноса код протокола за размену порука приказаних у табели 6, у распону од 2,53ms до 4,3ms. Иако први тест има мање

прецизно мерење времена јер механизам синхронизације часовника додаје извесну грешку, може се закључити да је архитектура са брокером порука коришћењем IoT апликативних протокола са објави/претплати механизмом размене порука ефикаснија и уноси нешто ниже кашњење током преноса порука него конзервативна имплементација асинхроне испоруке порука преко апликативног сервера услед имплементације са вишеструким слојевима за обраду протокола.

Као опште запажање, шароликост имплементација *JavaScript* клијената и имплементација брокера порука доприноси да одлука о одабиру IoT протокола апликативног слоја за пренос порука у архитектури са брокером порука далеко од једноставног. Грешка у имплементацији *MQTT* брокера порука *Mosquitto*, која резултира веома лошим перформансама у случају коришћења *JavaScript* клијента и *WebSocket* комуникације спречава препоручивање *MQTT* протокола за било који сценарио IoT апликација са веб клијентима. На другој страни, перформансе *MQTT* протокола засноване на *HiveMQ* брокеру порука су прилично сличне другим протоколима.

Генерално узевши, сваки протокол има одређене мане: *MQTT* перформансе изразито зависе од коришћеног брокера порука; *DDS* захтева употребу додатног посебног посредног сервера у случају веб клијента, што ограничава флексибилност; *AMQP* има највећу флуктуацију брзине протока порука од свих протокола, што је параметар који је у неким апликацијама од посебне важности; иако *XMPP* протокол нема крупнијих недостатака, измерена времена кашњења и брзина протока порука никада нису међу најбољима за било који тестни случај. И на крају, ако апликативни сценарио дозвољава коришћење само *Java* клијената, онда је *DDS* убедљиво најефикаснији IoT апликативни протокол за размену порука, јер остварује од 50% до више од 136% већу брзину протока порука од осталих протокола.

4 КЛАСИФИКАЦИЈА ПОСТОЈЕЋИХ РЕШЕЊА СА СЕМАНТИЧКИ ЗАСНОВАНОМ ИНТЕГРАЦИЈОМ СЕНЗОРСКИХ МРЕЖА

Архитектуре са семантички заснованом интеграцијом сензорских мрежа су истакнуте у поглављу 3.4.3, као један тип IoT платформи са напредним функционалностима. Пред такве архитектуре постављају се многи пројектантски изазови и проблеми који се требају решити у циљу ефикасне интеграције сензорских мрежа, упркос чињеници да истраживачи приступају овом проблему са различитих гледишта као што су подршка специфичним случајевима употребе или апликативним доменима и прилагођавању крајњим корисницима. Реалне архитектуре се ипак фокусирају на ефикасно решавање подскупа датог скупа параметара.

4.1. Кључни пројектантски параметри архитектура за семантичку интеграцију сензорских мрежа

4.1.1. Основна организација

Типичан приступ интеграцији сензорских мрежа је кроз вишеслојну организацију која логички дели компоненте система, као у примеру генеричке архитектуре из прошлог поглавља. Главнина семантички-заснованих функција се налази типично у процесном слоју који обухвата функције за филтрирање, агрегацију и трансформацију сензорских података, одржавање семантичког модела података, као и више интелигентних функција којима се омогућава семантички засновано закључивање. Варијације у организацији дозвољавају различите интерфејсе између слојева, даље раздвајање на више слојева или груписање неких од стандардних слојева у један слој. У зависности од расподеле и интеракције између слојева постоје и клијент-сервер, хијерархијске и *peer-to-peer* организације.

4.1.2. Скалабилност

Скалабилност се дефинише као одговарајуће понашање система у циљу обезбеђивања подразумеваних перформанси у случају пораста волумена сензорских података или повећања броја подржаних или активних корисника односно апликација. Ефикасна архитектура треба да одговарајуће скалира односно да испуни потребне захтеве за перформансама. Типичан проблем са скалабилношћу у постојећим архитектуралним приступима је случај када је једна или више од критичних функција читавог система централизована, без обзира која је то функционалност. То може бити централизовани систем за складиштење и архивирање сензорских података, јединица за прикупљање сензорских података, систем за обраду и извршавање упита, решавач упита (енг. *query resolver*), пружалац сервиса итд. Пројектантски је изазов да се ове критичне функције реализују дистрибуирано у циљу остваривања одговарајуће скалабилности.

4.1.3. Модалност сензорских података

Постојеће распоређене хетерогене сензорске мреже варирају у карактеристикама сензорских података. Да би се остварила ефикасна фузија разнородних података, архитектура би требало подржавати што више различитих модалитета података. То значи да корисници могу имати могућност добијања сензорских опажања као резултат неких догађаја (прекорачење неких дефинисаних вредности или задовољење неких сложених услова), затим сензорска мерења прикупљена у правилним временским размацима (аквизиционе), токове сензорских података у условима њиховог интензивног генерисања, архивиране/складиштене податке и агрегиране или филтриране податке. Подржавањем свих модела интеракције односно протокола апликативног слоја описаних у одељку 3.2, остварује се потребан услов за ефикасну испоруку побројаних модалитета сензорских података заинтересованим корисницима. Са друге стране, правилна конфигурација сакупљања сензорских података мора осигурати да не дође до преоптерећења или пренапуњености система услед неадекватно одабране стопе прикупљања нових сензорских података.

4.1.4. Флексибилност подржаних сензорских мрежа

У поглављу 2 је дат кратак осврт на неке од стандардних комуникационих протокола и оперативних система на које се ослањају сензорски чворови као извори сензорских опажања. Разноликост хардверских уређаја и системског софтвера који они извршавају намеће проблем како ефикасно интегрисати све те хетерогене изворе сензорских података. Уопштено говорећи, могуће је оставити провајдеру сензорских података да се прилагоди одређеном интерфејсу архитектуре или платформе за интеграцију сензорских мрежа, или архитектура обезбеђује сопствену имплементацију за интеракцију са одређеном класом сензорских мрежа или кроз приступ путем *средњег слоја* (енг. *middleware*). *Средњи слој* је софтверска инфраструктура која повезује платформе сензорских мрежа, оперативне системе, стек протокол и апликације [83]. Постоји неколико типова решења за приступ сензорским мрежама преко средњег слоја као што су: приступ у стилу база података, парадигма макро-програмирања, механизам приступа кроз прослеђивање порука на бази модела објави-претплати, виртуелна машински заснована интеракција, приступ преко мобилних агената и сервисно-оријентисани приступ [84].

4.1.5. Подршка техничким способностима сензорских мрежа

Најчешћи принцип код пројектовања платформи за интеграцију сензорских мрежа је следећи: техничке специфичности и комплексности сензорских мрежа су сакривене од крајњних корисника и не разматрају се приликом прикупљања сензорских опажања. Међутим, у циљу побољшања енергетске ефикасности и ради добијања прецизнијих информација из осматраног реалног окружења као и ради оптималног управљања мрежом и приступа сензорским подацима, неке архитектуре могу одржавати релевантне информације о техничким могућностима сензора, њихову тачност, ограничења, као и мрежне топологије и протоколе рутирања. Таква оптимизација може увести додатне тешкоће због потребе за паралелним обављањем других операција. Додатна комплексност потиче од

потребе да се уведу посебни безбедоносни механизми у циљу провере које информације ће бити расположиве којим типовима корисника. Техничке карактеристике разнородних сензора се ефикасно описују кроз сензорске онтологије.

4.1.6. Управљање сензорским мрежама и актуаторским функцијама

Неке платформе могу дозволити крајњим корисницима да имају потпуну или ограничену контролу над ресурсима сензорских мрежа, што може укључити и подешавање параметара режима рада сензора, или можда приступ актуаторским функцијама. Приступ овом скупу функција се не може једноставно делити између заинтересованих корисника, тако да мора бити обезбеђен одговарајући механизам арбитраже заједно са полисом ауторизације.

4.1.7. Онтологије

Флексибилнији начин за интерпретацију и управљање подацима из сензорских мрежа је могућ дефинисањем основних концепата из домена апликација и њихових односа са сензорима кроз развој одговарајућих онтологија. Коришћењем тако дефинисаних концепата, корисници и апликације су у могућности да комуницирају са интегрисаним сензорским мрежама на вишем нивоу, чиме се омогућава интероперабилност. Guarino [85] је предложио развој "различитих врста онтологија према њиховом степену општости" и класификовао их је као: онтологије највишег нивоа (енг. *top-level ontologies*) или горње онтологије, које описују опште концепте независно од било ког домена; онтологије домена и задатака које се баве одређеним доменом или задацима специјализацијом концепата из горњих онтологија; и апликативне онтологије које обухватају улоге ентитета приликом обављања дефинисаних активности. Овај приступ са мрежом онтологија је често коришћен у семантичким сензорским мрежама, кроз развој одвојених онтологија за сензорски и апликативни домен, а употребом неких од популарних онтологија горњег слоја за постизање интероперабилности. Најсвеобухватнији рад на пољу сензорских онтологија је урадила

Инкубаторска група за семантичке сензорске мреже (енг. *Semantic Sensor Network Incubator Group*) WWW конзорцијума (енг. *World Wide Web Consortium – W3C*), која је развила Онтологију семантичких сензорских мрежа (енг. *Semantic Sensor Network Ontology*) која обухвата сензорске уређаје и сензорске мреже, њихове техничке могућности и сензорска опажања (енг. *sensor observations*) [12, 86, 87]. Ова онтологија се ослања на *DOLCE+DnS UltraLite* горњу онтологију и тежи да постане широко прихваћена, јер омогућава проширења са доменски специфичним онтологијама. Креиране онтологије могу више да моделују техничке или апликативне аспекте и да користе различите горње онтологије. Између осталог, Compton и сарадници [11] истичу *OntoSensor* онтологију [88] која је изграђена на *OGC SWE SensorML* спецификацији и користи горњу онтологију *IEEE Suggested Upper Merged Ontology* [89], међутим, она не покрива сензорска опажања и мерења. Такође једна од популарних горњих онтологија је *Semantic Web for Earth and Environmental Terminology (SWEET)* онтологија развијена у свемирској агенцији NASA [90]. Као што је описано у поглављу 2.5, за представљање онтологија, стандардно се користе семантичке веб технологије *OWL* [43] и *RDFS* [42].

4.1.8. Примена семантичког приступа

Увођењем онтологија у систем омогућује се употреба семантичких технологија и приступа унутар различитих нивоа архитектуре. У складу са прихваћеним информационим моделом, семантички приступ би се могао користити за опис расположивих сервиса, за представљање сензорских опажања и мерења, за опис особина извора сензорских података, претрагу података преко семантичких упита и за опис карактеристика интерних процесних компонената.

4.1.9. Модел представљања података

Следеће значајно питање дизајна је интерни сензорски модел података у циљу представљања прикупљених сензорских опажања и сензорских описа. Генерално, користећи једноставан формат сензорских података, као на пример модел заснован на бинарном или релационом формату, омогућавају

се боље перформансе, потребно је мање меморије за складиштење и мањи пропусни опсег за комуникацију, али пошто је то модел ниског нивоа, неминовно долази до слабе интероперабилности у условима хетерогених сензорских података. Као формат података средњег нивоа може се користити *XML*, пошто задржава захтеве за складишним капацитетима разумно ниским, омогућава известан степен интероперабилности између сензорских мрежа на основу синтаксних правила, али нема могућност додавања семантике сензорским подацима чиме би се обогатио њихов опис. За представљање сензорских података више апстракције користе се семантичке веб технологије и то *RDF* модел погодан за изражавање концепата и односа дефинисаних у прихваћеним онтологијама.

4.1.10. Апликативни интерфејс и формат података

Архитектуре за семантичку интеграцију сензорских мрежа деле исте апликативне интерфејсе и формате података у циљу интеракције са корисницима као и архитектуре описане у поглављу 3. Користе се стандардни интерфејси и модели интеракције са синхроном и асинхроном испоруком података, а некада се уз семантички представљене податке по *RDF* моделу могу паралелно користити неки не-семантички формати попут *XML* и *JSON*.

4.1.11. Упитни језици

Најједноставнији начин креирања корисничких упита је коришћењем приступа који омогућава корисницима да наведу произвољне параметре унутар самог захтева за подацима, што је веома слично *HTTP* моделу захтева. Међутим, ипак најприкладнији начин за кориснике да приступе и захтевају жељене сензорске податке је преко упитних језика, чији је избор под утицајем али не мора бити ограничен изабраним интерним моделом података. Ако је модел сензорских података заснован на релационом моделу, SQL језик или његови деривати су природни избор. Овај избор оставља кориснике да се баве сировим подацима, па је неопходна додатна обрада како би се добило више информација и знања из података. Представљањем сензорских података преко XML формата, оставља се могућност различитих формата за

изражавање упита, а стандардни је *XPath* [91]. *XPath* језик омогућава упите којима се издвајају жељени подаци кретањем унутар стабла података представљеним *XML* форматом. Надскуп *XPath* упитног језика је *XQuery* [92], знатно изражајнији упитни језик који омогућава конструкте налик на SQL језик. Усвајањем вишег нивоа апстракције представљања сензорских података и апликативних концепата кроз *RDF* тројке, очекује се да корисници изразе своје упите користећи *SPARQL* језик. У постојећим платформама, могу се пронаћи надоградње и проширења ових упитних језика, рецимо у циљу постављања посебних просторно-временских параметара. У случају постојања неколико паралелних формата података у различитим нивоима унутар архитектуре система, потребно је извршити превођење или трансформацију упита на неком нивоу.

4.1.12. Закључивање

Као једна од главних предности увођења семантичких технологија у архитектуре за интеграцију сензорских мрежа је способност извођења знања високог нивоа из прикупљених сензорских опажања, закључивањем на бази описа из онтологија и скупа логичка правила. Процес закључивања укључује побољшану претрагу са генерализованим концептима, детекцију догађаја високог нивоа преко сирових сензорских података (нпр. закључивање да је напољу мећава ако је температура испод 0 степени, а влажност изнад неког нивоа), логично и доменско закључивање применом одређених алгоритама коришћењем логичког програмирања и описних логика.

4.1.13. Откривање сервиса

Функција која омогућава корисницима да добију информације о доступним сервисима које пружа платформа за интеграцију сензорских мрежа назива се откривање сервиса (енг. *discovery of services*). Насупрот томе, провајдери сензорских података могу регистровати њихове услуге кроз одговарајуће механизме описа. Одређена архитектура може омогућити динамичну регистрацију сервиса и самим тим откривање таквих сервиса, док друге

архитектуре могу имати само статички скуп доступних сервиса. Корисници могу имати могућност да открију одговарајуће сервисе путем семантичких упита. Такав приступ заснива се на семантичким описима регистрованих сервиса кроз одговарајуће онтологије.

4.1.14. Композиција сервиса

Једна од најсофистицираних функција које одређена архитектура може понудити корисницима је способност декларативног описа жељеног сервиса. Постоје два начина спецификације жељеног сервиса: дефинисањем комплетног тока података, почевши од основних извора сензорских података све до пролазака кроз жељене процесне блокове, или наводећи особине коначног тока података. Иако није обавезно, ослањање на семантичке технологије је најприкладнији начин за имплементацију ове функције. Типично, постоји посебна компонента која се назива планер која је одговорна за креирање тражене врсте сервиса.

4.1.15. Квалитет сервиса и информација

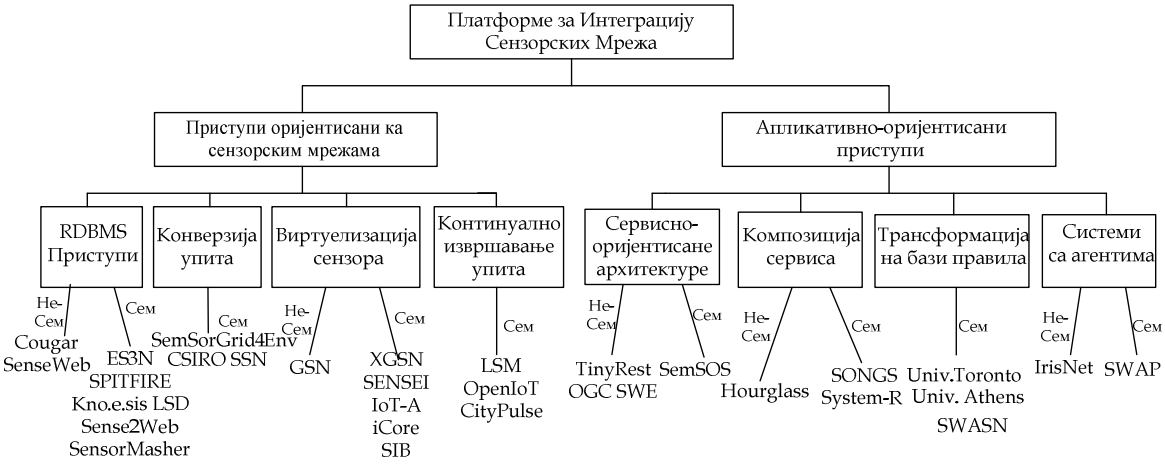
Претпоставља се квалитет сервиса и информација из перспективе апликација, што укључује следећа питања: (1) поверење, које се односи на поверење у вредности података и сродних мета-података, као и на поузданост карактеристика процесних елемената, (2) тачност, која зависи од обезбеђеног модела сензорских техничких могућности, (3) расположивост, управљану карактеристикама сензорских мрежа и ограничењима као што су радни режим, топологија мреже и коришћени протокол, и (4) испорука података, што се односи на кашњење података и инертност у окружењу са непоузданим везама итд.

4.2. Класификација постојећих решења

Главни проблеми у креирању нове таксономије су: критеријум класификације и стабло класификације. Овде су критеријуми за класификацију изабрани тако да одражавају суштину истраживачког основног становишта. Стабло

класификације је добијено применом изабраног критеријума. Листови стабла класификације су примери (истраживачки напори), који су укратко предочени у презентацији постојећих решења.

У овом прегледу постојећих решења, обухваћене су и архитектуре које не користе семантику података, ако је таква архитектура утицала на истраживање у одређеном правцу, што је касније резултирало применом семантичког приступа. На другој страни, у неким приступима је само додавањем семантичке анотације података постигнуто побољшање ефикасности коришћеног приступа. Платформе су класификоване по кључној идеји која их карактерише, што никако не значи да друге платформе не садрже исте или сличне блокове, али код којих ипак доминира други пројектантски приступ односно принцип.



Слика 14 - Класификација приступа за интеграцију сензорских мрежа

Пошто је улога платформе за интеграцију сензорских мрежа да делује као интерфејс између хетерогених сензорских мрежа и корисничких апликација, истраживачи су у могућности да приступе проблему кренувши од нивоа сензорских мрежа, тј. *приступ одоздо према горе*, или од нивоа корисничких апликација, тј. *приступ одозго према доле*. Зато као главни критеријум класификације, анализиране архитектуре се одређују према изабраном приступу који може укључивати: *приступ оријентисан ка сензорским мрежама* и *приступ заснован на апликацијама*. Класификационо стабло је приказано на

слици 14**.** У листовима стабла се налазе конкретне архитектуре и платформе које су класификоване у одговарајуће класе решења.

У првом приступу, истраживачи покушавају да предложе оптималан начин руковања, представљања, складиштења и сажимања расположивих извора сензорских података у циљу ефикасног решавања проблема хетерогености сензорских мрежа, разноликих техничких карактеристика, њихових ограничења, протокола и генерисаних сензорска опажања и мерења, како би се пружио једноставан и ефикасан приступ хетерогеним сензорским мрежама од стране горњих слојева у архитектури система, а тиме и апликацијама. У другом приступу истраживачки фокус је да се омогући кроз што једноставнији интерфејс и механизам интеракције са корисницима односно апликацијама, добијање жељених информација из интегрисаних сензорских мрежа, ослобађајући кориснике знања о специфичности и комплексности тих сензорских мрежа. У наставку је дат опис сваког од идентификованих приступа.

4.3. Приступи оријентисани ка сензорским мрежама

У оквиру приступа оријентисаних ка сензорским мрежама, можемо идентификовати четири подгрупе: *приступи засновани на базама података*, *приступи засновани на конверзији упита*, *приступ заснован на виртуелизацији сензора и приступи са континуалним извршавањем упита*. Све ове подгрупе се даље могу поделити на приступе са и без коришћења семантике података.

4.3.1. Приступи засновани на базама података

Ова категорија решења се карактерише базом података као централним чвориштем свих сакупљених сензорских података, а самим тим и све претраге и манипулације сензорским подацима се врше преко базе података. Испорука података корисницима се врши по моделу захтев-одговор, односно повлачењем (енг. *pull-based*), као резултат извшења *SQL* или *SPARQL* упита.

Пројектантски је изазов да се мапирају хетерогени неструктурирани сензорски подаци у релациони модел базе података. Међутим, са применом семантичких технологија, архитектуре се заправо заснивају на складишту *RDF* података (енг. *RDF Triple Store*), чија имплементација се такође може ослањати на *RDBMS*. Подаци се складиште представљени у *RDF* серијализованом формату, остварујући интероперабилност кроз дефиниције концепата из сензорског домена у онтологијама, у складу са принципом повезаних сензорских података (енг. *Linked Sensor Data*) што омогућује повезивање са другим семантичким скуповима података. Главни недостатак овог приступа је скалабилност, с обзиром да сервери база података и коришћена *RDF* складишта података нису изразито дистрибуирани системи, а сензорска опажања су веома динамичне природе са интензивним генерисањем нових података, што уз паралелно извршавање корисничких упита, постаје тежак задатак коришћених централизованих система за складиштење података. Међутим, овај приступ може бити нарочито користан због могућности примене алгоритама проналажења скривеног знања (енг. *data mining*) и машинског учења над складиштеним сензорским подацима из прошлости.

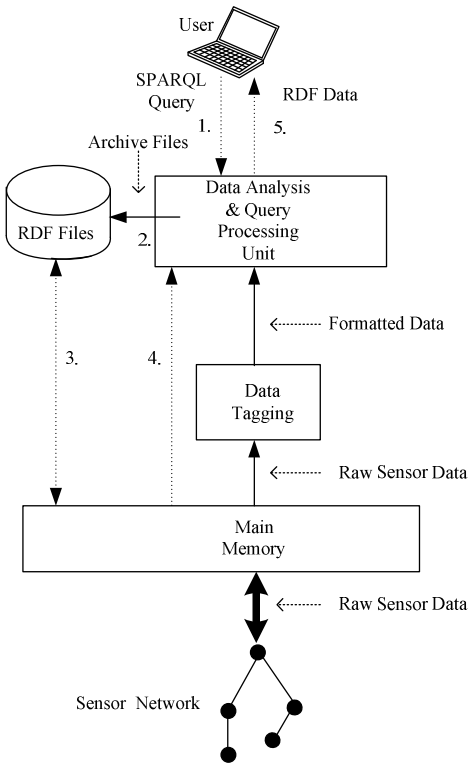
Конкретна решења базирана на бази података укључују не-семантичке приступе као што је *Cougar* [93] и *SenseWeb* компаније Microsoft [94], који представља пример максималне ефикасности описаног приступа. *Cougar* је један од првих истраживачких радова према интеграцији сензорских мрежа и користи објектно-релациону базу података са посебним апстрактним типовима објеката који одговарају одређеним сензорима, а могу имати разне методе за читање вредности сензора или детекцију вредности у односу на критичне. Тим методама се може приступити (односно захтевати њихово извршавање) SQL упитима, ако су испуњени други услови (типа сензор се налази на жељеној локацији) односно у дефинисаним временским интервалима. *SenseWeb* [94] користи *SenseDB* базу података за кеширање прикупљених сензорских мерења. Архитектура омогућава корисницима да захтевају одређене сензорске податке, а најпре се проверава да ли они постоје у бази података, како би се смањио приступ ка сензорским пролазним

уређајима (енг. *gateway*). Коришћењем посебне индексне структуре *COLR-Tree* (*Collection R-Tree*) омогућава се ефикасна просторно-временска претрага сензорских података и њихова агрегација како би се релевантна сензорска мерења испоручила корисницима. Архитектура нуди подразумевану имплементацију драјвера на пролазним уређајима, како би се омогућио униформни приступ хетерогеним сензорским мрежама. Постоје и компоненте које врше трансформацију (конверзију јединица мере и слично), процесирање података и њихову фузију, у циљу реализовања широког скупа апликација. *SenseWeb* представља веома ефикасно и стабилно решење са добрим перформансама, иако се користи углавном једноставна обрада сензорских опажања без рецимо детекције догађаја вишег нивоа.

Семантички базирана решења почињу са *ES3N (Exploiting Semantics in Sensor Networks for Silos*) [95] који је један од првих примера семантички базираног приступа. Касније долази до интензивног коришћења овог приступа у решењима која реализују принципе Повезаних сензорских података (енг. *Linked Sensor Data*) представљањем сензора и њихових опажања преко *URI* референци. Решења као што су *Kno.e.sis Linked Sensor Data* [47], *Sense2Web* [48], *SPITFIRE* [96] и *SensorMasher* [97] деле веома сличну архитектуру која се заснива на популарним *RDF* складиштима и то: *Kno.e.sis* се ослања на *Virtuoso*, *Sense2Web* и *SPITFIRE* користе *Jena* API које нуди избор неколико компатибилних RDF складишта (*Sense2Web* у позадини користи *SDB*[34] складиште које се ослања на *RDBMS*). Међутим, динамика сензорских података који се обрађују са овим архитектурама је знатно другачија. *Kno.e.sis* узима описе сензора и сензорска опажања из прошлости за време трајања неколико познатих урагана са око 20000 метеоролошких станица у САД, претвара измерене вредности у *RDF* формат уз додавање просторних координата као и веза ка местима дефинисаних у другим онтологијама (*GeoNames*[35]), уз анотацију временских тренутака када су извршена опажања, тако да корисници могу да шаљу просторно-временске-тематске упите.

[34] https://jena.apache.org/documentation/sdb/
[35] http://www.geonames.org/ontology/

Платформа *Sense2Web* обрађује само описе сензора без сензорских опажања, са додатим тематским мета-подацима (тип сензора и његове особине) и просторним подацима, који се могу односити на физичке локације као што су соба, зграда, спрат, пролаз итд, или могу да такође реферишу на локације дефинисане у другим семантичким скуповима података, на пример појам место из *DBPedia*[36] скупа. Обе платформе користе *Extensible Stylesheet Language Transformations (XSLT)* шаблоне за претварање података из *XML* формата у *RDF*. *SPITFIRE* платформа користи претраживаче (енг. *crawler*) како би се динамички прикупљали подаци из сензорских мрежа преко *REST* интерфејса или се одређени сензорски подаци добијају парсирањем одређених објава на вебу. Подаци се конвертују у *RDF* формат уз евентуално генерисање вредности виртуелних сензора, који се односе на неке појмове и стања из реалног света (нпр. паркинг место је слободно или заузето) и такви подаци се смештају у *RDF* складиште које униформно обрађује ове податке извршавањем корисничких *SPARQL* упита.



Слика 15 - Архитектура E3SN решења заснованог на RDBMS (адаптирана слика из [95])

[36] http://wiki.dbpedia.org/

Решење *ES3N* [95] је настало 2006. године на Универзитету у Џорџији и један је од пионирских приступа интеграције сензорских мрежа са применом у агрикултури. Решење се заснива на *RDF* складишту *Jena*[37] које чува *RDF* податке у табелама *RDBMS*, а аутори су развили сопствену онтологију са сензорима и концептима из апликативног домена. У складу са архитектуром приказаном на слици 15, прикупљени основни сензорски подаци се тагују са временским маркама и као дневне *RDF* колекције се смештају у *RDBMS*, али такође задржавају и у главној меморији, над којим *Jena* извршава корисничке *SPARQL* упите. Ако кориснички упити (фаза 1 са слике 15) захтевају сензорске податке из прошлости (фаза 2), *RDF* подаци се копирају у главну меморију (фаза 3), а *SPARQL* упит се извршава над подацима у меморији (фаза 4) и кориснику се враћа резултат (фаза 5). Корисници могу послати три врсте упита: истраживачке, који захтевају податке настале одређеног датума и времена; мониторске, с циљем дохватања свих *RDF* података одређеног датума; опсег упите, са циљем дохватања сензорских података насталих у одређеном временском интервалу али чији атрибути задовољавају одређене услове.

Као једно од пинорских прототипских подухвата, очигледно да *ES3N* има низ недостатака које потичу од ограничене скалабилности *RDF* складишта због централизованог приступа бази података, затим има сиромашан информациони модел, ограничену флексибилност и проширљивост (немогућност испоруке тока пристиглих података). Међутим, ово решење је извршило утицај на развој других решења у будућности.

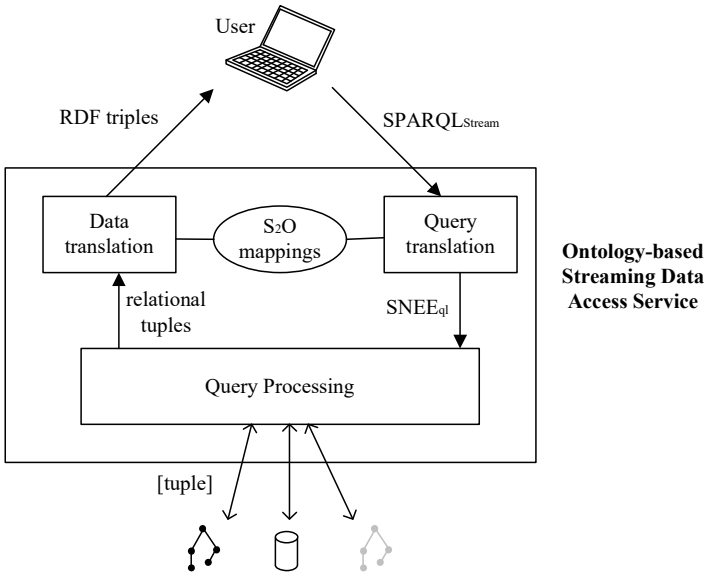
4.3.2. Приступ конверзије упита

Приступ конверзије упита користи природни формат сензорских података и одговарајући упитни језик како би се кориснички упити трансформисали ка циљном упитном језику одређеног сензорског извора података. Овај приступ подразумева потребу за одржавањем информација о доступним изворима података, првенствено изворног упитног језика одређеног извора података,

[37] https://jena.apache.org/

затим формата и природе генерисаних података, али такође могу бити укључене информације о техничким могућностима сензора, топологији мреже, ограничењима извора напајања ради боље оптимизације упита. Резултат изворних упита треба бити конвертован у циљни формат података који се испоручује корисничким апликацијама. Потенцијални недостатак перформанси лежи у чињеници да се за сваки захтев корисника морају обавити две конверзије: конверзија корисничког упита у изворни упит извора сензорских података и претварање резултата упита у циљни семантички формат порука ка кориснику.

Сва решења у класи приступа са конверзијом упита користе семантичке технологије. Прво решење, *CSIRO* семантичка сензорска мрежа [98], конвертује упите у компоненти званој сервисни посредник консултујући политику потрошње енергије и управљања извором напајања унутар *WSN*, на језик средњег слоја *Snlog* [98] који обухвата техничке спецификације сензорске мреже и сензора. Решење засновано на $SPARQL_{STREAM}$ упитима [99] је део семантичког интегратора унутар шире архитектуре креиране у EU FP7 пројекту *SemSorGrid4Env* [100], што је најсвеобухватније решење у овој групи.



Слика 16 – Архитектура система са конверзијом $SPARQL_{STREAM}$ упита на природне упите извора сензорских података (адаптирана слика из [99])

Архитектура која прихвата $SPARQL_{STREAM}$ упите је приказана на слици 16. Корисници задају упите у проширењу стандардног *SPARQL* језика са

конструктима за семантичке токове података. Ови упити се преводе у језик *SNEEql* [101] који је проширење *SQL* упитног језика и може да комбинује токове података са складиштеним подацима у релационим базама података. Превођење се врши на основу мапирања садржаног у језику *S_2O (Stream-to-Ontology)* који мапира концепте и термине из онтологије у термине токова података. *S_2O* је изведен из *R_2O (Relational-to-Ontology)* језика за мапирање, који дефинише односе између релационих табела и класа у онтологијама и сродних атрибута у табелама са својствима инстанци класа из онтологија. Такође више табела се може пресликати у једну класу из онтологије, један ред из релационе табеле у једну инстанцу класе итд. Упити токова података *SNEEql* се извршавају у блоку обраде упита који излаже своју функционалност као *SNEE-WS* сервис и даље усмерава упите ка сензорским мрежама или релационој бази података. Релациони токови података се опет конвертују у семантичке на основу *S_2O* мапирања. Јасно је да се на више места у архитектури врши конверзија и одлучивање о прослеђивању упита, као и трансформација података, што може деградирати перформансе.

4.3.3. Приступ виртуелизације сензора

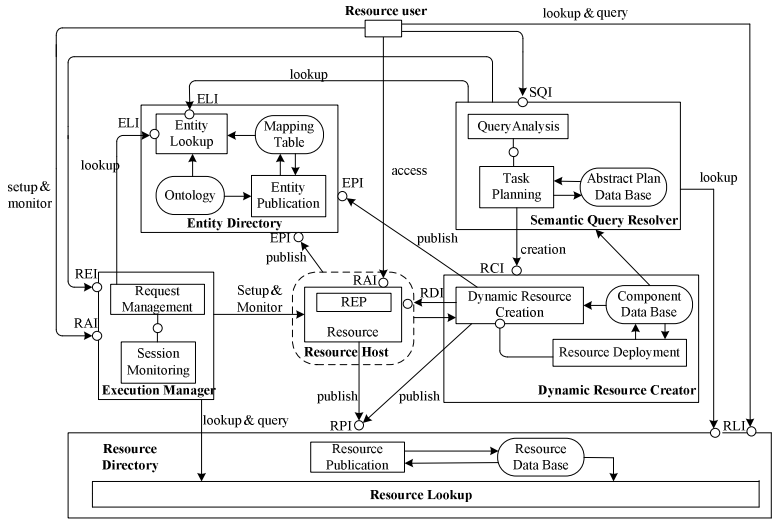
Код овог приступа, сензори и други уређаји су представљени апстрактним моделом података и апликацијама је обезбеђена могућност директне интеракције са релевантним уређајима преко дефинисаних интерфејса користећи прихваћени апстрактни модел. Без обзира да ли је имплементација дефинисаног интерфејса остварена на сензорским базним станицама или на пролазним уређајима, генерисани сензорски токови података морају да буду у складу са прихваћеним виртуелним моделом који треба да омогући интероперабилност. Уопштено говорећи, било који заједнички формат података, заснован на семантичком моделу података може се користити за представљање података. Може чак постојати и вишеструки модел података у распону од простог бинарног, преко текстуалног *XML* формата, до семантичког *RDF* модела, који паралелно коегзистирају и циљају на различите нивое апстракције података, испуњавајући различите потребе корисника. Реализована решења типично нуде добру скалабилност јер су кориснички

захтеви дистрибуирани директно ка извору сензорских података, а такође и високе перформансе, ефикасну фузију података из хетерогених сензорских мрежа и флексибилност у погледу сажимања токова података итд.

Несемантички приступ се користи у решењу *GSN* [102], које заправо припада класи решења средњег слоја. *GSN* уводи апстракцију *виртуелни сензор* који представља ток сензорских података или фузију више таквих токова, а процесирање над њима се описује *SQL* језиком. Провајдери сензорских података морају да се преко својих омотача повежу на *GSN* и тако испоруче своје податке. Семантичка надградња *GSN* је у решењу *XGSN* [103].

Архитектуре са овим приступом и са коришћењем семантичких технологија предложене су у великим *EU* пројектима серије *FP7* који се надовезују један на други и то *SENSEI* [104, 105] и *IoT-A (Internet of Things Architecture)* [106], а такође и у пројектима SIB [107] и *iCore* [108]. *SENSEI* је био највећи *EU FP7* пројекат у позиву 1, са веома амбициозним циљевима креирања скалабилне архитектуре која би обезбедила лаку интеграцију сензорских и актуаторских мрежа (енг. *WSAN*) и тиме омогућила тзв. хоризонтално искоришћавање сензора, актуатора и процесних компонената у апликацијама из разних бизнис домена и промоцију отвореног тржишта понуђача и потрошача информација из реалног света и/или актуатора. У *SENSEI* архитектури (слика 17) апликације добијају униформни приступ информацијама из реалног света (преко сензора и контекстних сервиса) и могућност интеракције (преко актуатора и контролних сервиса) кроз *слој ресурса (*енг. *Resource Layer)* који се опет ослања на *Комуникационо-сервисни слој*. Централни концепт у архитектури је *ресурс (енг. Resource)* и он представља апстракцију физичких уређаја као што је сензор, актуатор, или њихове комбинације, односно процесора или софтверске компоненте, који су виртуелни ресурси. *Ресурсу* се приступа преко *интерфејса ресурса* (енг. *Resource Access Interfaces - RAI)*, а захтеве сервисира уређај *домаћин ресурса* (енг. *Resource Host)*. *Домаћин ресурса* региструје припадајући *ресурс* слањем *описа ресурса* (енг. *Resource Description)* ка централној компоненти званој *директоријум ресурса* (енг. *Resource Directory)*. *Опис ресурса*

садржи тип информација који пружа *ресурс*, његов приступни интерфејс и методе, које задатке извршава итд. Веза између виртуелног и реалног света се остварује кроз везу *ресурса* и *ентитета од интереса* (енг. *Entity of Interest)*, апстракције која представља људе, објекте, места, а те везе се одржавају у *директоријуму ентитета* (енг. *Entity Directory)*.



Слика 17 – Архитетктура пројекта SENSEI са виртуелизацијом сензора кроз апстракцију ресурс (адаптирана слика из [104])

Постоји неколико предвиђених модела интеракције: у најпростијем моделу, *корисник* (енг. *Resource User)* тражи одређени *ресурс* од компоненте *директоријум ресурса* кроз рандеву механизам и директно му приступа на основу описа *интерфејса RAI* добијеног у *опису ресурса.* У другом моделу, корисник шаље декларативни захтев компоненти *решавач семантичких упита* (енг. *Semantic Query Resolver - SQR)* специфицирајући контекстни или сензорски податак који му је потребан односно актуаторски задатак који треба да се изврши. *Решавач семантичких упита* затим анализира захтев и проналази ресурсе који се могу користити за задовољење захтева укључујући и квалитет информација или квалитет актуације, а ако је потребно може иницирати креирање динамичких ресурса код компоненте *динамички креатор ресурса* (енг. *Dynamic Resource Creator)*. Алтернативно, компонента *извршни менаџер* (енг. *Execution Manager)* може извршити захтев у име корисника. У једноставном сценарију, он извршава директно захтев и враћа информације. У дугорочном сценарију претплате, *извршни менаџер* успоставља сесију између корисника и *ресурса* према плану извршења.

Информациони модел [105] садржи три нивоа апстракције. *Ниво сирових података* обухвата сензорске податке којима су евентуално додати основни мета-подаци попут квалитета информација, јединица мере и слично. *O&M ниво* је сличан *OGC SWE* [33] али је базиран на *SENSEI SensorData* онтологији која омогућава додавање произвољних мета-података. И последњи ниво је *Контекстно-информациони слој,* који моделира контекстуалне податке високог нивоа о ентитетима од интереса и њихова својства. Корисник у својим захтевима може да специфицира који ниво информација захтева.

Мане једне овако комплексне архитектуре је недостатак подршке за управљање историјским сензорским подацима, регулисање аквизиције за токове података у реалном времену, фузија података у реалном времену са складиштеним подацима и подршка за *RFID* уређаје. Још једна пожељна, а недостајућа карактеристика, је могућност претраживања великих количина сирових сензорских и контекстуалних података користећи временске, просторне и друге атрибуте.

У пројекту *IoT-A* [106] неки од ових недостатака су исправљени, али само на нивоу концептуалног модела. Пре свега, информациони модел је доста богатији и укључује: *модел ентитета* (енг. *Entity model)* са виртуелним ентитетом као кључном апстракцијом са додатим временским, просторним и другим доменским атрибутима; затим *модел ресурса* (енг. *Resource model)* који такође има *ресурс* као кључну апстракцију за сензорске уређаје, овога пута са коришћењем екстерне стандардизоване *SSN (Semantic Sensor Network*) онтологије [12]; и на крају *сервисни модел* (енг. *Service model)* који садржи сервисне профиле са линковима између *виртуелних ентитета* и *ресурса* који нуде сервисе о повезаном ентитету. Предвиђена је подршка за смештање података из прошлости о виртуелним ентитетима и ресурсима, у одговарајућим слојевима архитектуре. Међутим, *IoT-A* пројекат је предложио само апстрактну референтну архитектуру, односно модел који би требало да

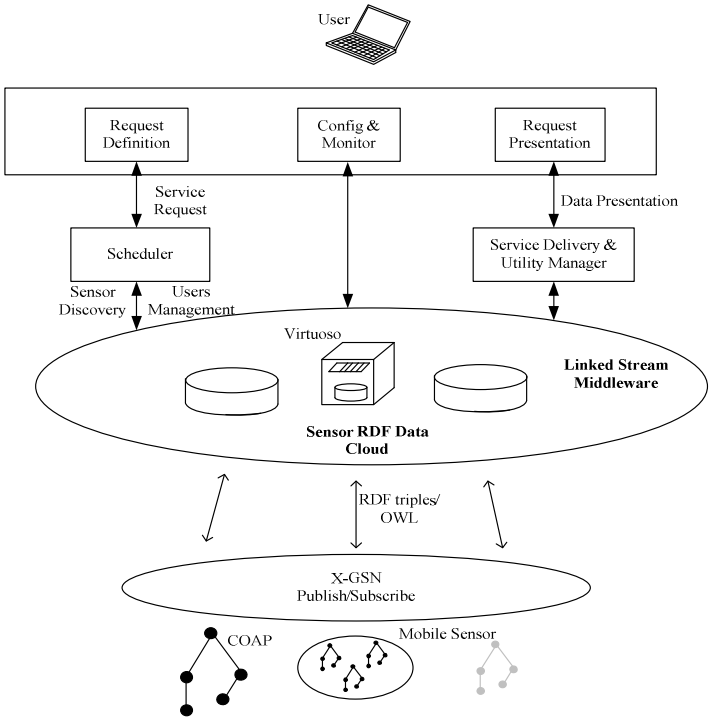
прате конкретне имплементације, тако да нема резултата тестирања из реалног окружења.

Пример конкретног пројекта који прати *IoT-A* референтну архитектуру је дат у [107], у којој се семантички сензорски подаци објављују од стране виртуелних ентитета преко семантичких брокера као скуп *RDF* тројки, односно као скуп *SPARQL* упита који врше унос нових сензорских опажања или ажурирање стања. Слично у пројекту *iCore* [108] креиран је програмски овир са фокусом примене на *IoT* апликације за паметне градове, уводи се апстракција виртуелни објекат и композиција виртуелних објеката у циљу сакривања особености хетерогених сензорских мрежа.

4.3.4. Приступ са континуалним извршавањем упита

Овај приступ за испоруку сензорских података користи систем за непрекидно извршавање упита у циљу филтрирања најновијих сензорских опажања и њиховог прослеђивања заинтересованим корисницима у складу са њиховим упитима. Проширење система може ићи у правцу креирања агрегираних токова сензорских података, када се корисницима испоручују или релевантне сензорске вредности у неком интервалу или неке статистичке вредности, затим креирање токова вишег нивоа, односно резултат детекције догађаја вишег нивоа обрађивањем основних сензорских опажања. Код хибридног приступа се комбинују решења за непрекидно извршавање упита над токовима сензорских података и решења за складиштење сензорских *RDF* података која чувају или статичке сензорске описе или сензорска опажања из прошлости и извршавају *SPARQL* упите над тим подацима. Пројектантски је изазов да се оствари филтрирање токова података на основу статичких семантичких сензорских података или да се ефикасно комбинују најсвежији и сензорски подаци из прошлости, пошто су у питању два одвојена решења. Решење које користи овај приступ је *Linked Sensor Middleware (LSM)* [109] коришћењем имплемемтације *CQELS* упитног језика [110], а такође је део веће платформе у облаку *OpenIoT* [111], а исто тако систем са обрадом *CQELS* упита је језгро *EU* пројекта *CityPulse* [112].

Решење из референце [113] обрађује само токове сензорских података и остварује проток од неколико десетина хиљада *RDF* тројки у секунди обрађивањем узорака упита коришћењем *CTP-automata (Conjunctive Triple Pattern automata)*. На основу задатих корисничких узорака упита, креирају се коначни аутомати стања за сваки узорак тројки које се сажимају у аутомат стања спајањем еквивалентних чворова односно стања. По доласку нових *RDF* тројки, упаривањем вредности са стањима у аутомату стања, пролази се кроз задати аутомат стања и ако се стигне у крајње стање које кореспондира неком упиту, та *RDF* тројка задовољава упит и прослеђује се кориснику.



Слика 18 – OpenIoT Архитектура (адаптирана слика из [111])

Организација *OpenIoT* архитектуре је приказана на слици 18. Сензори се интегришу у систем преко решења средњег слоја *X-GSN* методом виртуелизације сензора уз семантичку анотацију описа и вредности мерења уз могућност прослеђивања података преко објави/претплати механизма. Семантички сензорски токови података се складиште у облаку на платформи *LSM,* која интерно користи *RDF* складиште *Virtuoso*[38] и опслужује *SPARQL* корисничке упите. За представљање семантичких сензорских података,

[38] https://virtuoso.openlinksw.com/

платформа користи *SSN* онтологију, али такође има и сопствену онтологију ради лакшег интегрисања сензора. Корисници су у могућности да затраже податке преко посебне компоненте за презентацију, а такође да користе сервис откривања сензора и да захтевају одређени сервис преко компоненте за дефинисање захтева (енг. *Request Definition*). Компонента *Scheduler* покушава да пронађе адекватне сензоре и токове сензорских података како би испунила захтеве корисника, а уједно управља правима приступа. Основни проблем ове архитектуре је скалабилност, односно у којој мери *RDF* складиште *Virtuoso* може да ефикасно складишти токове сензорских података и да истовремено извршава корисничке семантичке упите. Предности платформе су напредне функционалности остварене укључивањем компонената из других приступа попут виртуализације сензора и композиције сервиса.

4.4. Апликативно-оријентисани приступи

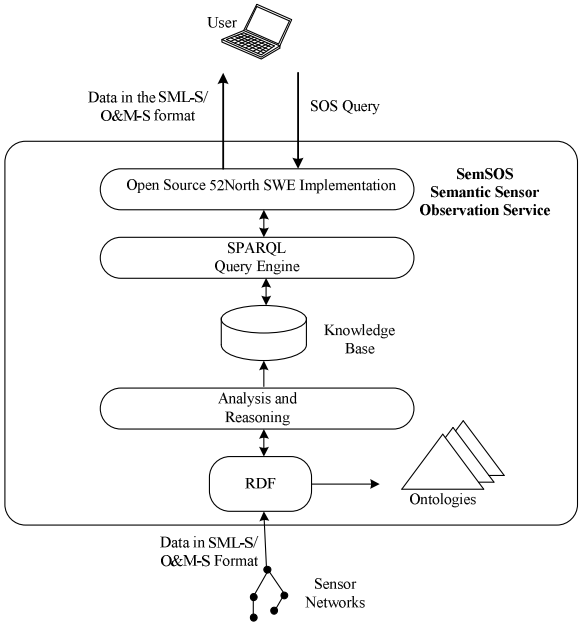
Као што је већ речено, ***апликативно-оријентисани приступи*** покушавају да обезбеде корисничким апликацијама најефикаснији начин за добијање потребних података из интегрисаних сензорских мрежа. Међутим, фокусирањем на интеракцију на високом нивоу између апликација и платформе за интеграцију, уз омогућавање комплексних функција попут извођења новог знања, некада доводи до смањења перформанси, што спречава шире прихватање ових решења. Могу се идентификовати четири подгрупе које деле исти основни принцип *приступа одозго према доле*: *приступ сервисно-оријентисане архитектуре*, *приступ композиције сервиса*, *приступ трансформације података заснован на правилима* и *приступ заснован на агентима*.

4.4.1. Приступ са сервисно-оријентисаном архитектуром

Приступ са сервисно-оријентисаном архитектуром обезбеђује стандардни сервисни интерфејс са дефинисаним методама и форматима порука за добијање сензорских опажања и мерења. Штавише, архитектура може понудити и друге функције као што су добијање информација о карактеристикама сензора, сервис за претплаћивање на изабрана сензорска опажања и мерења, извршавање упита, опционалне актуаторске функције

итд. Доминантна интеракција у овој архитектури је по моделу захтев-одговор односно повлачењем података, а у мањој мери могу се наћи модел испоруке података гурањем као приликом испоруке сензорских опажања вођених догађајима. Недостатак постојећих реализација је немогућност фузије најновијих токова сензорских података са подацима из прошлости. Иако нема експлицитних ограничења код конкретних имплементација, овај приступ има тенденцију да се користи код вертикално оријентисаних решења које покривају један апликативни домен. Сервисно-оријентисане архитектуре укључују једноставно и ефикасно несемантичко решење *TinyREST* [114] и имплементације *OGC SWE* референтне архитектуре [33] у пројекту *EU FP6 SANY* [115] под иницијативом Европске Свемирске агенције [116] и *The 52North SensorWeb Community* [117]. Семантички приступ је реализован у решењу *SemSOS* [118] додавањем семантичких анотација *OGC SWE* моделу података.

TinyREST [114] представља једноставну архитектуру у којој пролазни уређај прихвата *REST* захтеве корисника и то *GET* за добијање вредности неког сензора, *POST* за задавање акције актуатора и *SUBSCRIBE* за претплаћивање на нова сензорска мерења. Сензори се адресирају преко *URI* референце, а пролазни уређај усмерава поруке ка одговарајућим сензорским чворовима који покрећу *TinyOS* оперативни систем и извршавају добијене захтеве.



Слика 19 – Архитектура семантичког СОС сервиса – SemSOS
(адаптирана слика из [118])

SemSOS (Semantic Sensor Observation Service) [118] проширује *52North OGC SWE* имплементацију додавањем семантичких анотација у стандардни *OGC O&M* формат порука преко *XLink (XML Linking Language)* [119] технологије која омогућава креирање и опис линкова између ресурса описаних у онтологији, креирајући тако *O&M-XML* формат. Наиме, концепти из *OGC O&M* спецификације као што су сензорска опажања (енг. *Observations*), појаве из реалног света (енг. *Feature*) и друге, описани су у језику за опис онтологија *OWL*, јер *OWL* дозвољава креирање хијерархије концепата и ограничења између њихових веза. Таква представа је названа *O&M-OWL*.

Ово проширење садржи просторне концепте као што су тачка, полигон и координате, који су преузети из *GML (Geography Markup Language)* језика [120], временске концепте попут временског интервала преузетог из *OWL-Time* онтологије [121] и концепата из апликативно-доменске онтологије, као што су рецимо мећава (изведена од концепта појава из реалног света) и снежна олуја, преузетих из онтологије временских услова. Помоћу тог проширења, могуће је креирати просторно-временско-тематске семантичке упите. Архитектура са слике 19 предвиђа да се од улазних параметара *SOS* упита креира *SPARQL* упит којим се претражују *RDF* сензорски подаци представљени у *O&M-OWL* формату. Резултат упита се враћа кориснику након конверзије у *O&M-XML* формат, који поштује *OGC SWE* синтаксу, али такође садржи семантичке анотације за оне кориснике који су у могућности да их интерпретирају. Описано решење представља прототип елегантног проширења *OGC SWE* стандардне имплементације са циљем додавања семантике података и могућности креирања семантичких упита, без намере да се оствари скалабилна архитектура.

4.4.2. Приступ композиције сервиса

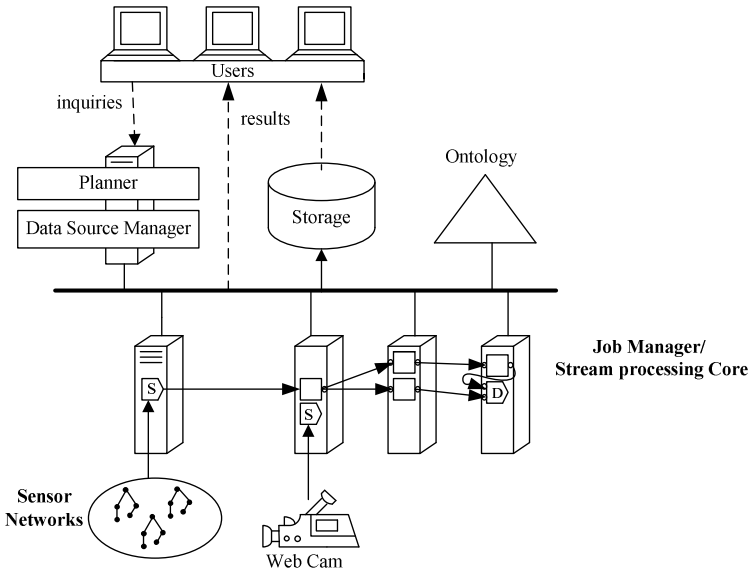
Приступ композиције сервиса омогућава корисницима да дефинишу произвољне сервисе или токове сензорских података са посебним карактеристикама од интереса. Систем ће покушати да састави такав проток

података применом низа специфичних процесирања над одговарајућим изворима сензорских података, што ће резултирати креирањем тока података који је у складу са траженом спецификацијом. Потпуна изражајност корисничких захтева може се постићи омогућавањем описа жељеног тока података и процесирања семантичким моделом. Тиме се остварује семантички базирано закључивање које се користи приликом тражења оптималне композиције од расположивих компонената. Овај приступ нуди најфлексибилније решење из перспеткиве апликација, иако перформансе могу деградирати због комплексног задатка проналажења одговарајућих сервисних компонената у реалном времену.

Пројекат *Hourglass* [122] реализован на Универзитету Харвард још 2004. године је пример несемантичког решења који користи овај приступ. У њему је кључни концепт *кружни пут* (енг. *Circuit)* који представља путању података између извора сензорских података и њихових потрошача и укључује више сервиса који извршавају операције над тим подацима. Сервиси могу бити генерички попут баферисања и филтрирања, или специфични за апликацију, а њихове особине су тема, тачка приступа итд. Компонента *Circuit Manager* креира *кружни пут* тока података на основу описног језика у *XML*, што укључује скуп оператора, тема и атрибута захтеваних сервиса, тако што проналази адекватне расположиве сервисе на основу информација о њима садржаних у регистру и повезујући их у ток података.

Знатно моћнија решења користе семантичке приступе и укључују архитектуру *SONGS* [123] развијену у компанији Microsoft и архитектуру развијену у компанији IBM [124] на бази *System S* решења средњег слоја. Код архитектуре *SONGS*, кључне компоненте су семантички сервиси који процесирају семантичке токове података. Планер сервиса чува опис сервиса као скуп логичких правила са ограничењима и имплементиран је као извршна јединица са логичким закључивањем. Корисници задају упите са жељеним семантичким токовима података, а планер сервиса разлаже упит на скуп сервиса који се додељују на извршавање мрежним чворовима.

Распоређивач сервиса на сваком чвору покушава да оптимизује извршавање задатака у оквиру сервиса описаних посебним језиком или да искористи већ постојеће сервисе. Повезивањем распоређених семантичких сервиса креира се жељени проток семантичких података.



Слика 20 – Архитектура за композицију семантичких сензорских токова података развијена у компанији IBM (адаптирана слика из [124])

На слици 20 је приказана архитектура развијена у компанији IBM. Корисници задају упите са семантичким описима жељених токова, коришћењем концепата из развијене онтологије која покрива типове сензора (камера, звучни сензор) и произведених података (видео сегмент, аудио сегмент), са временским и просторним информацијама. Извори података генеришу ток података (енг. *Stream Data Objects)* који се састоје од парова елемената података и њихових семантичких описа у *RDF* формату преко концепата из онтологије. Процесни елементи се описују узорцима тока, који дефинишу врсту елемената података и њихову семантику који су дозвољени на улазу и произведених токова на излазу. Процес могуће интерконекције токова података и процесних елемената се своди на уклапање узорка дозвољеног улазног тока процесног елемената са описом тока података. То је могуће ако се за сваку променљиву из узорка тока може пронаћи замена изведена из улазног тока, а то извођење се заснива на дефиницијама и везама из коришћене

онтологије. Компонента *планер* (енг. *Planner)* проналази граф процесирања који би произвео ток података у складу са задатим узорком тока, тако што комбинује скуп токова са компатибилним процесним елементима. То се ради у две фазе: у првој фази се генеришу чињенице о процесним елементима закључивањем на бази дескриптивног логичког програмирања користећи запис у језику за планирање процесирања тока (енг. *Stream Processing Planning Language - SPPL*), што се складишти и користи код више упита. У другој фази, кориснички упит се преводи у *SPPL* планирани циљ који треба да произведе компонента *решавач* (енг. *SPPL solver)* повезивањем одговарајућих компонената. Компонента *менаџер послова* (енг. *Job Manager)* распоређује резултантни план за извршавање на *систему за процесирању токова* (енг. *Stream Processing Core)*. Резултат експеримената је потврдио да просечно време проналаска задатог тока података износи 2,5 секунде, што ипак ограничава примену овог решења у сценаријима са већим бројем корисника. Скалабилност би могла бити побољшана ако би се *планер* реализовао као дистрибуирана компонента, са репликованим информацијама о карактеристикама процесних елемената.

4.4.3. Приступ трансформације података заснован на правилима

Ово је чест приступ у решењима са коришћењем семантичких технологија, или друга решења користе неке компоненте које су централне у овом приступу. Извођење новог знања или откривање догађаја вишег нивоа се постиже функцијама мапирања које се ослањају на везе између концепата обухваћених онтологијом домена и онтологијом сензорских опажања и сензорских мерења. Корисницима је дата могућност семантичког претраживања и примена алгоритама закључивања тако што се подаци трансформишу у семантички засновану репрезентацију из формата нижег нивоа. Може постојати више трансформација кроз архитектуру у складу са различитим слојевима у информационом моделу. Са овим приступом су реализована само решења која есплоатишу семантичке технологије и то: *SWASN (Semantic Web Architecture for Sensor Networks)* [125] платформа део *CommonSense* визије компаније *Ericsson*, проширљива архитектура реализована

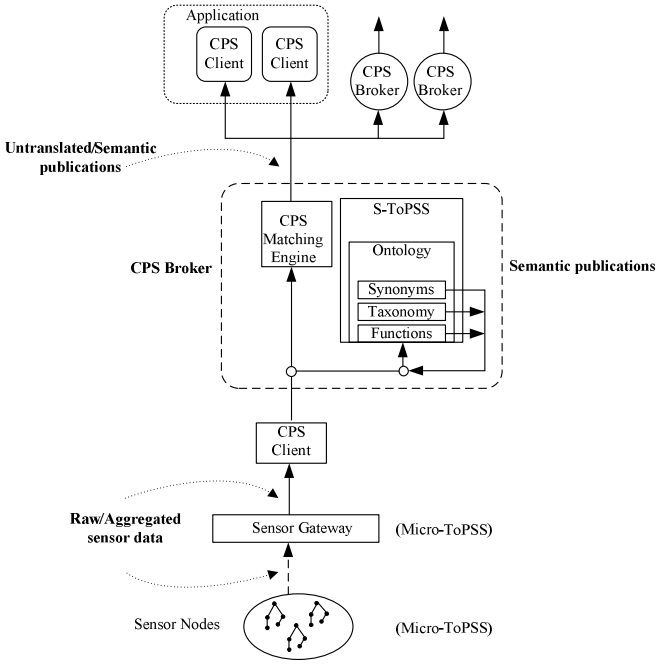
у *Националном техничком универзитету у Атини (НТУА)* [126] и семантички заснован систем за фузију сензорских података реализован на Универзитету у Торонту [127].

Архитектуре *SWASN* и *НТУА* су веома сличне, са вишеслојном организацијом и коришћењем система *Jena* за рад са семантичким подацима. Код *SWASN* је најнижи *слој извор сензорских података* и обухвата хетерогене сензорске мреже; следећи је *онтолошки слој* у коме се сензорски подаци претварају у семантичке уз могућност да за сваку сензорску мрежу постоји посебна онтологија; затим следи *семантички веб процесни слој* који се заснива на *Jena API* за манипулацију са *RDF* подацима и извршавање *SPARQL* упита, а извођење новог знања се изводи коришћењем *Jena* закључивача и јединице за генеричка правила или евентуално неког екстерног решења за закључивање; последњи *слој је апликативни* са клијентским апликацијама које приступају функционалностима платформе преко *HTTP* интерфејса.

НТУА модуларна архитектура [126] такође садржи најнижи *слој података* са хетерогеним сензорским уређајима, у коме се разликују класичне *WSN*, локацијски сензори и мултимедијални извори података. Изнад је *процесни слој* у коме се сензорски подаци трансформишу у *XML* податке, примарно поштујући *OGC SWE* спецификацију, а може се интегрисати и *GSN* [102] средњи слој. Последњи је *семантички слој* у коме се врши трансформација података на бази правила у формату "on *event* if *condition* then *action*", односно одређене акције се врше ако се десио неки догађај и испуњени су одређени услови. Постоје две врсте правила и то: *XML* правила мапирања, којима се *XML* поруке претварају у онтолошке инстанце, и семантичка правила, када се нове онтолошке инстанце изводе из постојећих. Користећи онтолошке инстанце, сервер за закључивање (такође *Jena* компатибилни) креира и управља базом знања над којом корисници могу да извршавају упите или да буду обавештени о детекцији догађаја вишег нивоа у складу са дефиницијама у онтологији. Нема података о реалним перформансама описаних

архитектура, али јасно је да ослањањем на централизовани систем *Jena* повлачи ограничења у погледу скалабилности и перформанси.

Архитектура са Универзитета у Торонту [127] приказана на слици 21 се заснива на систему за размену порука вођеног садржајима (енг. *Content-based Publish/Subscribe - CPS),* који се користи за интеракцију са сензорским мрежама и за остваривање семантичких функција над подацима. Функцију *CPS* сензорског система обавља *Micro-ToPSS* платформа средњег слоја инсталирана на сваком сензорском чвору и реализована као проширење *TinyScript* језика [128] и његове виртуелне машине. *Micro-ToPSS* платформи се могу слати захтеви за праћење одређених сензора. Прикупљени или агрегирани сензорски подаци пролазе кроз пролазни уређај и даље преко *CPS* клијената врши се препакивање у сензорска објављивања и шаљу се брокеру. Брокер порука покреће семантичко извршно окружење за размену порука *Semantic-Toronto Publish/Subscribe System* (*S-ToPSS*) и врши уклапање објава и претплата на основу концепата и њихових релација у доменској онтологији.



Слика 21 – Архитектура са семантичком фузијом података и преносом порука на бази садржаја (адаптирана слика из [127])

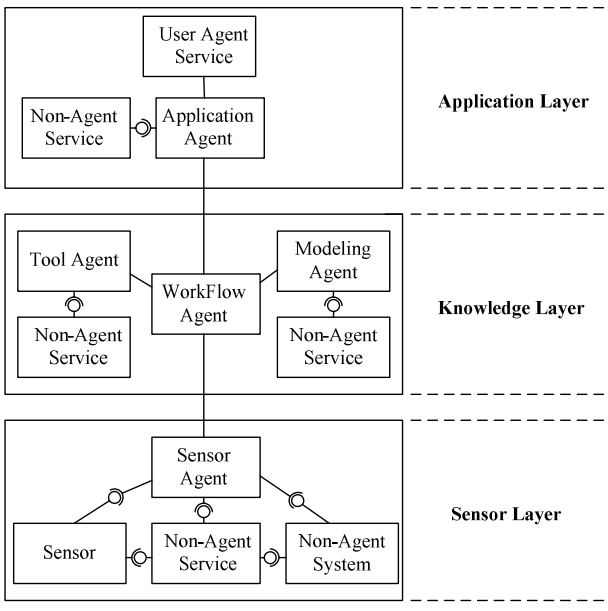
Постоје три технике уклапања. *Транслација синонима* се заснива на мапирању еквивалентних термина; *транслација таксономије* користи хијерахију релација

и користи се када објаве садрже генералније термине у односу на оне из претплата; трећа техника користи функције мапирања за транслирање атрибут-вредност торки сирових података у семантичке атрибут-вредност торке и ова техника може бити искоришћена за детекцију догађаја вишег нивоа. Апликације се могу претплатити на сирове и агрегатне податке, са додатом семантиком или не. Фузија података се остварује са могућношћу претплате апликација на корелацију између података из различитих сензорских мрежа. Мана архитектуре је подршка само за токове података вођених догађајима, без подршке за аквизиционе токове сензорских података као и дохватање сензорских података из прошлости. Фузијом ових података, архитектура би нудила могућност детекције догађаја вишег нивоа насталих у прошлости.

4.4.4. Приступ заснован на агентима

Овај приступ је карактеристичан по ангажовању неколико врста агената распоређених у више логичких слојева. Агенти су софтверске компоненте способне за обављање одређених задатака попут прикупљања сензорских података, трансформацију података, извршавање упита итд, и сарадњом са другим агентима остварују жељене функционалности. За интерну комуникацију између агената могу се користити неке од стандардних платформи за агенте или се реализује специфична имплементација. Типично, агенти припадају једном од неколико слојева у зависности од типа функционалности за који су одговорни. Може постојати неколико типова агената у једном логичком слоју. Агенти из горњег слоја запошљавају агенте из нижих слојева. У конкретној имплементацији може се десити да агенти користе семантичке сензорске податке и/или се семантички модели користе за опис процесних могућности агената. Постојећа решења обухватају несемантичку сензорску инфраструктуру Интернет размера названу *IrisNet* [129] и семантички засновани мулти-агентни систем за *Sensor Web* архитектуру *SWAP* [130].

IrisNet (Internet-scale Resource-Intensive Sensor Network Services) [129] је хијерахијска архитектура заснована на агентима који остварују сензорске сервисе. Један сензорски сервис се везује за један дистрибуирани *XML* документ који садржи сензорске податке партиционисане на више рачунарских чворова. Постоје *сензорски агенти* који прикупљају сензорска опажања и обрађене их шаљу *организационим агентима* који се налазе на чворовима и задужени су за извршавање упита над *XML* документом. Упити се изражавају у *XPATH* декларативном упитном језику и извршавају се тако што *организациони агенти* врше упит најпре над локалном базом података и кешираним подацима на свом чвору и ако пронађу да недостају неки подаци, шаљу подупит *организационим агентима* на другим чворовима и тако рекурзивно даље ка другим чворовима ако је потребно. Резултат свих подупита се саставља у коначни резултат упита који се шаље кориснику. Велики број дистрибуираних *сензорских* и *организационих агената* омогућавају веома скалабилну архитектуру иако постоји низ ограничења у погледу подржане модалности података.



Слика 22 – Архитектура са агентима и семантичким приступом (адаптирана слика из [130])

Архитектура семантичког приступа са агентима у пројекту *SWAP (Sensor Web Agent Platform)* [130] приказана је на слици 22 са агентима распоређеним у три одвојена слоја. У *Сензорском слоју, сензорски агенти* су задужени за интеракцију

са сензорима и актуаторима или директно или преко неког од *OGC SWE* сервиса (*SOS, SAS, SPS*). Сензорски агенти размењују поруке структуриране у складу са спецификацијом из онтологије, што укључује мета-податке о појави која је опажана, јединици мере, временским атрибутима итд. Експертско знање у систему је садржано у *Слоју знања* (енг. *Knowledge Layer*) у којем постоје три врсте агената: *агенти радних процеса* (енг. *workflow agents)* су кључне компоненте система, затим *агенти за алате* (енг. *tools agents)* који извршавају предефинисане процесе без потребе за другим ресурсима и *агенти за моделовање* (енг. *modeling agents)* који извршавају комплексно процесирање, а могу захтевати додатне податке. Типичан сценарио извршавања почиње од *агената радних процеса* који прихватају податке од *сензорских агената* и позивају одговарајуће *агенте за алате* и *агенте за моделовање* способне да изврше потребне операције на датим подацима. Обрађени подаци се могу даље прослеђивати другим *агентима радних процеса* или бити прослеђени ка *Апликативном слоју,* ако је обрада завршена. Сва обрада у овом слоју и интерпретација знања је базирана на вишеструким онтологијама, а за опис процеса и задатака коришћен је *OWL-S* [131], проширење OWL језика за моделовање процеса. У *Апликативном слоју* постоје две врсте агената и то *апликативни* и *кориснички*. *Апликативни агенти* се користе за састављање специфичних апликација на основу излазних података из *агената радних процеса*, док *кориснички агенти* омогућавају избор расположивих апликација крајњим корисницима и обавештавају их о појави нових или промени постојећих апликација. Нема информација о подржаним корисничким упитима за сензорским подацима, подршком за подацима из прошлости, токовима података, иако концепт архитектуре обећава добру скалабилност.

4.5. Компаративна анализа постојећих решења

У овом одељку биће дата компаративна анализа идентификованих архитектура за семантичку интеграцију сензорских мрежа и упоредни приказ особина појединих архитектура за пројектантске параметре наведене у одељку 4.1.

Табела 8 – Упоредни приказ одабраних сензорски-оријентисаних архитектура

Solution	Basic Organiz-ation	Scalab-ility	Data Modality support	SN Flexi-bility	SN capabilit y	SN mana-gement & actuation	Ontolog y	Employ-ed Semant-ics	Data Represen tation	Query language	Knowle-dge inference	App. interface and data format	Discover y of service	Service Compos-ition	Security
Sense Web [94]	Three logical layers	Medium	Stream data, Archived, mobile	High, Via drivers or DataHub	Particularl y	No	-	-	Relational data format	SQL	No	Web Service	Yes, transforme rs discovery	No	Medium
The ES3N [95]	Multi-layered	Low	Low level. Archived and acquisition -al data	?	No	No	Own implement ation	Data, queries	RDF	SPARQL	Via queries	?	No	No	No
Kno.e. sis LSD[47]	Vertical service-oriented	Low	Low, Static Linked Sensor Data	-	No	No	O&M-OWL	Sensors and observatio ons	RDF	SPARQL	Yes	HTTP, ODBC	No	No	No
SPIT-FIRE [96]	Centralize d RDF Triple Store	Distribute d sensor Crawlers	Pull-based sensor data collection	Medium – Through RESTfull interface	Through sensor description	No	Extended SSN amd Dolce	Semantic entities description	RDF	SPARQL	No	RESTfull	Search of semantic entities	No	No
SemSor Grid4 Env [100]	Three-layers service-oriented	Medium to High	High level, streams, feeds, stored data	Medium-high, Over natural query languages	Particularl y, by describing service providers	No	Four levels, SWEET & DOLCE, SSN, domain	Queries, Data sources description , sensor data	Native data format	Ontology based	Via queries	REST, HTML, GML, GeoJSON	Yes, via stSPARQL	No	?
GSN [102]	Peer-to-peer	High	Medium level, Streams, stored data	Medium, Through Wrappers	No	No	-	-	XML	SQL	No	XML	No	Low, creation of complex streams	?
SENSEI [104]	Directory based	High	High level, no support for archived data	High	?	Yes, arbitration is provided	SWEET. SemsorOnt ology, Context Ontology	Sensor & entity data, resource semantic look-up	O&M XML and RDF in parallel	Semantic queries for resources	?	REST, three data level	Yes, lookup of entities, resources	Medium, dynamic resource creation	High, AAA Architectu re
SIB IoT-A [107]	Semantic Broker	High	N/A	Ubiquitous ID (uID) resolution architectur e	N/A	N/A	N/A	Semantic brokers	RDF	SPARQL 1.1	No	HTTP	Semantic Informatio n Broker Discovery	No	?
Open IoT [111]	Sensor Data Cloud	Medium	High level, semantic streaming data	High, using X-GSN	Supported by the Informatio n model	-	SSN, OpenIoT	Service description sensors, data	RDF	SPARQL	-	HTTP, JDBC	Yes	Yes	Users' mngm.
City Pulse [112]	Based on Data Bus	High	Sensor streams,	Sensor Virtualizat ion	Through Data Wrappers	No	4 main modules, Stream Annotattio n, Quality	Sensor data streams, Events	RDF	CSPARQL	Decision support module	REST	Semantic event discovery	Event service compositio n	?

Табела 9- Упоредни приказ апликативно-оријентисаних архитектура

Solution	Basic Organiz-ation	Scalab-ility	Data Modality support	SN Flexi-bility	SN capabilit y	SN mana-gement & actuation	Ontolog y	Employ-ed Semant-ics	Data Represen tation	Query language	Knowle-dge inference	App. interface and data format	Discover y of service	Service Compos-ition	QoS & QoI	Security
Tiny REST [114]	Client-Server	Low	Low	Via Device Support Protocol	Network deploymen t	Yes, actuation	-	-	?	-	-	REST + HTML	Home Service Framewor k	No	No	No
OGC SWE [33][34]	Vertical oriented	Medium	Medium, Poll, event, Streaming	SensorML compatible	SensorML support	?	No	-	XML	XML structured in SOS interface	-	Web service, XML	Yes	No	Data precision and accuracy	Low, Web-service roles
Sem SOS [118]	Vertical oriented	Medium	Low, Poll, stored	SensorML	SensorML	?	O&M, GML, OWL-Time	O&M data, queries	OWL instances	SPARQL	JENA Engine	Web Svc , XML with XLINK annotation	Yes	No	Data precision and accuracy	Low, Web-service roles
Hour-glass [122]	Peer-to-peer	Medium-High	Medium, Stream & persisted data	Medium to Low	Yes, by Service announce ments	No	-	-	?	No	No	?	Yes, by Service announce ments	Yes, by declaring circuit structure	Medium, Buffering, heart-beat monitoring	No
SONGS [123]	Hierarchic al	Low	Event-based	Low ?	Yes, by Run-time resource info	Yes	Applicatio n-domain oriented	Services, data streams,	XML ?	Semantic based	CLP-R Constraint Progr. Extension	?	Yes, by user queries	Yes, via queries of semantic streams	Data confidelity	No
IBM [124]	Distribute d Stream processing	Medium	Data Streams, event-based	?	No	No	Single, Spatial and temporal	Data Streams, Process. elem.	RDF	SPARQL-like	Yes, OWL-DLP	?	Yes, Via user queries	Yes, via stream pattern queries	?	?
Univ. Toronto [127]	Publish/ subscribe message passing	Medium	Event-based sensor data	Through TinytScript VM	No	No	Sensors to high-level events mapping	High-level events detection	?	No	Semantic engine	?	No	No	No	No
NTUA [126]	Pluggable architectur e	Medium	Event-based and polled access	High	No	No	?	Mapping rules, sensor data	RDF and XML	XPATH & SPARQL ?	Jena Reasoning Engine	Web service	No	No	No	Secure data protocols
The SWAS N [125]	Four layers service - oriented	Medium	Push and poll based	Through Gateways	No	No	Hybrid, sensors, location, domain	Sensor data, transforma tion	RDF	SPARQL	Jena Reasoning Engine	HTTP	No	No	No	No
IrisNet [129]	Hierarchic al	High	Processing with senselets	?	No	No	-	-	XML	XPATH	No	HTTP	No	No	Tolerance of cached data consistenc y	No
The SWAP [130]	Three -tier	High	Poll-based. Event-based	?	?	No	SWEET, OWL-S, OWL-Time	Processing agents description s	Semantic Based ?	?	No	Web Service	Yes by Applicatio n Agents	?	?	?

Постојећа решења заснована на базама података и складиштима RDF података типично користе централизоване системе инсталиране само на једном рачунарском чвору. Таква организација ограничава скалабилност приступа и стога ова решења нису погодна за коришћење у сценаријима са великим бројем сензора и корисника, односно за апликације глобалних размера. Међутим, решења у облаку заједно са дистрибуираним *NoSQL* системима могла би знатно да побољшају овај приступ и његову скалабилност уз адекватну имплементацију складиштења и обраде *RDF* података и *SPARQL* упита. У следећем поглављу биће дат предлог једне архитектуре која побољшава овај приступ увођењем прилагођеног дистрибуираног *RDF* складишта.

Аутори архитектура заснованих на превођењу упита у потпуности користе семантичке технологије које се фокусирају на семантички упитни језик и конверзију ка упитним језицима изворних података. Додатна пажња у истраживањима је посвећена семантичком опису извора података који олакшава фузионисање сензорских података са другим семантичким токовима података. Ово истраживање је произвело проширења *SPARQL* језика као што су $SPARQL_{STREAM}$ и $_{ST}SPARQL$, који су нашли примену и у другим областима. Ипак, остаје неизвесно колики су лимити перформанси таквих система у случају интензивног саобраћаја сензорских података са великим бројем корисника.

Приступи са виртуелизацијом сензора иако настали у великим *EU* пројектима, нису значајно заживели ни после неколико година од завршетка тих пројеката. Иако је информациони модел био доста напредан, био је константно подложан променама, као последица уочених недостатака. Иако је генерално добра скалабилност овог приступа, она је добијена жртвовањем могућности ефикасне фузије разних сензорских извора података као и чувања сензорских података из прошлости. Пројекат *IoT-A* је декларативно подржавао рад са историјским сензорским подацима различите апстракције, али конкретна решења то нису имплементирала. Ипак, решења *GSN* и *X-GSN* су

прилично експлоатисана у слоју података разних архитектура ради трансформације сирових сензорских података у токове података вишег апстрактног нивоа укључујући и семантички модел, који се даље обрађују у вишим слојевима архитектуре система.

Приступ са континуалном обрадом упита је постао доминантан код нових платформи. Нуди могућност комплексније обраде података од архитектуре са брокером порука и IoT протоколима апликативног слоја за пренос порука, а има компаративне перформансе. Са друге стране, интеграција са системима за складиштење *RDF* података и даље није ефикасна, што знатно лимитира примене у неким апликативним сценаријима.

Сервисно-оријентисане архитектуре користе решења заснована на стандардима и примењују конзервативан приступ, али ипак нуде стабилну имплементацију. Примарна достигнућа стандардизације су остварена кроз рад *OGC SWE* радне групе. Концептуални сензорски модел је зрео и већ је широко прихваћен од истраживачке заједнице или директно или индиректно кроз моделе присутне у разним сензорским онтологијама. Приступи са овом архитектуром се константно усавршавају, рецимо *REST* интеракција је присутнија од *SOAP* базираних веб сервисних интерфејса. Даљи развој овог приступа је и даље присутан у неким од великих Е пројеката попут *BIG IoT*[39]. Ипак, остаје доста простора за иновативност у погледу ефикасније примене семантичког приступа.

У тренутку настанка решења са приступом композиције сервиса, тадашња технологија није могла да ефикасно подржи креиране идеје. Са појавом специјализованих хардверских решења за обраду токова података (енг. *data flow*) расположивих и у платформама у облаку, овај приступ може поново доћи у фокус истраживачке заједнице као што је наговештено у [132], са знантно побољшаним перформансама. То би посебно дошло до изражаја у сценаријима употребе које предвиђа визија IoT са отвореним тржиштем за

[39] http://big-iot.eu/project/

пружање специфичне обраде над сензорским подацима. Један вид утицаја идеја овог приступа су решења са континуалним извршавањем упита, који ипак нуде једноставније процесне функционалности. Као прелазни приступ, могао би се креирати приступ са понудом већег броја расположивих комплексних процесирања сензорских података, као резултат композиције више сложених процесних блокова.

Принципи решења са трансформацијом података заснованим на правилима су мање или више инкорпорирани у већини других решења, али утисак је да ова решења нису отишла даље од прототипских. Разлог за то је често ослањање на могућности популарних *RDF* складишта као што је *Jena*, која опет не омогућавају напредне перформансе и велики број конкурентних корисника. Такође, интелигентне функције логичких закључивача су делегирана решењима из *Jena* система. Решење са Универзитета у Торонту [127] је обећавало у погледу остваривих перформанси и функционалности, али побољшања тог решења се нису касније појавила.

Системи засновани на агентима дистрибуирањем агената одговорних за различите функционалности остварују солидне перформансе и обећавају подршку веома скалабилних решења. Агенти су коришћени делом и у другим приступима, рецимо код решења *SIB* [107] које испуњава захтеве *IoT-A* референтне архитектуре. Могућа је примена хибридног приступа неког од анализираних приступа уз адекватно ангажовање софтверских агената. У поменутим решењима са агентима, има доста простора за побољшање кроз подршку за разне модалитете сензорских података, паралелну обраду историјских података са алгоритмима машинског учења итд.

Упоредни преглед карактеристика архитектура оријентисаних ка сензорима и апликацијама дати су у табели 8 и табели 9 респективно. Сада после десетак година од прве појаве архитектура са семантички-заснованом интеграцијом сензорских мрежа, може се закључити да је веома тешко испунити очекиване перформансе у условима сталног повећања броја сензорских уређаја и

заинтересованих корисника, а уједно понудити неке од интелигентних функција на бази семантичког закључивања и слично. Искуство архитектура из великих *EU* пројеката са амбициозним циљевима је да нису шире прихваћена у времену након завршетка тих пројеката. Са друге стране, знатно једноставнији али очигледно веома ефикасни приступи са архитектуром на бази брокера порука и IoT апликативним протоколима за пренос порука су доста присутнији. Разлог за ограничену распрострањеност архитектура са семантичким приступом лежи у понекад комплексним информационим моделима представљеним у сензорским онтологијама. Као резултат, препоручена *SSN* онтологија [12][86] је доживела ревизију крајем 2017. године [87] у циљу смањења њене сложености, а такође су се јавили и други предлози једноставнијих сензорских онтологија попут *IoT-Lite Ontology* [133]. Још један разлог недовољне присутности архитектура са семантичким приступом је што развој савремених *RDF* складишта не иде у правцу подударном са захтевима динамичких сензорских апликација и интензивног генерисања сензорских опажања, већ искључиво у правцу ефикасне обраде велике количине статичког скупа *RDF* података представљених у складу са парадигмом *Linked Data*.

У следећем поглављу биће дат предлог архитектуре која се ослања на дистрибуирано *RDF* складиште са ефикасном подршком за обраду динамички генерисаних сензорских опажања.

4. ПРЕДЛОГ АРХИТЕКТУРЕ ЗА СЕМАНТИЧКУ ИНТЕГРАЦИЈУ СЕНЗОРСКИХ МРЕЖА

Анализа из претходног поглавља је показала недостатке постојећих решења за семантички засновану интеграцију сензорских мрежа. Очигледан је недостатак ефикасних, скалабилних *RDF* складишта података који би омогућили складиштење семантичких сензорских опажања и каснију ефикасну манипулацију над тим подацима, поготово у циљу проналажења скривеног знања или извођења новог знања. Код решења *Linked Stream Middleware (LSM)* [109], које је језгро платформе *OpenIoT* [111], идеја је да се хибридним приступом споје добре особине приступа са континуалним извршавањем упита за обраду најсвежијих токова сензорских података, са приступом оријентисаним према базама података који би подржао чување сензорских опажања из прошлости и статичких сензорских података. Међутим, аутори *LSM* платформе су пријавили [109] да са иоле већим приливом сензорских података, једно од најпознатијих доступних *RDF* складишта *Virtuoso*[40], отказује рад и није у могућности да подржи такву идеју својим перформансама.

Сличан проблем се својевремено појавио са наглим порастом количине података и броја активних корисника у апликацијама претраживача Интернета, друштвених мрежа, глобалних аукција, дакле апликација глобалних Интернета размера, са захтевима за високом доступношћу и подршком за милионе корисника. Покушај да се ове апликације реализују у традиционалној архитектури ослањањем на системе за управљање релационим базама података за складиштење података нису успели, јер су ови системи типично централизовани и нису били у могућности да ефикасно скалирају са порастом количине података и корисника. Такође, ти подаци нису били структурирани и варирали су од кратких порука, до мултимедијалних садржаја у реалном времену, што опет није погодовало релационом моделу података за складиштење.

[40] https://virtuoso.openlinksw.com/

Захтеви за подршком ових апликација глобалних размера, мотивисали су истраживаче да развију нова дистрибуирана складишта података која нуде високу доступност података и у стању су да управљају петабајтима података распоређених на хиљадама рачунарских чворова. Ови системи се обично називају *NoSQL (NotOnly SQL)* решења[41]. За разлику од традиционалних *RDBMS*, ови системи најчешће не нуде релациони модел података и обично имају индексни начин приступања подацима, а подржавају радије слабу него јаку конзистентност над подацима, што у датим апликацијама није превише од интереса. *NoSQL* решења су обично изразито дистрибуирана (али не морају да буду) и распоређују податке по чворовима коришћењем структура као што су дистубуиране хеш табеле. Дистибуиране хеш табеле су развијене за потребе брзе претраге података код *peer-to-peer* система великих размера. Временом се издвојило неколико категорија ових решења и то: кључ-вредност складишта, колонска складишта, складишта докумената и графовске базе података.

Додатни технолошки напредак у последњих неколико година је настао ширењем платформи у облаку и појефтињењем RAM меморија, што је резултирало појавом меморијских база података (енг. *in-memory database systems*) [134] типично инсталираних на меморијском облаку (енг. *memory cloud*). За разлику од релационих база података и већег броја *NoSQL* складишта који чувају податке на диску, ови системи своје податке чувају у оперативној меморији што доприноси веома брзом одзиву ових система. *TrinityDB*[42] [135] и *Redis*[43] су типични представници ових решења.

Узимајући у обзир претходну анализу постојећег стања архитектура за семантички засновану интеграцију сензорских мрежа и технолошких решења за дистрибуирано складиштење неструктурираних података у меморији, дефинисани су захтеви архитектуре чијом би реализацијом били побољшани

[41] http://nosql-database.org/
[42] https://www.microsoft.com/en-us/research/project/trinity/
[43] https://redis.io/

кључни недостаци постојећих решења и тиме ефикасно подржани случајеви употребе разних сензорских односно IoT апликација. Ти захтеви су следећи:

- Креирање архитектуре која се заснива на дистрибуираном *RDF* складишту са чувањем података у меморији ради остваривања високих перформанси односно могућности брзог смештања нових сензорских очитавања и брзог извршавања семантичких упита у језику *SPARQL*.
- Паралелно прихватање нових сензорских опажања интензивно генерисаних и извршавање *SPARQL* корисничких упита за дохватањем сензорских података из прошлости или комбиновањем нових и старих сензорских података.
- Подршка за чување велике количине статичких сензорских података који су заправо описи сензора и уређаја.
- Креирање индекса треба да и у погледу величине заузетог простора и степена комплексности обраде нових *RDF* тројки буде брза и једноставна операција како би се омогућило брзо додавање нових тројки претпостављајући динамичко генерисање нових података у окружењу са потенцијално већим бројем клијената који повремено објављују нове *RDF* тројке. Технике попут комплексног претпроцесирања *RDF* тројки у сврси оптималног партиционисања *RDF* скупа по чворовима или компресије ради компактног складиштења *RDF* тројки, у овом контексту су неефикасне, јер се претпоставља рад у реалном времену, а не рад као у системима за чување *LOD (Linked Open Data)*. Код таквих *RDF* складишта се претпоставља убацивање *RDF* тројки улазног скупа података на почетку рада, а не паралелно са задавањем упита, или се евентуално очекује веома ретко додавање нових тројки.
- Првенствени циљ архитектуре је подршка апликација из сензорског домена, што повлачи чињеницу да се анализирају сензорске онтологије и њихов информациони модел односно типичан узорак генерисаних сензорских опажања као и најчешћи тип упита.

- Подршка за просторно-временске податке треба бити остварена по могућству кроз специјализоване структуре које ефикасно чувају и манипулишу таквим подацима. То истовремено значи да је фокус на извршавању протсторно-временских упита.

У складу са горњим захтевима у наредном одељку биће дат опис предложене архитектуре.

4.1. Предлог архитектуре засноване на дистрибуираном RDF складишту за семантичку интеграцију сензорских мрежа

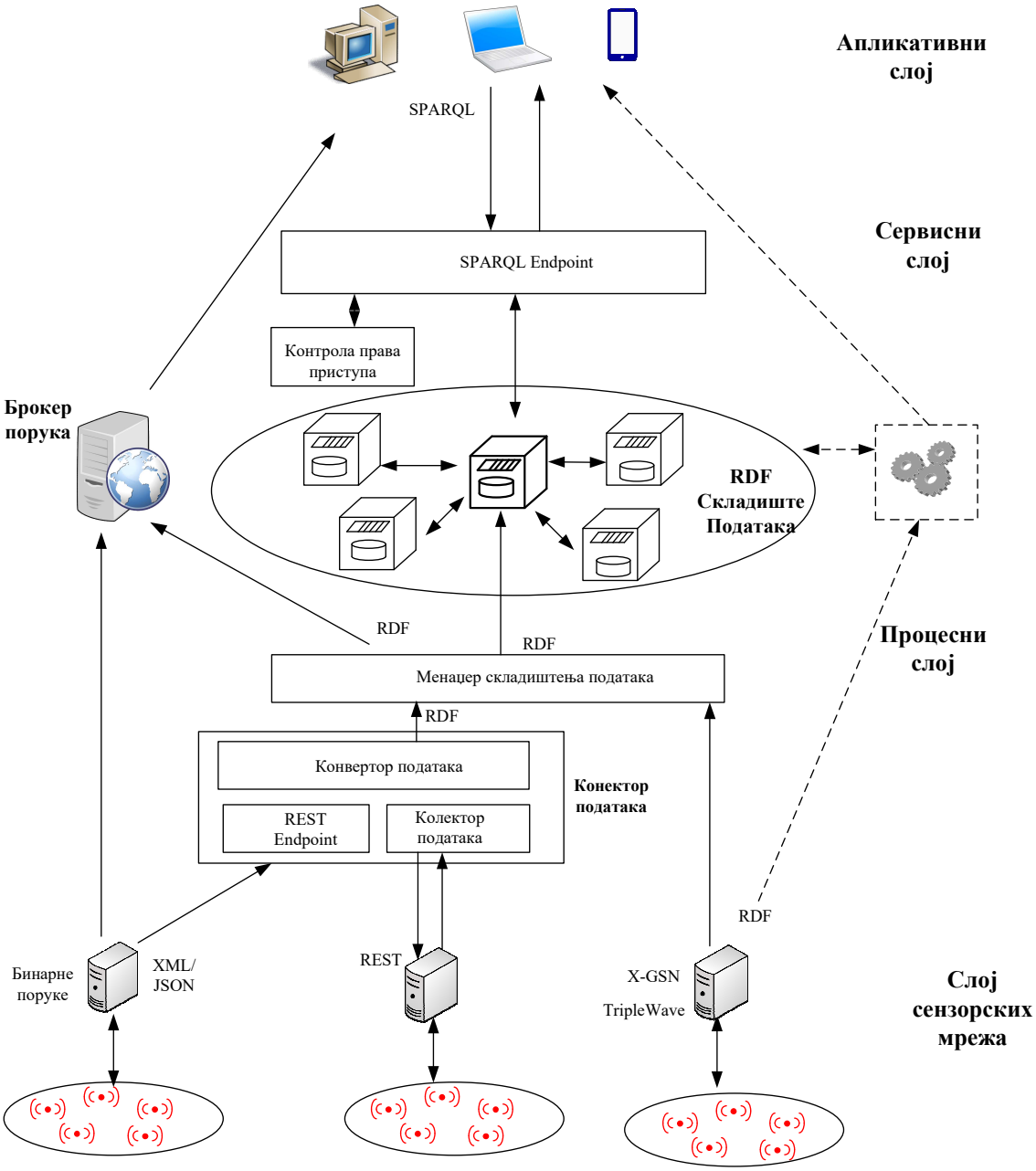
На слици 23 је представљена архитектура за интеграцију сензорских мрежа заснована на дистрибуираном *RDF* складишту. Идеја је да *RDF* складиште чува прикупљене сензорске податке који укључују и статичке сензорске описе и динамички генерисана сензорска опажања у складу са прихваћеним моделом у стандардизованој сензорској онтологији *W3C SSN* [12, 86, 87] или сличној. Архитектура примарно користи приступ оријентисан ка бази података, при чему се дистрибуирањем *RDF* складишта и смештањем сензорских података у оперативној меморији, покушава да отклони уско грло протока података који је постојао у традиционалним решењима са коришћењем диска. Међутим, обрада токова сензорских података није занемарена, иако на знатно једноставнијем нивоу него код приступа са континуалном обрадом упита.

Архитектура је логички подељена у неколико слојева. У слоју сензорских мрежа се прикупљају сензорски подаци из сензорских мрежа и преко пролазне компоненте се прослеђују даље кроз архитектуру. Претпоставља се генерисање семантичких сензорских података преко система као што су *XGSN* [103] и *Triple Wave* [136], а семантички сензорски подаци се могу генерисати и изворно поготово на *SBC* компактним рачунарским модулима (енг. *Single*

Board Computers), тј. без конверзије из других формата. Слично, на основу мапирања представљених у одређеним језицима попут *R2RML* [103][136] или *XML2OWL* [47], у *конвертору података* могуће је примљене податке представљене у бинарном, релационом, *JSON* или *XML* формату, конвертовати у токове семантичких сензорских података (енг. *semantic sensor streams*). Подразумевани начин је да се сензорски подаци шаљу из пролазних компоненти, а слично као у пројекту *SPITFIRE* [96], архитектура може интегрисати и *колектор података* који захтева податке из сензорских мрежа преко стандардизованих интерфејса (између осталог, то може бити преко *OGC SWE SOS* сервиса). Преглед савремених техника претраге за изворима сензорских података је дат у референци [137].

Главни ток семантичких сензорских података се прослеђује даље ка *RDF* складишту, односно ка одговарајућем чвору, под контролом менаџера складиштења података, који у складу са усвојеном стратегијом партиционисања података на основу предиката распоређује *RDF* тројке. Складиштени семантички подаци се одмах могу прослеђивати корисницима као резултат извршавања упита.

Архитектура допушта и директно прослеђивање токова сензорских података корисницима преко брокера порука, као и у стандардној архитектури са брокером порука описаној у поглављу 3.4.2. За разлику од те архитектуре овде се претпоставља да се уз бинарне, *JSON*, *XML* или на сличан начин представљене податке, прослеђују и семантички сензорски подаци, уз адекватно именовање теме.



Слика 23 – Предлог архитектуре за семантичку интеграцију сензорских мрежа засноване на дистрибуираном RDF складишту

Овим се на један крајње поједностављен начин врши базично филтрирање семантичких података, на пример, корисници могу захтевати да им се прослеђују подаци са новим сензорским опажањима са одређених сензора или само описи нових сензора и слично. Ова функција ни на који начин није замишљена као замена за функцију континуалног извршавања упита присутне у одређеним архитектурама. Као могућност каснијег проширења се предвиђа интеграција са процесором континуираних упита попут *CQELS* [110], на начин да се парцијални резултат *SPARQL* упита може проследити

процесору упита и на тај омогућити филтрирање свеже пристиглих сензорских података на основу услова које задовољавају сензорски подаци из прошлости. Тиме би се заправо проширио тзв. прозор који обухватају процесори континуалног тока података. Рецимо, такав један упит би захтевао да се кориснику прослеђују подаци са температурног сензора, који је у некој области у току јучерашњег дана измерио вредност већу од дефинисаног прага.

Стандардно, апликације преко *SPARQL* приступне тачке захтевају семантичке сензорске податке преко адекватних *SPARQL* упита, при чему се консултују права приступа.

4.2. Архитектура RDF складишта за подршку семантичке интеграције сензорских мрежа

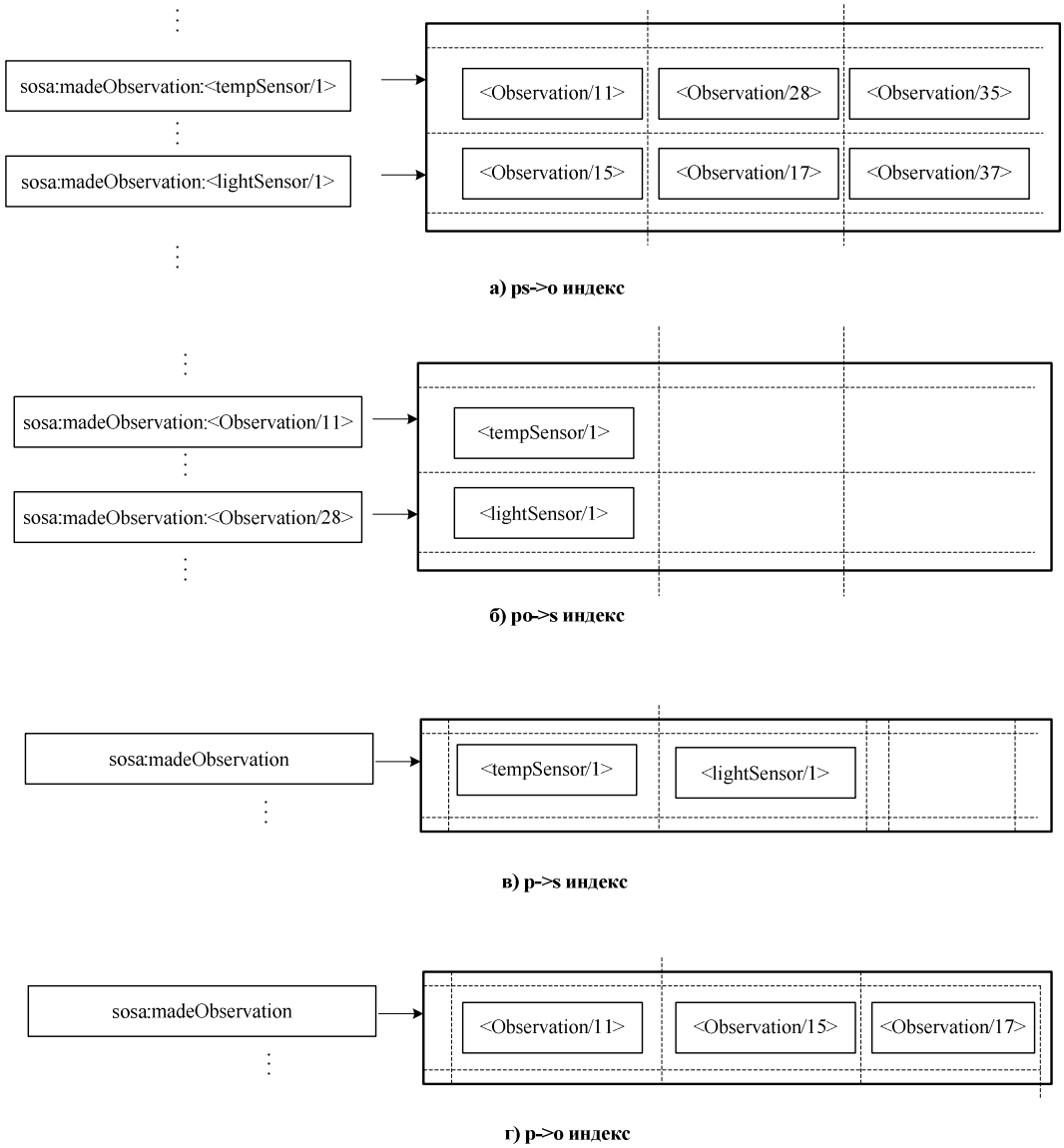
Кључне пројектантске одлуке код сваког *RDF* складишта је начин складиштења *RDF* тројки, односно начин њиховог индексирања, који се користи приликом извршавања *SPARQL* упита и евентуално начин дистрибуције скупа података ако је одабран дистрибуирани приступ.

Имплементација *RDF* складишта као и начин чувања *RDF* тројки је историјски пратила достигнућа у савременим системима база података [138, 139]. Тако је почетком и средином 2000-их доминирао приступ са релационим базама података и чувањем *RDF* тројки у једној или више табела базе података, а са појавом колонских база података, а затим и других *NoSQL* решења, доминира индексни приступ код чувања *RDF* тројки, најчешће са 6 индекса по једној *RDF* тројки, али постоје приступи и са 3 индекса [138, 139].

Пре описа коришћеног начина индексирања, треба напоменути да се подразумева да су *RDF* тројке представљене у форми <*s p o*>, где је *s* субјекат, *p* предикат, а *o* објекат. Субјекат и предикат морају бити представљени са *URI* (заправо *IRI - Internationalized Resource Identifier*) термином јер представљају концепте, а објекти могу бити или *URI* или литерали. Формална дефиниција је дата у одељку 2.5.1.

У складу са циљевима наведеним на почетку поглавља, изабран је тзв. предикатски приступ индексирања у којем се тројке чувају у индексима базираним на предикатима, конкретно *ps->o* и *po->s* индексима за једну *RDF* тројку. Традиционални предикатски приступ би био *p->so* и *p->os* што би значило да се за сваку тројку са предикатом *p,* чувају сви парови *s* и *p* који постоје за дати предикат *p*. С обзиром да је типичан број предиката мали (реда пар стотина, до највише неколико хиљада, а код сензорских онтологија испод сто), а број субјеката и објеката може да буде и више милиона, са традиционалним индексом би се добила структура података која је непогодна за ефикасно индексирање у структурама са хеш табелом. Заправо, ако је број предиката n_p, било би потребно n_p хеш табела са потенцијално великим бројем улаза. Знајући да перформансе хеш табеле значајно падају ако је попуњеност већа од 50%, добио би се велики неискоришћени простор ако бисмо желели добре перформансе.

У одабраном приступу, индекс односно кључ за улаз у индексну структуру се добија конкатенацијом вредности предиката и субјекта, односно предиката и објекта у другом случају, у складу са сликом 24. Индекс *ps* указује на листу свих објеката који су преко датог предиката *p* у вези са субјектом *s,* а слично индекс *po* указује на листу субјеката са којима је дати објекат *o* преко предиката *p* у вези. Овакав приступ нам омогућава да имамо заправо само две хеш табеле, које би имале велики број улаза, реда милиона, а опет број елемената у листи повезаних објеката односно субјеката би био умерен и наравно зависи од дистрибуције субјеката и објеката, али типично може ићи од неколико до више хиљада, ретко више од тога.



Слика 24 – Приказ структуре података са предикатским индексима

Оваква распоређеност кључева погодује савременим кључ-вредност складиштима и дистрибуираним колонским складиштима заснованим на дистрибуираним хеш табелама и моделом података са фамилијом променљивог броја колона као код *Google BigTable* [140]. То су решења попут *Apache HBase* [141] и *Apache Cassandra*[44] [142]. Међутим, тренутно најефикаснија решења за складиштење такође користе дистрибуиране хеш табеле за дистрибуцију података по рачунарским чворовима, али податке смештају у меморији. Примери су већ поменути *TrinityDB* [135] и *Redis*. Како би се ефикасно користила дата складишта, могуће је креирати јединствене индексе

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

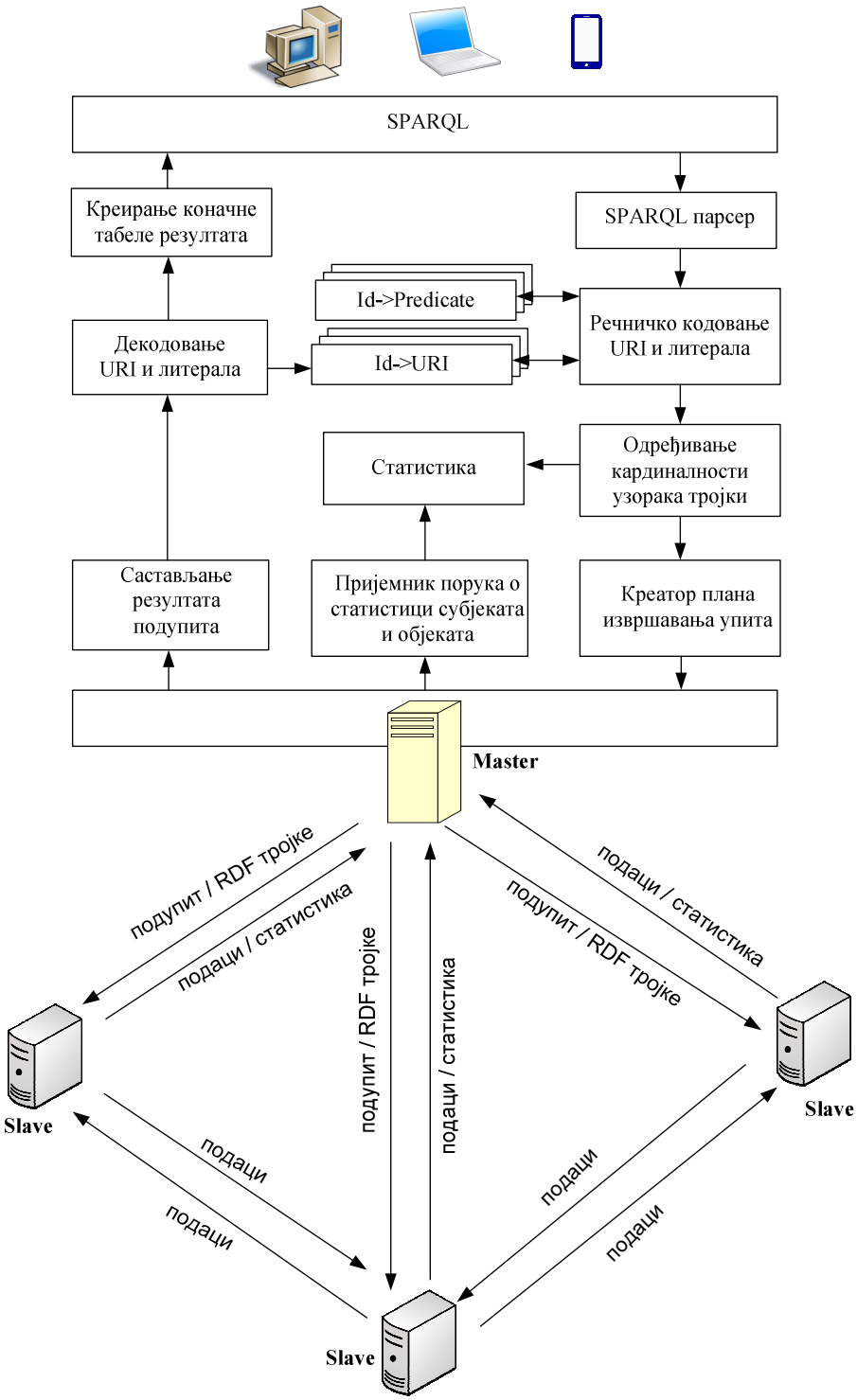
који омогућавају коришћење само једне хеш табеле односно дистрибуираног кључ-вредност складишта, а само једним флегом означити да се ради о *ps* или *po* индексу. У детаљима имплементације ово ће бити детаљније објашњено.

Осим *ps->o* и *po->s* индекса, чувају се такође и *ps* и *po* односно *p->s* и *p->o* индекси који садрже листу свих субјеката и објеката који постоје за дати предикат. Њихово коришћење ће бити објашњено у делу евалуације упита, али такође су корисни због чувања статистике дистрибуираности субјеката и објеката по предикатима, односно кардиналности узорака тројки.

Други критеријум приликом пројектовања *RDF* складишта се односи на дистрибуираност индекса. Идеја је да се индекси дистрибуирају по расположивим рачунарским чворовима у стандардној *master-slave* организацији, тако да се индекси који припадају једном предикату налазе на једном чвору. Овакав приступ оставља могућност потпуне хетерогене имплементације структуре индекса у зависности од конкретног предиката и дистрибуираности субјеката и објеката који иду уз тај предикат. Распон могућих имплементација иде од централизоване организације, када се индекси свих предиката налазе на истом чвору и у истој структури података. Крајња дистрибуираност би ишла ка томе да се индекс сваког појединачног предиката налази на посебном рачунарском чвору или да поседује специфичну структуру индекса, али такав један приступ не би имао пуно смисла у контексту извршавања упита и потребе за операцијом спајања резултата подупита. Добро балансиран приступ би омогућавао груписање сродних предиката у индексној структури на истом чвору, како би се честе операције спајања приликом извршавања упита извршавале локално на истом чвору. Следећи разлог издвајања структуре индекса на истом или засебном чвору је другачија структура индекса што посебно долази до изражаја код просторно-временских података.

Осим експлицитне дистрибуције индекса по чворовима на основу предиката, могућа је и додатна дистрибуираност индекса по расположивим рачунарским чворовима, која се може постићи дистрибуираном имплементацијом

структуре података за чување индекса заснованој на дистрибуираној хеш табели, са неким од поменутих дистрибуираних складишта. Принцип рада таквог једног складишта је да се кључ за улаз у дистрибуирану структуру пресликава преко хеш функције у чвор који чува припадајућу вредност тог кључа.



Слика 25 – Архитектура RDF складишта

Архитектура *RDF* складишта је приказана на слици 25. Горњи део слике представља компоненте унутар *master* компоненте, а доњи приказује организацију и везу између *master* и *slave* компоненти. Прихватање корисничких упита, иницијално препроцесирање, обраду и план извршавања упита извршава *master* компонента, док чување индекса обављају *slave* компоненте (у дистрибуираној организацији), али такође и *master* компонента (у централизованој, а по потреби и у дистрибуираној организацији). Комуникација између *mastera* и *slave* компоненти је асинхрона, остварује се преко *TCP* конекције преко библиотеке *KryoNet*[45], која је ефикасна алтернатива *MPI (Message Passing Interface)* интерфејсу у условима дистрибуиране архитектуре са стандардним PC рачунарима. У питању је библиотека за мрежну комуникацију са малим кашњењем и сопственом имплементацијом компактне бинарне серијализације објеката преко формата *Kryo* поменутим у поглављу 3.3.2., а нарочито је популарна у дистрибуираним платформама за обраду огромне количине података (енг. *big data*). Наравно, могућа је и централизована варијанта, у којој постоји само *master* компонента. Пошто је реализација прототипа извршена у програмском језику *Java*, то допушта да се компоненте извршавају на било којем оперативном систему који подржава *Java* виртуелну машину (енг. *JVM*), што омогућава флексибилност у конфигурисању компонената архитектуре.

Стандардна операција код свих *RDF* складишта је да се због велике дужине *URI*-ја и литерала, они речнички кодују преко целобројних идентификатора, дужине 32 или 64 бита, или евентуално преко записа друге дужине, али је пожељно да буде цео број бајтова.

То значи да се на почетку изврши речничко кодовање тако што се редом узимају вредности идентификатора (*id)* и ти целобројни *id*-јеви се чувају у индексним структурама. У посебној хеш табели се чува мапирање вредности датог *URI*-ја или литерала ка *id*-у, што се користи у случају поновног

[45] https://github.com/EsotericSoftware/kryonet

појављивања истог термина како би се доделио исти *id*. У овој имплементацији, предикати се засебно кодују зато што је обично довољно један или два бајта да би се кодовали сви предикати, односно максимално 255 или 65535, с обзиром да је код 0 увек резервисан за предикат *rdf:type*, коришћен за означавање којој класи припада неки субјекат. Сада већ може да се објасни начин формирања индекса за случај ширине *id*-јева субјеката и објеката од 40 бита и 8 бита за кодовање предиката:

Одвојене хеш табеле:
- *ps* кључ [$s_{40}..s_0p_7..p_0$]
- *po* кључ [$o_{40}..o_0p_7..p_0$].

Јединствена хеш табела:
- *ps* кључ [$s_{40}..s_0p_7..p_01$]
- *po* кључ [$o_{40}..o_0p_7..p_00$]

Након кодовања *URI*-ја и литерала, приступа се одређивању кадиналности субјеката и објеката за узорке тројки датог *SPARQL* упита ради процене селективности неког узорка тројки (енг. *selectivity estimation*). Сваки узорак тројки може на месту субјекта, предиката и објекта имати или променљиву или везану вредност (енг. *bound*), што се може означити са знаком *?* ако се користи променљива или словом *S, P, O* за везану вредност, у зависности од позиције. На основу типа узорка тројки, узимају се вредности из табеле 10 за попуњавање кардиналности субјеката и објеката. Наиме, за случај узорка тројки када објекат није везан (случај *SP?*), узима се статистика за предикат *P* и број његових субјеката и објеката. Субјекат је овде везан за одређену вредност, па се процењује да објекат може имати $|obj_p|/|subj_p|$ вредности, пошто субјекат има кардиналност 1. Изрази $|obj_p|$ и $|subj_p|$ означавају укупан број вредности објеката и субјеката на предикату *p* респективно, што је број елемената у структури индекса *p->o* односно *p->s*. Треба напоменути да се у датим листама субјеката и објеката чувају и поновљене вредности. Наравно, у питању су хеуристичке вредности за кардиналност, које у неким ситуацијама одступају значајно од тачних вредности, пошто се претпоставља униформна расподела субјеката и објеката. Ови индекси су дистрибуирани на *slave* компонентама (ако је изабрана дистрибуирана имплементација), тако да је потребно периодично слати поруке са статистиком ка *masteru*. Једини изузетак

од горњих правила је случај за узорак тројки *?PO* за предикат *rdf:type* што је чест случај у *SPARQL* упитима. Наиме, за сваку класу се чува број инстанци, тако што се приликом убацивања нових *RDF* тројки детектује тројка одређивања типа (предикат *rdf:type*) приликом кодовања *URI*-ја. Ова операција се увек извршава на *master* чвору, тако да одржавање статистике бројања инстанци неке класе није захтевно.

Табела 10 – Вредности кардиналности за субјекте и објекте код различитих узорака тројки (за вредности означене са * потребне су информације о релацијама између класа).

Тип узорка тројки	Кардиналност субјекта	Кардиналност објекта
SPO	1	1
SP?	1	$\lvert obj_p \rvert / \lvert subj_p \rvert$
?PO	$\lvert subj_p \rvert / \lvert obj_p \rvert$	1
?P?	$\lvert subj_p \rvert$	$\lvert obj_p \rvert$
S?O	X^*	X^*
S??	1	$\sum \lvert obj_p \rvert^*$
??O	$\sum_p \lvert subj_p \rvert^*$	1

На основу процењене кардиналности субјеката и објеката врши се евалуација извршавања *SPARQL* упита, што ће бити детаљно објашњено у следећем одељку. Након одређивања врсте подупита и њиховог редоследа извршавања *master* шаље подупите одговарајућим *slave* компонентама, које враћају резултате извршавања подупита, или прослеђују парцијалне резултате другим *slave* компонентама. По пријему резултата, може да се врши спајање међурезултата упита и након тога декодовање *id*-јева у *URI*-је и литерале и саставља се коначни резултат упита попуњавањем табеле са селектованим променљивама.

4.3. Креирање плана извршавања SPARQL упита

Претрага *RDF* података се врши користећи концепт *Узорака Основног Графа* (енг. *Basic Graph Patterns – BGP*) који је низ повезаних упита креираних за *RDF* модел података. *BGP* се дефинише на следећи начин:

Дефиниција 2: Нека је $\boldsymbol{U}$ потенцијално бесконачан скуп *URI* ресурса који идентификују чворове у графу. $\boldsymbol{U}$ је скуп атома (стрингова). Нека је $\boldsymbol{V}$ коначан скуп променљивих, различит од $\boldsymbol{U}$. Узорак тројке (енг. *triple pattern*) је елемент скупа $(\boldsymbol{U} \cup \boldsymbol{V}) \times (\boldsymbol{U} \cup \boldsymbol{V}) \times (\boldsymbol{U} \cup \boldsymbol{V})$. Другим речима, узорак тројке је тројка (најчешће интерпретирана као израз) у којој се уместо субјекта, предиката и објекта појављују или атоми или променљиве. *Узорак Основног Графа* (*BGP*) је конјукција $(s_1, p_1, o_1) \wedge \ldots \wedge (s_m, p_m, o_m)$ једног или више узорака тројки.

Један *SPARQL* упит представља узорак основног графа и састоји се из конјукције узорака тројки. У процесу евалуације узорка основног графа, повезујући узорке тројки на основу истих променљивих у њима, може се креирати граф упита. У том процесу могу се идентификовати следећи случајеви:

- Ако узорци тројки на месту субјеката имају исту променљиву добијамо звездасту везу између групе узорака тројки. На тај начин су груписана својства неке инстанце, односно упитом се селектују инстанце које деле та својства (односно предикате). То у општем случају не морају али могу бити инстанце неке класе, односно инстанце које припадају некој генерализованој класи дефинисаној у *OWL* или *RDFS* онтологији. Оваква веза узорака тројки спојених преко једне променљиве се зове *звездаста група* (енг. *star-shaped group*). Са предложеном организацијом индекса преко предиката, можемо у звездасту групу неке променљиве из упита укључити и променљиве на месту објекта, јер са становишта имплементације на основу креираних индекса нема разлике, односно само ће друга структура бити коришћена у процесу разрешавања упита. Резултат спајања (енг. *join*) звездасте групе је пресек конјуктинвих узорака тројки и укључује оне инстанце (чворове) из *RDF* графа који имају *RDF* тројке са предикатима из узорака тројки ка другим инстанцама или литералима, што представља пресек скупа субјеката или евентуално објеката по датим предикатима.
- Ако су две променљиве повезане преко неког предиката, таква веза говори о асоцијацији између две групе изабраних инстанци, односно да

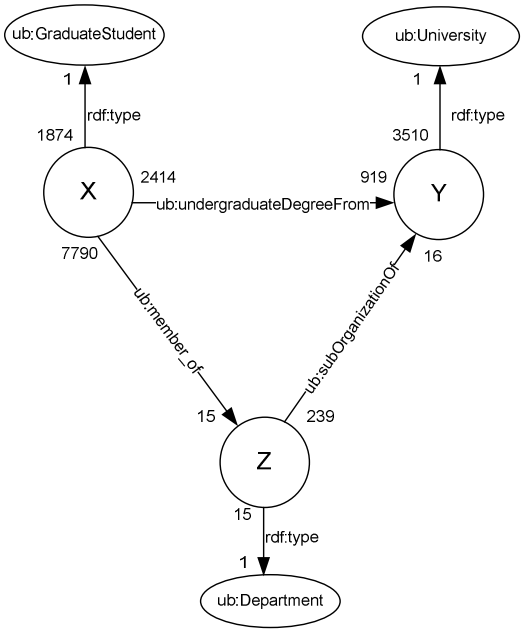
је потребно извршити спајање по датим предикатима. У терминологији токова података (енг. *data flow*), радије се говори о *пресликавању (*енг. *mapping)* једног скупа инстанци у други скуп инстанци, при чему могу настати дуплиране вредности неких инстанци, тј. веза између инстанци је типа више ка више. Дакле, за променљиве X и Y имамо $X \rightarrow_p Y$, односно у нотацији спајања $X \bowtie_p Y$.

Граф *SPARQL* упита се креира тако што се за чворове графа узму променљиве, а гране у графу су предикати између променљивих. Такође, уводи се обележавање кардиналности чворова графа за сваку грану из статистике дистрибуираности субјеката и објеката по предикатима. Везане променљиве су такође чворови у графу, а њихова кардиналност је увек 1. Граф упита ће бити коришћен у процесу креирања редоследа извршавања упита.

Процес креирања графа *SPARQL* упита ће бити илустрован конкретним примером на тестном скупу података *LUBM-0 (Lehigh University Benchmark)*[46], иначе популарном генеричком скупу *RDF* података намењеном за тестирање *RDF* складишта, с обзиром да је конфигурабилна величина генерисаног скупа података за онтологију из домена универзитета, којом се моделују везе између наставног особља, студената, припадајућих одсека, предмета, објављених радова, књига и других ентитета. За пример је узет стандардни упит 2, а резултујући граф је приказан на слици 26:

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z
WHERE
{?X rdf:type ub:GraduateStudent .
  ?Y rdf:type ub:University .
  ?Z rdf:type ub:Department .
  ?X ub:memberOf ?Z .
  ?Z ub:subOrganizationOf ?Y .
  ?X ub:undergraduateDegreeFrom ?Y}
```

[46] http://swat.cse.lehigh.edu/projects/lubm/



Слика 26 – Граф SPARQL упита за тестни упит #2
генеричког тестног скупа LUBM-0

На основу креираног графа, врши се евалуација редоследа извршавања операција спајања у циљу проналажења инстанци основног узорка графа. Основни критеријум при евалуацији плана извршавања упита је минимизација цене коштања појединачних операција спајања, чиме се постиже и минимизација укупне цене коштања извршавања упита. Овај критеријум тежи да минимизује број инстанци које су резултат операције пресека приликом спајања звездасте групе узорака тројки или операције мапирања код спајања инстанци различитих променљивих упита. Процена броја инстанци се врши на основу коришћене статистике и то тако да се:

- У случају операције пресека, као резултантни број инстанци узима се минимална кардиналност по неком узорку тројки дате променљиве, односно, за дату променљиву μ:

$$|\cap_{\mu}^{*}| = \min |\mu|_{p} \qquad (7)$$

  Јасно је да је ово горња граница реалног броја инстанци резултата пресека.
- Код операције пресликавања односно спајања променљивих по неком предикату p узима се пројекција кардиналности на основу текуће

кардиналности (у смислу процењене кардиналности у току евалуације упита) скалиране са почетним односом кардиналности између субјеката и објеката на датом предикату p. Дакле, за операцију мапирања $X ->_p Y$ имамо:

$$|Y_{->p}| = |X| * |obj_p| / |subj_p| \quad (8)$$

ако је X на месту субјекта код узорка тројки на предикату p, односно

$$|Y_{->p}| = |X| * |subj_p| / |obj_p| \quad (9)$$

ако је X на позицији објекта.

На основу усвојеног критеријума, креиран је алгоритам креирања редоследа извршавања упита приказан у наставку:

АЛГОРИТАМ 1: Одређивање редоследа извршавања упита

```
1 for each vertex in query_graph, do
2     select vertex min cardinality
3     create star_shaped_join_task (vertex) estimated with vertex cardinality
4     sorted_task_candidate_list ← star_shape_join_task
5 end
6 while sorted_task_candidate_list is not empty, do
7     current_execution_task ← head of sorted_task_candidate_list
8     query_execution_task_list ← current_execution_task
9     curent_vertex cardinality ← target_vertex_estimated_cardinality of current_execution_task
10    if current_execution_task is star_shaped_join_task, then
11        mark curent_vertex as star_shaped_joined
12        remove current_vertex merge_join_task from sorted_task_candidate_list
13    else if current_execution_task is mapping_ join_task, then
14        mark related edge as visited
15        remove related edge_mapping_join_task from sorted_task_candidate_list
16    end
17    if current_vertex is not_star_shaped_joined, then
18        create star_shaped_join_task (current_vertex) estimated with current_vertex cardinality
19        sorted_task_candidate_list ← star_shaped_join_task
20    else
21        for each edge in edges_to_variable_vertices of curent_vertex, do
22            if edge is not visited then
23                create mapping_join_task from curent_vertex to edge_other_side_vertex
```

```
24                  mapping_ join_task estimated_cardinality ← curent_vertex_cardinality*edge_mapping_rate
25                  sorted_task_candidate_list ← mapping_ join_task
26              end
27          end
28      end
29  end
```

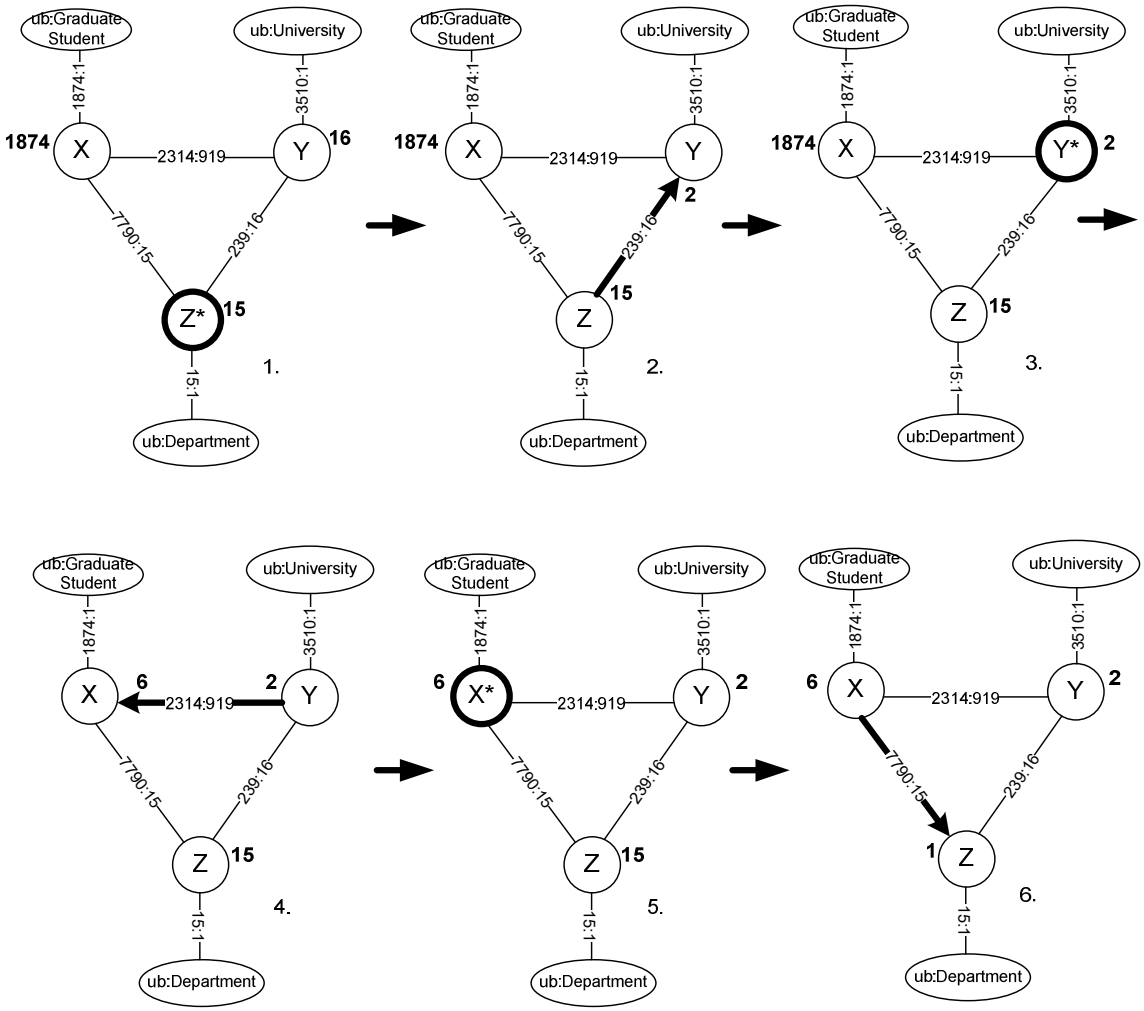
Дакле, најпре се креирају таскови са операцијом спајања сједињавањем (енг. *merge join*) над променљивом (чвор у графу упита) са звездастом везом узорака тројки (спајање над звездастом групом узорака тројки) у *SPARQL* упиту који су сортирани по минималној кардиналности датог чвора. У свакој итерацији узима се таск са минималном процењеном кардиналношћу која би била резултат извршавања тог таска. За текући извршни таск спајања над звездастом групом најпре се обележи дати чвор да је извршио „звездасто спајање“ и креирају се кандидат таскови за сваки узорак тројки датог чвора за који до тада није креиран исти такав таск мапирања али у супротном смеру, усмерен ка чворовима променљивих (гране чворова везаних променљивих учествују увек у звездастом спајању). Након извршавања таска мапирања односно спајања са другом променљивом може се креирати или таск са спајањем над звездастом групом узорака тројки, ако до тада није извршено звездасто спајање и постоји бар још једна непосећена грана од тог текућег чвора, или опет таск са мапирањем ка следећој променљиви.

Треба напоменути да се код операција звездастог спајања користе индекси *p->s* и *p->o* осим код гране (узорка тројки) ка неповезаној променљиви, када се из индекса *ps->o* и *po->s* за дати термин добија листа могућих инстанци за ту променљиву. Редослед извршавања спајања између грана-узорака тројки се динамички одређује у току самог извршавања у зависности од броја елемената у низу у одговарајућој индексној сортираној структури (стаблу).

Сада је јасна идеја да се више предиката-особина који припадају одређеној класи из онтологије групишу на истом рачунарском чвору, јер у типичном *SPARQL* упиту операција спајања се врши над узорцима тројки које одговарају својствима исте класе. У идеалном случају, спајање конјуктивних узорака

тројки над једном променљивом из *SPARQL* упита ће се извршити на једном чвору. Наравно, оваква архитектура допушта да индекси неких предиката буду репликовани на више чворова како би то довело до локалног извршавања операција спајања.

Процес евалуације упита ће бити опет илустрован на истом примеру упита број 2 из *LUBM-0* тестног скупа података и приказан је на слици 27.

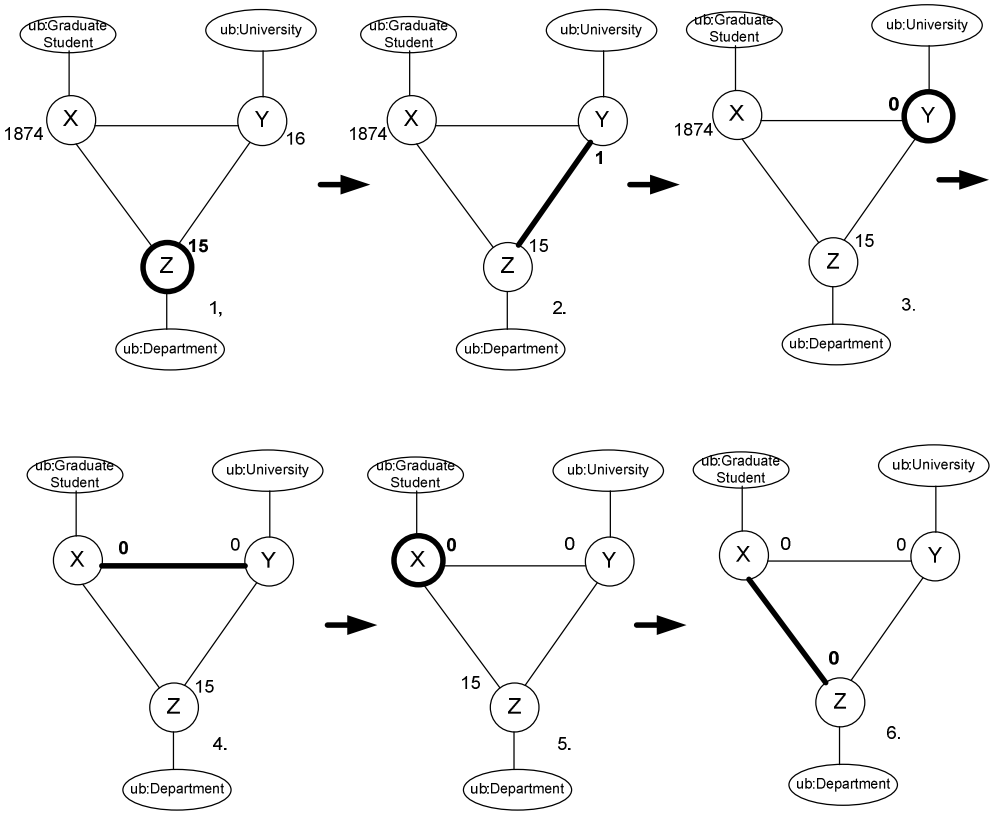


Слика 27 - Креирање редоследа извршавања упита, са процењеном кардиналошћу након сваке операције

Након почетног одређивања минималне кардиналности за сваки чвор, прва операција је $|\cap_{Z^*}|$, звездасто спајање за чвор *Z* и са процењеном кардиналношћу 15. У следећем кораку се врши мапирање из чвора *Z* ка чвору *Y* са процењеном кардиналношћу 2, тј. $|Z \rightarrow Y| = 2$. У кораку 3. се врши звездасто спајање на чвору *Y* и остаје иста кардиналност 2, да би се у кораку 4

извршило мапирање ка чвору *X* и процењеном кардиналношћу 6, што остаје и у кораку 5. Последњи корак је мапирање *X* -> *Z*, уз процењену кардиналност 1, што значи да се додатно редукује скуп инстанци промењиве *Z*.

Тачност процене кардиналности резултата сваке операције у процесу евалуације упита могуће је одредити упоређивањем са подацима узетих у току стварног извршавања упита по горњем редоследу. Редослед је приказан на слици 28.



Слика 28 – Кардиналности чворова графа упита након сваке операције у току стварног извршавања SPARQL упита

Почетни корак је потврдио исту кардиналност као процењену (што је и логично, јер је кардиналност за број инстанци типа *ub:Department* тачна информација), али већ у другом кораку је дошло до одступања, добијена је 1 инстанца промењиве *Y* уместо 2. Заправо, процена је била 15*16/239=1,004 на основу почетних кардиналности, али пошто се користе само целобројне вредности за кардиналност, заокружено је на прву већу вредност. Ипак, већ при звездастом спајању по промењиви *Y* није било пресека, а процена је да је да ће се задржати исти број инстанци, поготово због малог улазног броја.

Крајњи резултат је да нема ниједне инстанце која задовољава дати упит што је у складу са очекивањем[47].

Треба приметити да су у овом случају операције спајања биле зависне у односу на претходну што не мора увек бити случај, поготово у случају већег графа. Карактеристична особина је да се независно могу урадити звездаста спајања за неколико чворова који су најселективнији, од којих даље креће ток извршавања операција спајања ка другим чворовима, а у неким чворовима упита долази до пресека скупа добијених инстанци по различитим путањама (наравно, мисли се у логичком смислу, а не физички), одакле се евентуално наставља ток извршавања задатака ради разрешења упита. Ови токови извршавања операција унутар истог упита су потпуно независни и носе са собом скуп релевенантих инстанци за коначан резултат упита, што је карактеристично за системе токова података (енг. *data flow systems*) [132]. С обзиром на то да нема бочних ефеката, токови операција извршавања упита могу читати структуре података без синхронизације између себе (али не и у односу на операције додавања нових *RDF* тројки) што омогућава паралелно извршавање код процесора са више језгара. У циљу максималног искоришћења појединих чворова, извршно окружење креира више програмских нити за извршење задатака извршавања упита, а искуство је показало да је *2*број_логичких_процесора* добар избор броја програмских нити, којима ће се распоређивати задаци упита.

4.4. Детаљи имплементације

Прототипска имплементација је урађена у програмском језику *Java* што са собом повлачи одређене консеквенце по питању детаља имплементације, што је тема овог одељка. Наиме, због великог броја *id*-јева у систему и структура индекса, потенцијално ће бити велики број креираних објеката што може да вишеструко повећа алоцирани простор у динамичкој зони меморије (енг. *heap*) у односу на потребни. Додатно се тиме оптерећује *скупљач ђубрета* (енг. *garbage collector*) што може доста да успори извршавање апликације. Стога је била

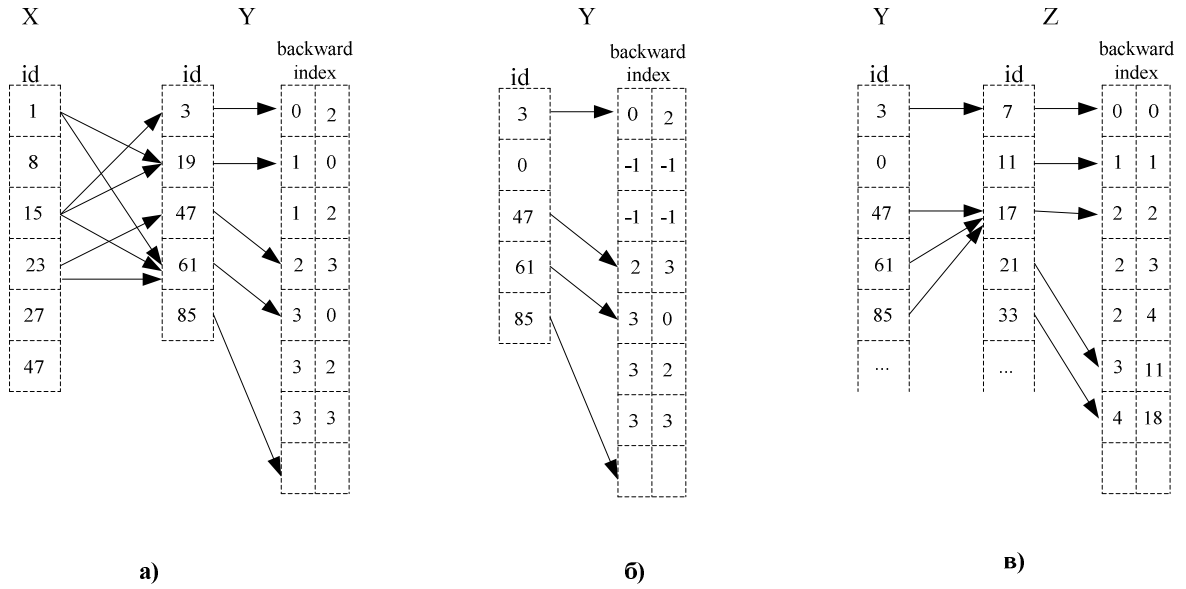
[47] http://swat.cse.lehigh.edu/projects/lubm/answers_query2.txt

тежња током имплементације да се где је то могуће користе низови уместо листи, а посебно у процесу извршавања упита, односно за чување међурезултата и мапирања између инстанци променљивих упита.

Индексне структуре су креиране да би ефикасно подржале извршавање операције упита, а као што је показано у претходном одељку, то су операције „звездастог спајања“ што се најефикасније реализује техником сортираног спајања сједињавањем (енг. *sort merge-join*) и мапираног спајања (енг. *hash join*). У првом случају ефикасност спајања се остварује сортирањем индекса односно у овом случају *id*-јева који представљају *URI*-је и литерале. Зато су *p->s* и *p->o* индекси реализовани као уређени низ целобројних *id*-јева коришћењем структуре стабла, која гарантује комплексност O (log n) за операције додавања нових елемената и проналажење постојећих. Већ је поменуто да се у случају звездастог спајања често користи и спајање са везаном променљивом која има кардиналност 1, а за то се користе индекси *ps->o* и *po->s*, па је и у том случају избор пао на структуру стабла за уређивање *id*-јева. Очигледно је да је максималан број елемената резултата спајања једнак кардиналности дате променљиве, што одговара броју елемената дате променљиве на позицији субјеката или објеката на узорку тројки са најмањом кардиналношћу. Тако се на почетку спајања креира (заправо копира листа из индексне структуре) низ елемената (у конкретној имплементацији 32-битних простих типова *int*) дужине минималне кардиналности за дату променљиву, а са сваким даљим спајањем резултатни елементи се преписују преко тог почетног низа.

Код операције мапирања ситуација је већ другачија, пошто се мора чувати веза ка изворним елементима преко којих је извршено пресликавање односно спајање по предикату. Додатно, након пресликавања може бити извршено звездасто спајање за неку променљиву, али у односу на ситуацију из претходног пасуса када је то била почетна операција у неком току операција извршавања упита, овде се мора очувати веза ка претходним елементима, а штавише омогућити да након извршеног новог мапирања, елементи друге променљиве имају конзистентну везу ка претходним елементима, односно још један корак уназад ка елементима са почетка процеса операција мапирања.

Уобичајена техника код спајања по предикату је мапирано спајање (енг. *hash-join*), али у овој имплементацији је коришћена посебна структура за чување резултата мапирања приказана на слици 29. Намера је да се одржи сортирани низ *id*-јева, али и веза ка претходним елементима преко којих је извршено мапирање. То се реализује преко показивача на претходне индексе и структуре претходног индекса (*backward index)* која чува редослед изворних (претходних) елемената који су пресликани.



а) **б)** **в)**

Слика 29 – Индекси структура података међурезултата извршавања упита

Операције над структуром међурезултата извршавања упита су следеће:

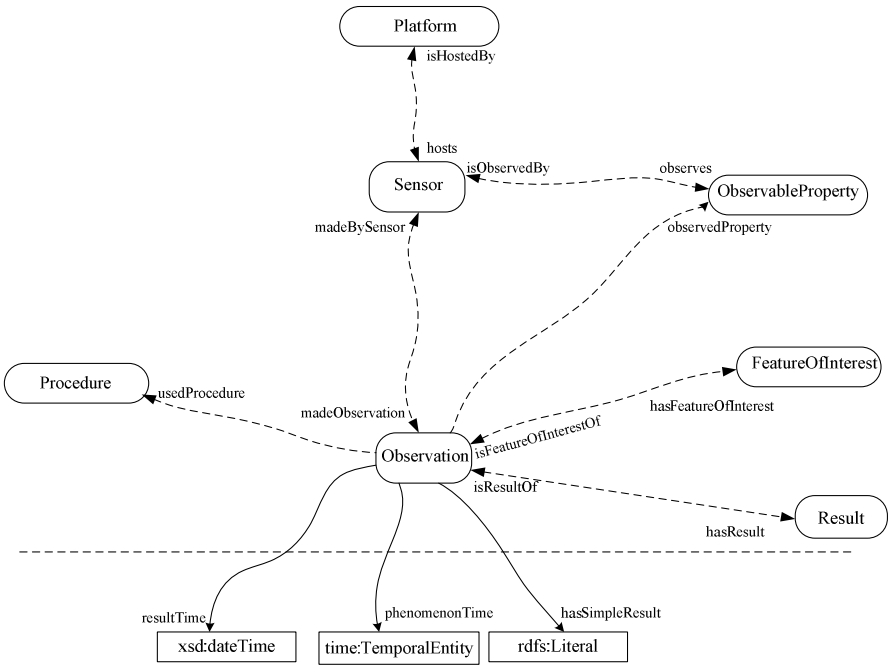
- Операција пресликавања (спајање између променљивих графа упита):
  - Додавање новог *id-а*: убацује се нови *id* у уређену листу *id*-јева. Редни број изворног елемента се додаје у интерну уређену листу мапираног елемента.
  - Додавање постојећег *id-а*: у интерној листи тог *id*-а додаје се редни број изворног елемента.
  - На крају операције мапирања, врши се „одмотавање“свих интерних редних бројева изворних *id*-јева, са којима се креира *backward index*, који чува мапирање уназад. За сваки *id* постоји

посебан показивач ка почетку мапираног индекса за тај *id* (слика 29а).

- Код сортираног спајања:
  - након извршеног спајања са инстанцама исте променљиве, избацивање елемената се врши уписивањем 0 у низу *id*-јева, а -1 на свим позицијама одговарајућег мапираног индекса (слика 29б).
- Ако се врши даље мапирање ка новој променљиви, понавља се процес (слика 29в).
- Коначно, на крају приликом креирања коначног резултата упита, врши се каскадно „одмотавање“, односно креирање редова резултата упита кроз угњеждено итерирање кроз структуру *претходног индекса*. Наиме, свака променљива ће на дати редни број индекса ако поседује *претходни индекс*, пронаћи сопствене редне бројеве својих повезаних *id*-јева. Ако је преко те променљиве извршено мапирање друге променљиве, даље ће бити прослеђени сопствени редни бројеви тој мапираној променљиви, како би вратила све мапиране елементе. Процес иде ланчано све док има мапираних променљивих.

4.5. Складиштење сензорских опажања, резултата мерења и временских и просторних вредности

Типична карактеристика онтологија за семантичке сензорске мреже је да се раздвајају статички сензорски подаци који описују сензоре и друге уређаје, од динамичких сензорских података који представљају сензорска опажања (енг. *sensor observations*). Приликом израде стандардизоване *W3C SSN* онтологије 2011. године, извршена је анализа 10 сензорских и 7 онтологија које се фокусирају на сензорска опажања [86] и резултати анализе су преточени у предлог тада нове сензорске онтологије. Ипак, крајем 2017. године се појавила ревизија стандардне *W3C SSN* онтологије која је знатно поједноставила неке аспекте око генерисања сензорских опажања [87]. Било је и других предлога нових једноставнијих сензорских онтологија попут IoT-Lite [133, 143], а такође у ЕУ пројекту *INTER-IoT* [144] извршена је анализа новијих сензорских онтологија.



Слика 30 – Секција сензорских опажања у SOSA-SSN онтологији (адаптирана слика из референце [87])

Недостатак првобитне верзије *W3C SSN* онтологије је потреба за генерисањем знатног броја *RDF* тројки ради исказивања нове вредности мерења сензорских опажања. То је довело до оптерећења и неефикасне обраде семантичких сензорских података, што је резултирало смањеном популарношћу за семантичке сензорске сервисе и платформе.

Секција *Observation* из *Sensor, Observation, Sample, and Actuator (SOSA)* [87] онтологије приказана на слици 30, као издвојеног дела ревизије *Semantic Sensor Network (SSN)* онтологије, садржи кључне класе за сензорска опажања и пратеће концепте. Циљ је да се идентификује најчешћи узорак креирања *RDF* тројки нових сензорских очитавања, што одређује и скуп узорака тројки код *SPARQL* упита у намери да се спајањем дохвате та сензорска очитавања. У складу са идентификованим узорком треба креирати одговарајуће индексне структуре које ће да омогуће ефикасно чување и претраживање датих података.

На основу дијаграма са слике 30, види се да се приликом сваког сензорског опажања креира нова инстанца класе *sosa:Observation* која може да реферише

сензор који је извршио мерење преко предиката *sosa:madeBySensor*, или релација креће од инстанце сензора преко *sosa:madeObservation* предиката ка новој инстанци сензорског опажања *sosa:Observation*. Резултат мерења се специфицира преко предиката *sosa:hasResult*, тј. инстанца класе *sosa:Observation* се повезује са инстанцом *sosa:Result* која даље специфира резултат мерења представљеног неком онтологијом која дефинише типове података и количине, као што је на пример *QUDT* онтологија [145], док се за представљање времена може користити онтологија за време тзв. *Time Ontology* [121]. Ово је промена у односу на спецификацију онтологије SSN из 2011. године, где је предвиђено да се вредности представљају са *DOLCE* онтологијом. Међутим, значајна промена у спецификацији *SSN* онтологије из 2017. године је краћи начин представљања резултата мерења преко предиката *sosa:hasSimpleResult* ка инстанци типа *rdfs:Literal* са кодованом вредношћу мерења унутар литерала, а слично и време када се десила процедура опажања наводи се преко предиката *sosa:ResultTime.* Рестрикције постављене у *OWL* дефиницијама над могућим вредностима објеката предиката гарантују да нема алтернативе по питању типа и кардиналности, односно да је једна инстанца *sosa:Observation* јединствено креирана од само једног *sosa:Sensor*. Очигледно је да веза од инстанце сензорског опажања до вредности мерења иде преко следећих предиката приказаних у табели 11:

Табела 11 – Предикати у SSN онтологији за представљање резултата и времена мерења

Вредност сензорског мерења	
Краћи начин	sosa:hasSimpleResults
Дужи Начин	sosa:hasResult::qudt-1-1:numericValue
Време извршења очитавања	
Краћи начин	sosa:resultTime
Дужи Начин	sosa:hasResult::time:inXSDDateTimeStamp

Семантички упити често захтевају податке у зависности од опсега измерених вредности сензора у неком временском интервалу. Стога је идеја да се путања од сензорских опажања ка измереним вредностима кодује са посебним вештачки додатим предикатом који би директно повезао сензорска опажања и

измерене вредности и кроз адекватне структуре убрзао извршавање упита, ако у онтологији нема предиката попут *sosa:hasSimpleResults* или *sosa:resultTime* или ако они нису коришћени за кодовање резултата опажања.

То је случај код сензорске онтологије *O&M-OWL*[48], која је коришћена за представљање сензорских опажања са око 20000 метеоролошких станица у Америци током трајања урагана као део пројекта MesoWest[49]. Са *O&M-OWL* онтологијом, за представљање вредности и времена мерења је потребно 10 *RDF* тројки. Таква семантичка представа заузима знатно већу дужину од вредности мерења изражене као реалан број (тип *float* заузима 32 бита у *Java* програмском језику) и времена мерења, које се може претворити у тип *long* (64 бита).

Додатна чињеница је да су *RDF* тројке везане за сензорска опажања (*sosa:SensorObservation*) углавном константне, тј. референцирају исти сензор (*sosa:Sensor*), процедуру (*sosa:Procedure*), посматрано својство (*sosa:ObservableProperty*) и карактеристику од интереса (*sosa:FeatureOfInterest*), а разликују се *RDF* тројке за представљање вредности мерења и времена очитавања. Из ове опсервације је дошла идеја да се раздвоји константан и променљив део у представљању сензорских опажања. Наиме, не би се чувале *RDF* тројке за свако ново сензорско опажање већ би се у фази препроцесирања нових *RDF* тројки оне филтрирале, а сачувала би се само једна инстанца сензорског опажања појединачног сензора, односно извршила би се редукција графа у делу сензорских опажања. Са друге стране, креирала би се инстанца класе *MerenjeUVremenu* (празан чвор – *blank node*) која је у вези са том јединственом инстанцом сензорског очитавања неког сензора. Даље, инстанца *MerenjeUVremenu* садржи својства вредност мерења и време мерења. Чињеница је да се *URI* инстанце *MerenjeUVremenu,* која би се генерисала за свако ново сензорско опажање, не мора чувати у *RDF* складишту јер се може премостити вештачким предикатом путање, тј. конкатенације предиката (код скупа

[48] http://wiki.knoesis.org/index.php/LinkedSensorData#Linked_Observation_Data
[49] http://mesowest.utah.edu/

података са *O&M-OWL*[50] онтологијом, то је имплементирано са вештачким предикатима *om-owl:result_xsd:float* и *om-owl:samplingTime_inXSDDateTime)*, али олакшава процес дистрибуције *RDF* тројки ка адекватној индексној структури. Ако евентуално дође до промене код константних вредности *RDF* тројки сензорских опажања неког сензора, сачуваће се та нова инстанца сензорског опажања, а даље ће се *RDF* тројке са временом и вредности мерења односити на последњу инстанцу сензорског опажања.

Дакле, циљ је доћи до индексне структуре са скупом вредности (*сензорско_опажање_URI, вредност_мерења, време_мерења*) уз подразумеване предикате између тих вредности. Сада се користе посебни индекси за претраживање распона вредности мерења и времена мерења. Користе се укупно четири индекса:

- *measuredValue:sensorObservationId,*
- *sensorObservationId:measuredValue,*
- *timeValue:sensorObservationId* и
- *sensorObservationId:timeValue.*

Имплементација ових индекса у реализованом прототипу је остварена преко класа које имплементирају интерфејс *NavigableSet* у *Java* програмском језику (класа *TreeSet* рецимо*)* и то преко компаратора композитних индекса у којем се врши поређење и сортирање индекса током операција додавања новог индекса или претраживања опсега вредности. Поређење почиње од првог параметра композитног индекса (рецимо код првог индекса то је *measuredValue)*, па ако су они једнаки, прелази се на следећи (у овом случају *sensorObservationId)* итд.

У случају коришћења дистрибуираних складишта за чување индекса, поменуте индексе је могуће реализовати ако постоји лексикографско уређење кључева, који се посматрају као низ бајтова, а претрагом неког кључа, ако се он

[50] http://knoesis.wright.edu/research/semsci/application_domain/sem_sensor/ont/sensor-observation.owl

не пронађе, позиционирање се врши на први мањи постојећи кључ. Такав приступ повлачи за собом потребу да се за децималне бројеве користи представа са фиксном децималном тачком, ширине целобројног и децималног дела у зависности од потребне тачности (на пример 32+32 бита), а време се стандардно представља као 64-битни (*long)* запис. Чак и ако индекс није довољно прецизан да представи тачну вредност мерења, он се пресликава у улаз који чува листу тачних (прецизних) вредности, па се може даље наставити претрага проласком кроз листу. Овај приступ је могуће користити са решењем *Redis*[51], *Apache HBase* (тзв. *scan* оператор) [141], *LevelDB*[52] итд. Последња два решења подразумевано смештају податке на диск, али је могуће да се „монтирањем"(енг. *mounting*) *tmpfs*[53] фајл система у меморији (подржан у кернелу Linux оперативног система од верзије 2.4) оствари чување података у меморији, иако на индиректан начин (пример у референци [146]).

Идеја је следећа: у случају претраге опсега вредности (или конкретне вредности мерења), користе се кључеви *donjaVrednostMerenja:0* и *gornjaVrednostMerenja:MAX_INT*. Резултат ће бити листа композитних индекса који имају вредност мерења у задатом опсегу за разна сензорска опажања (ради једноставности излагања, није наведено да је један од параметара у композитном индексу и јединица мере). Време када је извршено сензорско опажање се у прототипској имплементацији чува као атрибут објекта композитног индекса, а алтернативно се може чувати као вредност пресликана у табели за дати кључ. Ако се тражи опсег мерених вредности конкретног сензора, то се постиже претрагом друге структуре индекса са кључевима:

- *sensorObservationId:donjaVrednostMerenja* и
- *sensorObservationId:gornjaVrednostMerenja*,

пошто су сензор и сензорско опажање у вези 1-1, а могуће је и за случај 1-*n*, где је *n* релативно мали број (то се користи ако памтимо нову инстанцу сензорског опажања у случају да је дошло до неке промене у референцирању

[51] https://redis.io/topics/indexes
[52] http://leveldb.org/
[53] https://en.wikipedia.org/wiki/Tmpfs

константног дела сензорског опажања). Потпуно аналогно се претражује интервал времена само са другом индексном структуром, а пресликана вредност у табелама је у том случају резултат мерења сензора. Кључеви би били:

- *donjaVrednostVremena:0* и *gornjaVrednostVremena: MAX_INT,*
- *sensorObservationId:donjaVrednostVremena* и *sensorObservationId:gornjaVrednostVremena.*

Просторне координате се чувају као две (латитуда и лонгитуда) или три координате (додатно алтитуда). Као први корак код просторних координата је пресликавање из вишедимензионалног у 1-димензионални простор, али уз захтев да се задржи близина тачке у представи 1-димензионалног простора. За ту намену се користи пресликавање преко просторних Z-1 кривих као што су Хилбертова трансформација [147] или *GeoHash* кодирање[54]. Даље се добијени индекс чува у структурама за просторне координате као што су *QuadTree* и *R-Tree* и њихове варијанте. У овој реализацији је коришћено *GeoHash* кодирање са варијантом *QuadTree* индексне структуре за претрагу.

4.6. Евалуација перформанси реализованог RDF складишта

За тестирање перформанси реализованог *RDF* складишта и архитектуралног приступа коришћен је већ поменути скуп сензорских мерења из пројекта *MesoWest* са подацима из метеоролошких станица у САД за време неколико урагана у периоду од 2003. до 2009. године у временском периоду од око 5 до 10 дана. *RDF* подаци описа сензора су одвојени од сензорских опажања. Подаци садрже мерења сензора температуре, влаге, брзине ветра, падавина итд. У овом тестирању, изабрани су подаци за ураган Чарли, који се десио од 5 до 10 августа 2004 године и скуп садржи 101.956.760 *RDF* тројки односно 9.333.676 сензорских опажања.

Како би се симулирала ситуација у реалном раду, тест случај предвиђа најпре учитавање основних семантичких описа сензора са метеоролошких станица, а

[54] https://en.wikipedia.org/wiki/Geohash

затим учитавање сензорских опажања за три дана сензорских опажања. Да би се тестирале перформансе *RDF* складишта за различита оптерећења, у тестовима је након почетног уноса сензорских опажања за три дана, настављено са уносом преосталих сензорских опажања (око два дана) у реалном времену али са знантно краћим временским размаком него што су реална мерења вршена, односно свако ново сензорско опажање је додавано након 100ms, али тако да се сва сензорска опажања са одабраног броја сензорских станица унесу за укупно 6s, униформно распоређена у том периоду. Мерења су вршена за сензорска опажања са 1000 до 6000 станица (подаци за ураган Чарли постоје за нешто више од 7700 метеоролошких станица) у корацима од по 1000 станица, а број сензора по станици није константан и просечно има 6-8 сензора попут сензора температуре, брзине ветра, влажности, смера дувања ветра, детектора падавина, детектора падања кише итд. Такође, сви сензори не генеришу вредности у истим временским размацима.

За тестирање су коришћена три *SPARQL* упита. Први упит селектује сензоре који су измерили вредност мању од 65° фаренхајта (18.33° целзијуса) од 10 до 12 часова 10. августа 2004. године. Сличан упит је коришћен и код других аутора [47][148] који су користили исти скуп података са *Virtuoso RDF* складиштем, само за други временски период. Први *SPARQL* упит гласи:

```
prefix om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
prefix sens-obs:<http://knoesis.wright.edu/ssw/>
prefix xsd:<http://www.w3.org/2001/XMLSchema#>
prefix owl-time:<http://www.w3.org/2006/time#>

SELECT DISTINCT ?sensor
WHERE {?sensor om-owl:generatedObservation ?observation .
       ?observation rdf:type weather:TemperatureObservation .
       ?observation om-owl:samplingTime ?time .
       ?time owl-time:inXSDDateTime ?xsdtime .
       ?observation om-owl:result ?result .
       ?result om-owl:floatValue ?value .
       ?result om-owl:uom weather:fahrenheit .

        FILTER(?value <= "65.0"^^xsd:float)
        FILTER(?xsdtime >= "2004-08-10T10:00:00-
07:00^^http://www.w3.org/2001/XMLSchema#dateTime")
```

```
       FILTER(?xsdtime <= "2004-08-10T12:00:00-
07:00^^http://www.w3.org/2001/XMLSchema#dateTime") }
```

У прилогу су дата остала два упита. Други упит је мање селективан и односи се само на сензоре који су измерили вредност изнад 75° фаренхајта (23° целзијуса) уопште, а трећи упит је просторно-временски-вредносни упит, а врши претрагу сензора који су измерили температуру испод 65° фаренхајта (18,33° целзијуса) у периоду од 8 часова увече до 8 часова ујутру 8. и 9. августа 2004. године, апроксимативно на простору америчке државе Аризоне. У питању је упит сличан првом упиту, само што је више селективан додавањем филтрације по просторној координати сензора.

Упити се шаљу паралелно са уносом нових сензорских опажања, али са случајним временским размаком између упита, за разлику од константног временског размака између уноса *RDF* тројки. Мери се просечно време извршавања стандардног упита за дати тестни скуп, у конфигурацији прво са централизованим *RDF* складиштем (инсталиран само на једном рачунару), дакле у централизованој организацији, а у другом случају у дистрибуираној конфигурацији са *master* чвором и два *slave* чвора тако да један чвор чува вишедимензионалне временско-вредносне индексе за сензорска опажања, а други просторне координате и све остале *RDF* тројке. Резултати су упоређивани са јавно доступним *RDF* складиштем *Virtuoso 7.2.4.2*, које је већ коришћено у неким архитектурама за складиштење сензорских података у реалном времену [109][111][146]. Проблем код других *RDF* складишта је што она нису предвиђена за рад са подацима у реалном времену, већ се код њих подаци уносе на почетку и том приликом се креирају адекватне индексне структуре. Конкретан пример је складиште *RDF-3X* [149] код кога постоји функција учитавања целокупног скупа података и том приликом се креира статистика дистрибуираности *RDF* тројки. Додавањем нових *RDF* тројки креирају се помоћне индексне структуре, како би се кроз функцију реорганизације опет креирале јединствене структуре индекса и статистике. Јасно је да је основна намена овог складишта за скупове података који се веома ретко мењају, пошто би у случају рада са подацима у реалном времену

долазило до веома честе реорганизације целокупне индексне структуре, а то је веома скупа операција за велику количину података.

Треба напоменути да онтологија која је коришћена за овај скуп података, *O&M-OWL*, предвиђа да се локација дефинише за сензор, што подразумева статичке сензоре, за разлику од *SSN* где се локација може повезати за сензорским опажањем, односно да се дефинише локација за конкретно сензорско мерење чиме се подржавају мобилни сензори. Последица приступа у онтологији *O&M-OWL* је да има релативно мали број *RDF* тројки за просторне координате, које одговарају броју сензора, тако да то није велики број да би се дистрибуирао на посебан чвор, док би у случају *SSN* онтологије постојала потреба за дистрибуирањем тих података.

Резултати мерења извршавања датих упита су дати у табелама 13-15 и на сликама 31-33. Најпре ће бити дати преглед броја унетих *RDF* тројки у сваком мерењу и такође времена учитавања почетног скупа сензорских података (табела 12). Приметно је да су ова времена доста нижа код централизоване организације, што значи да операција додавања нових тројки није процесорски захтевна, тако да се постиже уштеда времена због неопходног времена преноса података до *slave* компоненти у случају дистрибуиране варијанте. Са друге стране, времена учитавања почетног скупа код *Virtuoso* складишта су за ред величине већа, од 8 до 15 пута. То је првенствено последица складиштења података на диск, али већ овај податак имплицитно говори о ограничењу складишта *Virtuoso* за рад са подацима у реалном времену јер они захтевају кратко време уноса у складиште.

Унос нових *RDF* тројки и слање упита се врши у језику *SPARQL* преко *HTTP* захтева методом *POST*. За успостављање *HTTP* конекције потребно је време реда величине 100ms услед размене заглавља као део *HTTP* протокола, осим ако се користе већ постојеће конекције, што се дешава када се захтеви шаљу у низу и тада време извршења упита на клијенту постаје скоро идентично времену измереном на серверу (пар милисекунди је разлика у том случају).

Због тога су у свим табелама код резултата мерења упита, дата одвојено времена извршавања упита измерена на серверу, односно на *master* чвору и клијентском рачунару који је послао упит из *LAN* окружења. Ради упоређивања перформанси са другим системима, коректно је дати време извршавања мерено на страни клијента, али ради процене оптерећености система, време измерено на серверу је корисније, јер говори колико је реално заузеће *RDF* складишта по једном упиту.

Табела 12 – Времена учитавања инцијалног скупа сензорских података (за прва три дана из тестног скупа)

Број станица	Број RDF Тројки	Време инцијалног учитавања почетног скуп података [s]		
		Централизована организација	Дистрибуирана организација	Virtuoso
1000	15,93M	250,13	342,28	2703,23
2000	30,19M	476,40	631,18	5706,80
3000	37,37M	544,14	769,21	7535,21
4000	48,27M	712,00	813,13	10.657,90
5000	58,52M	983,69	1218,66	X
6000	68,33M	1350,00	1468,73	X

Анализом резултата упита, можемо закључити да су прилично слична времена извршавања упита код централизоване и дистрибуиране организације, али ипак је код дистрибуиране организације време краће, поготово у случају података са већег броја станица. Објашњење лежи у чињеници да у централизованој организацији, са рачунаром са 16GB оперативне меморије и процесором Intel i5 3.2GHz са 4 логичка процесора (2 језгра), не долази до претераног оптерећења процесора услед техника редукције графа која је примењена за редукцију броја *RDF* тројки сензорских опажања које се складиште у меморији. Такође у централизованој варијанти, *master* чвор има довољно меморије за све пратеће структуре (за речничко кодирање пре свега) као и за чување *RDF* података. Ако би се ограничило

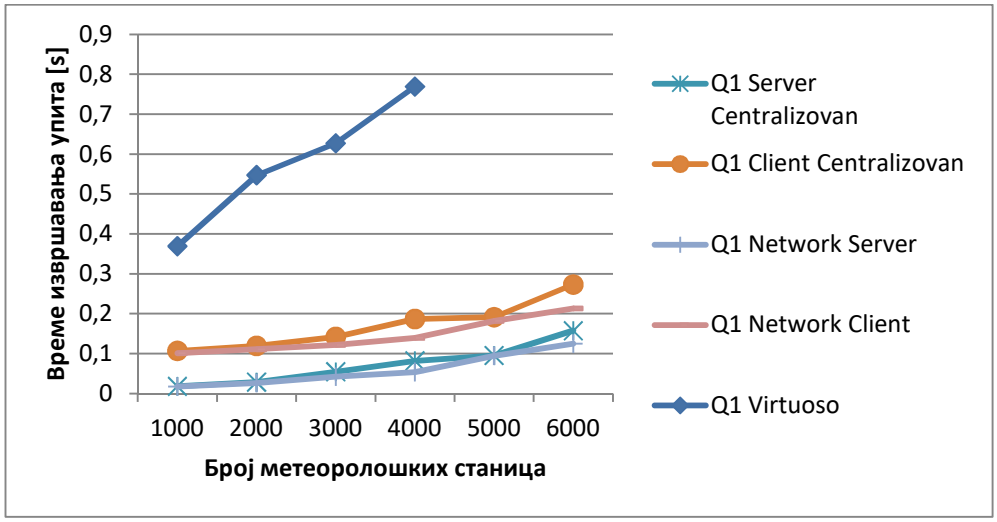
коришћење меморије *RDF* складишту (кроз ограничење дозвољене величине хипа код *JVM*), долази до драматичног повећања времена извршавања. Основни закључак је да дистрибуирана организација долази пре свега до изражаја код веома велике количине података.

Код свих упита, са порастом количине података, односно повећањем броја сензора, долази до повећања времена одзива. То је разумљиво јер структуре података за чување *RDF* тројки садрже знатно већи број целобројних вредности, тј. идентификатора у случају складишта *RDF* тројки обичних предиката, а такође и у структурама вишеструког индекса код складишта временско-вредносних података.

Најбрже време извршавања упита је код упита бр. 1, пошто је генерално селективан у односу на вредност мерења, време мерења и тип сензора, за разлику од упита бр. 2 који врши само филтрирање по вредноси мерења и типу сензора. Занимљиво је да је број међурезултата из временско-вредносног складишта сличан код упита бр. 1 и код упита бр. 2, чак је и мало већи код упита бр. 1 (за случај 3000 станица, око 8190 међурезултата код упита број 1, према 7600 код упита број 2) и очигледно је да се то односи на број сензорских опажања, али да је коначан број сензора, као резултат спајања са свим температурним сензорима из подразумеваног складишта ипак мањи, што доприноси краћем времену. Такође, упит 3 је сличан упиту 1, уз додатак просторног филтрирања сензора, али ипак се врши нешто већи број спајања због услова са просторним подацима (број међурезултата је исти код временско-вредносног складишта као за случај упита 1). Овде није извршено генерисање композитних временско-просторно-вредносних индекса, из разлога што су просторне координате дате само за сензоре, а не и за сензорска опажања па би се добио непотребно велики број понављајућих вредности. Такође, број међурезултата се веома споро мења са додавањем нових тројки, без обзира што је у питању интензивно додавање, што је и логично, јер мали број нових тројки задовољава услов упита.

Табела 13 – Времена извршавања упита бр. 1 – временско-вредносни упит (* види текст)

SPARQL упит #1	Централизована организација		Дистрибуирана организација		Virtuoso
Број станица	Сервер [s]	Клијент [s]	Сервер [s]	Клијент [s]	
1000	0,018	0,107	0,017	0,101	0,369
2000	0,028	0,119	0,026	0,111	0,547*
3000	0,055	0,142	0,042	0,121	0,627*
4000	0,082	0,187	0,054	0,139	0,769*
5000	0,095	0,191	0,095	0,182	X
6000	0,157	0,273	0,125	0,213	X



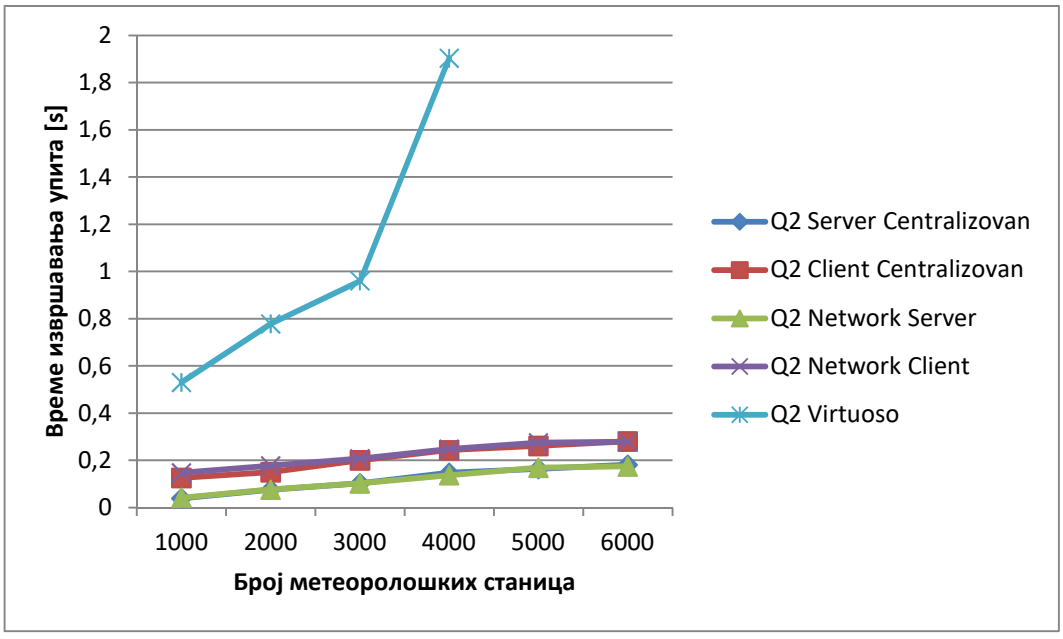
Слика 31 – Упоредни приказ резулата мерења извршавања упита бр. 1

Као генерални закључак, очигледно је да са порастом количине генерисаних сензорских опажања долази до изражаја дистрибуираност структуре индекса и самим тим паралелно извршавање упита постаје ефикасније, али разлика није претерано велика. То наводи на закључак да у случају семантичких скупова сензорских података који немају огромну количину података, архитектура са централизованим *RDF* складиштем и једним *master* чвором је сасвим довољна. Такође, додатно објашњење лежи у чињеници да су

данашњи микропроцесори реализовани са више језгара, а очекивања су да ће у будућности са преласком на 8nm технологију број језгара на једном физичком процесору ићи и до хиљаду [132], што оставља доста простора за искоришћавање паралелизма током извршавања *SPARQL* упита. Ипак, намећу се ограничења лимитираног пропусног опсега меморије, када предност има дистрибуирана организација *RDF* складишта.

Табела 14 - Времена извршавања упита бр. 2 –вредносни упит (* види текст)

SPARQL упит #2	Централизована организација		Дистрибуирана организација		Virtuoso
Број станица	Сервер [s]	Клијент [s]	Сервер [s]	Клијент [s]	
1000	0,039	0,126	0,042	0,147	0,530
2000	0,076	0,150	0,076	0,177	0,778*
3000	0,103	0,200	0,103	0,208	0,964*
4000	0,147	0,243	0,137	0,248	1,903*
5000	0,164	0,261	0,169	0,274	X
6000	0,181	0,280	0,174	0,279	X



Слика 32 - Упоредни приказ резултата мерења извршавања упита бр. 2

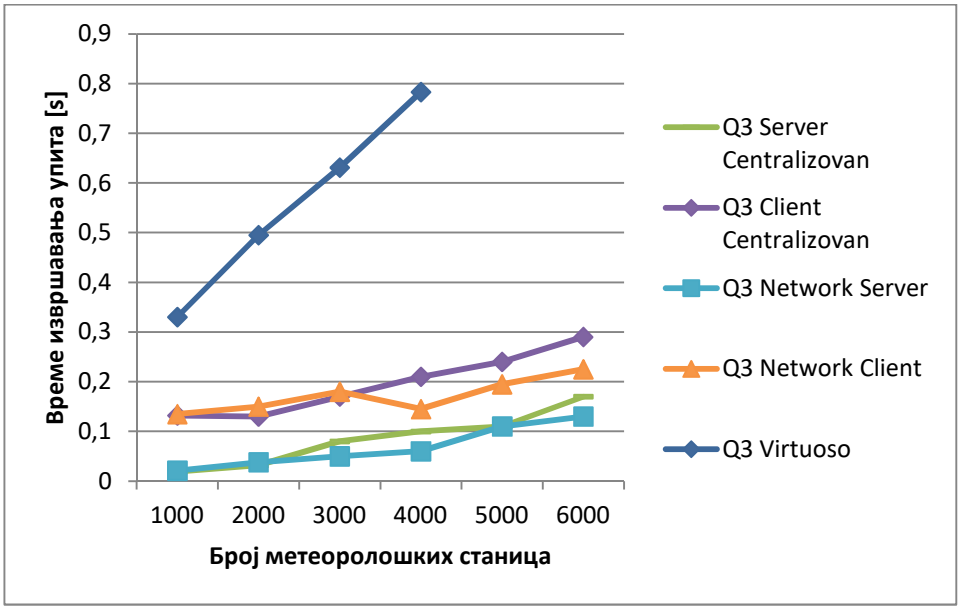
Из табела се види да је резултат извршавања упита на серверу увек испод 100ms осим у случају 6000 сензорских станица, што говори да је у питању веома употребљив систем за складиштење и манипулацију семантичким сензорским подацима у реалном времену. Тестирање је вршено паралелно са веома интензивним додавањем нових сензорских опажања, што је у пракси ретко, иако број сензора може потенцијално бити много већи, што доводи до закључка да је волумен података мање-више сличан.

Табела 15 - Времена извршавања упита бр. 3 – просторно-временски-тематски упит

SPARQL упит #3	Централизована организација		Дистрибуирана организација		Virtuoso
Број станица	Сервер [s]	Клијент [s]	Сервер [s]	Клијент [s]	
1000	0,019	0,132	0,021	0,135	0,330
2000	0,032	0,130	0,038	0,150	0,495*
3000	0,080	0,170	0,050	0,180	0,631*
4000	0,100	0,210	0,060	0,145	0,783*
5000	0,110	0,240	0,110	0,195	X
6000	0,170	0,290	0,130	0,225	X

Код *Virtuoso RDF* складишта ситуација је веома специфична. Већ приликом мерења за 2000 станица, време извршавања упита се кретало од 5 до 15 секунди, што је последица чињенице да је складиште било преоптерећено конкурентним уносом нових *RDF* тројки, па је долазило до нагомилавања захтева за уносом, и последично до веома дугог чекања на извршавање упита. Приказана времена извршавања упита се односе на ситуацију када је заустављен унос нових *RDF* тројки и претрага се вршила само над већ унетим сензорским подацима. Чак и у том случају, времена извршавања упита су била од 3 до 5 пута већа у односу на реализовано *RDF* складиште. Овакви резултати оправдавају почетне хипотезе и донете пројектантске одлуке ради

имплементације *RDF* складишта за рад са сензорским подацима у реалном времену. Такође, потврђена су искуства аутора *LSM* платформе [109] код којих је након интензивног уноса сензорских *RDF* тројки долазило до драстичног повећања времена одзива *Virtuoso RDF* складишта и његове неупотребљивости за дати случај употребе.



Слика 33 - Упоредни приказ резулата мерења извршавања упита бр. 3

Расположиви рачунарски ресурси нису омогућили да се истестирају крајње границе датог *RDF* складишта, јер су измерене перформансе биле на одличном нивоу у условима максималне оптерећености, што доводи до закључка о применљивости датог *RDF* складишта у архитектури интеграције сензорских мрежа са потенцијалном употребом за читав низ апкикативних домена. Свакако тема за даљи рад је наставак евалуације његових перформанси са још већим обимом семантичких сензорских података и у рачунарском окружењу већих расположивих процесорских, меморијских и комуникационих ресурса.

4.7. Сродни радови

Област имплементације *RDF* складишта је дужи низ година у фокусу научно-истраживачке заједнице. Потребе за ефикасном обрадом, складиштењем и манипулацијом *RDF* скупа података који прати графовски модел, довеле су до константног побољшавања ових система и примене потпуно различитих техника.

Најстарија решења попут *Sesame* [150] и *JENA* [151] ослањала су се на релационе базе података, смештајући *RDF* тројке најпре у једној великој табели, по принципу једна *RDF* тројка у једном реду, што је у случају сложених упита и вишеструких спајања било крајње неефикасно. *Oracle* [152] у решењу из 2005. године, групише субјекте и објекте по сродним предикатима у једној табели, бирајући предикате тако да се минимизује потреба за спајањем, односно омогућавајући да се упити изврше над једном табелом. Сличан проблем је и у предложеној архитектури у овом раду, односно како изабрати сродне предикате за потребе дистрибуције индекса на одређеном рачунарском чвору.

Abadi и други аутори [153] су направили знатан помак у арени *RDF* складишта са решењем заснованим на вертикалном партиционисању, односно смештању *RDF* тројки на основу предиката, где се парови субјеката и објеката за одређени предикат смештају у посебну табелу која одговара одређеном предикату. Решење се ослањало на колонске базе података, што су касније примењивали и други аутори, у решењима са вишеструким индексима, а најчешће имплементацијом са 6 пермутација индекса као код решења *RDF-3X* [149] и *Hexastore* [154]. *RDF-3X* је нарочито ефикасно решење и имплементира специфично чување компресованих индекса са кодовањем разлике између *id*-јева сортираних тројки.

У питању су све била централизована решења, а са појавом дистрибуираних платформи попут *Apache Hadoop*[55], појављују се дистрибуирана решења попут *H-RDF-3X* [155] које распоређује инстанце централизованог решења *RDF-3X* у

[55] http://hadoop.apache.org/

кластеру, а комуникација између инстанци се врши преко *Hadoop* платформе. Било је и других покушаја коришћења дистрибуираних *NoSQL* решења за *RDF* складишта попут *H2RDF* [156].

Следећи напредак је дошао кроз иновативне технике складиштења *RDF* тројки, тако да се у решењу *Trinity.RDF* компаније *Microsoft* [146] упити разрешавају техником откривања подграфа кретањем кроз дистрибуирани граф. Ово је прво решење које користи меморијски облак. Веома сличан приступ је коришћен и у решењу компаније IBM [157] само са реализацијом преко *RDBMS*. Решења из других праваца истраживања користе бит-матрице за представљање *RDF* тројки попут *BitMap* [158] и *TripleBit* [159] остварујући веома ефикасно спајање у случају великих *RDF* скупова података.

И коначно наступило је време изразито дистрибуираних решења која партиционишу *RDF* скуп података по дистрибуираним чворовима, најчешће користећи посебне технике мапирања субјеката. Таква су решења *TriAD* [160], *DREAM* [161], *S2RDF* [162] и *AdPart* [163], који базирају своју имплементацију на *MPI (Message Passing Interface)* стандарду. *DREAM* [161] врши потпуну репликацију *RDF* скупа података на чворове у кластеру, тако да се извршавање *SPARQL* упита балансира између чворова у складу са могућношћу паралелног извршавања упита. *S2RDF* [162] користи *Apache Spark*[56] платформу која податке чува у меморији и такође примењује ефикасне методе чувања структура објеката с обзиром да се извршава на *Java* виртуелној машини. *AdPart* [163] упошљава адаптивно партиционисање *RDF* скупа података у дистрибуираном окружењу.

И на крају вреди истаћи најновије покушаје процене кардиналности конјуктивних упита над *RDF* подацима, који се базирају на техници сумаризације графа [164].

[56] https://spark.apache.org/

6 Закључак

Област сензорских мрежа односно визија *Интернета Ствари* (енг. *Internet of Things - IoT)* сваким даном постиже нове напретке и генерише нове сервисе и производе. Ипак, са убрзаним развојем ове области, многи технолошки аспекти остају недовољно усавршени, или технологија још није на нивоу да реализује одређене визије производа и услуга ка корисницима.

Један од уочених проблема у области сензорских или IoT апликација је и даље доминација тзв. вертикалних решења, што подразумева да се сензори распоређени у оквиру једне сензорске мреже користе за предвиђени апликативни домен и та сензорска мрежа остаје изолована јер нема размене информација са другим мрежама или корисницима.

Фокус овог рада је анализа проблема интеграције сензорских мрежа како би се омогућило хоризонтално уједињавање сензорских мрежа повезаних на Интернет у циљу пружања корисницима могућности дохватања свих расположивих сензорских података и тиме креирање разних интелигентних сервиса. Фокус је на семантички заснованој интеграцији сензорских мрежа у ком случају се користе семантичке веб технологије за остваривање интероперабилности кроз описе сензора и представљање сензорских опажања.

Доприноси овог рада су следећи:

- Класификоване су и анализиране генеричке архитектуре за интеграцију сензорских мрежа у распону од малог, до великом степена скалабилности.
- За архитектуре са пролазним уређајем и брокером порука реализовано је симулационо окружење ради анализе перфомансi у односу на

кључне пројектантске параметре попут протокола и формата порука за размену. Дата је свеобухватна евалуација и упоредни приказ перформанси постојећих типова архитектура кроз поређење IoT апликативних протокола и других комуникационих протокола и формата порука, што омогућава лакши одабир приступа и протокола приликом пројектовања IoT апликација у односу на функционалне захтеве који се желе постићи.

- Анализирана су постојећа семантички базирана решења за интеграцију сензорских мрежа и дата је њихова класификација идентификовањем кључних пројектантских приступа. За сваки приступ је дата анализа предности и недостатака.
- Анализом кључних недостатака архитектура за интеграцију сензорских мрежа на бази семантике података, предложена је нова архитектура која се заснива на дистрибуираном *RDF* складишту високих перформанси у циљу подршке ефикасног управљања сензорским подацима у реалном времену, проналажења жељених вредности, а такође и њиховог чувања за каснију анализу применом метода машинског учења итд.
- У предложеном *RDF* складишту је дат низ пројектантских решења како би се остварile жељене перформансе, а нарочито подршка за интензивно генерисање нових семантичких сензорских података и претрагу резултата мерења уз додатне временско-просторне услове. То се односи на начин креирања вишеструких индекса, стратегију дистрибуције *RDF* тројки по рачунарским чворовима, план извршавања *SPARQL* упита, начин одржавања индекса за резултате мерења, временске и просторне податке, редукцију графа сензорских опажања ради ефикаснијег чувања и извршавања упита над њима.

Аутор овог рада је свестан могућности даљег унапређења предложене архитектуре. Неки од могућих праваца даљег истраживања су:

- Боља интеграција *RDF* складишта са процесорима континуалних упита ради проширења временског прозора у којем је могуће обрадити и филтрирати нове генерисане *RDF* тројке.
- Разматрање примене семантичких технологија код архитектуре са брокером порука односно код система за пренос порука по принципу објави-претплати.
- Коришћење *dataflow* хардверских система у циљу побољшања перформанси *RDF* складишта са подацима смештеним у оперативној меморији.
- Примена генерализоване технике редукције графа сензорских опажања и алгоритма плана извршавања *SPARQL* упита у другим областима обраде графовски организованих података, посебно у извршавању на процесорима са више језгара.

С обзиром да се многе прогнозе слажу око масовног коришћења IoT сервиса у условима веома интензивног ширења броја инсталираних сензора и сензорских апликација у нашем окружењу у будућности, аутор овог рада се нада да ће и овај рад допринети технолошком напретку у области IoT, чиме се остварују потенцијалне могућности побољшања услова живота у разним аспектима. Такође, аутор ће свакако наставити научно-истраживачки рад у овој области с циљем креирања још ефикаснијих архитектура.

7 Литература

[1] M. Weiser, "The Computer for the 21st Century," *Scientific American,* vol. 265(9), pp. 66–75, 1991.

[2] N. Gershenfeld, R. Krikorian, and D. Cohen, "The Internet of Things," *Scientific American*, vol. 291(4), pp. 76-81, October 2004.

[3] A. Sheth, P. Anantharam, and C. Henso, "Physical-Cyber-Social Computing: An Early 21st Century Approach," *IEEE Intelligent Systems*, vol. 28(1), pp. 78-82, 2013.

[4] Cisco, Internet of Everything, https://newsroom.cisco.com/ioe, приступ 01.04.2018.

[5] K. Delin, S. Jackson,"The SensorWeb: a new instrument concept," *Proceedings of the SPIE International of Optical Engineering*, vol. 4284, pp. 1–9, 2001.

[6] A. Sheth, C. Henson, S. Sahoo,"Semantic Sensor Web", *IEEE Internet Computing*, vol. 12(4), pp. 78–83, 2008.

[7] O. Corcho, R. Garcia-Castro, "Five Challenges for the Semantic Sensor Web," *Semantic Web Journal*, vol. 1(1-2), pp. 121–125, 2010.

[8] P. Barnaghi, A. Sheth, C. Henson, "From Data to Actionable Knowledge: Big Data Challenges in the Web of Things," *IEEE Intelligent Systems*, vol. 28(6), pp. 6-11, Nov-Dec. 2013.

[9] The Web Ontology Language (OWL), http://www.w3.org/TR/owl-ref/, приступ 01.04.2018.

[10] Resource Description Framework (RDF), https://www.w3.org/RDF/, приступ 01.04.2018.

[11] M. Compton, H. Neuhaus, K. Taylor, K.-N. Tran, "A Survey of the Semantic Specification of Sensors," in: *Proceedings of the Second International Workshop on Semantic Sensor Networks (SSN09),* Washington DC, USA, October 26, 2009.

[12] M. Compton et al. "The SSN Ontology of the W3C Semantic Sensor Network Incubator Group", *Journal of Web Semantics*, vol. 17(c), pp. 25-32, 2012.

[13] R. v. d. Meulen, "Gartner Says 6.4 Billion Connected "Things" Will Be in Use in 2016, Up 30 Percent From 2015,"

http://www.gartner.com/newsroom/id/3165317, November 2015, приступ 01.04.2018.

[14] Ericsson Mobility Report,

http://www.ericsson.com/res/docs/2015/mobility-report/ericsson-mobility-report-nov-2015.pdf, November 2015, приступ 01.04.2018.

[15] IDC, "THE DIGITAL UNIVERSE IN 2020:Big Data, Bigger Digital Shadows, and Biggest Growth in the Far East,

https://www.emc.com/collateral/analyst-reports/idc-the-digital-universe-in-2020.pdf, Dec. 2012, приступ 01.04.2018.

[16] Matt Asay, "The Internet of Things Will Need Millions of Developers By 2020," http://readwrite.com/2014/06/27/internet-of-things-developers-jobs-opportunity, June 2014, приступ 01.04.2018.

[17] Research & Innovation in Internet of Things, https://ec.europa.eu/digital-single-market/en/research-innovation-iot, приступ 01.04.2018.

[18] I.F. Akyildiz, W. Su*, Y. Sankarasubramaniam, E. Cayirci, "Wireless sensor networks: a survey," *Computer Networks*, vol. 38, pp. 393-422, 2002.

[19] Culler, D., E., Estrin, D., Mani B. Srivastava, M., B., "Guest Editors' Introduction: Overview of Sensor Networks," *IEEE Computer* 37(8): pp 41-49, 2004.

[20] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A survey," *Computer Networks*, vol. 54(15), pp. 2787-2805, Oct. 2010.

[21] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, "Internet of Things: Vision, applications and research challenges," *Ad Hoc Networks*, vol. 10(no. 7), pp. 1497-1516, Sep. 2012.

[22] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Communications Surveys Tuts.*, vol. 17(4), pp. 2347-2376, 4th Quart., 2015.

[23] Gavrilovska, Lj., Krco, S., Milutinovic, V., Stojmenovic, I., Trobec, R. (eds), "Application and Multidisciplinary Aspects of Wireless Sensor Networks," *Springer*, 2011.

[24] Hill, J., L., Szewczyk, R., Woo, A., Hollar, S., Culler, D., E., Pister, K., S., J., "System Architecture Directions for Networked Sensors, " In *Architectural Support For Programming Languages and Operating Systems* (ASPLOS) 2000, pp 93-104, 2000.

[25] IEEE Standard for Local and Metropolitan Area Networks—Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs), IEEE Std. 802. 15. 4-2011, 2011.

[26] Howitt, I., Gutierrez, J., A., "IEEE 802.15.4 low rate-wireless personal area network coexistence issues," *Wireless Communications and Networking* 3 (2003), pp. 1481–1486, 2003.

[27] N. Kushalnagar, G. Montenegro, C. Schumacher, IPv6 Over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals, IETF RFC 4919, August 2007.

[28] E. Ferro and F. Potorti, "Bluetooth and Wi-Fi wireless protocols: A survey and a comparison," *IEEE Wireless Commun.*, vol. 12(1), pp. 12–26, Feb. 2005.

[29] P. McDermott-Wells, "What is Bluetooth?" *IEEE Potentials*, vol. 23(5), pp. 33–35, Jan. 2005.

[30] T. Lennvall, S. Svensson, and F. Hekland, "A comparison of WirelessHART and ZigBee for industrial applications," in *Proc. IEEE International Workshop on Factory Communication Systems (WFCS),* pp. 85 –88., May 2008.

[31] TinyOS Tutorials, http://tinyos.stanford.edu/tinyos-wiki/index.php/Main_Page, приступ 01.04.2018.

[32] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki—A lightweight and flexible operating system for tiny networked sensors," in *Proc. 29th Annu. IEEE Int. Conf. Local Comput. Netw.*, pp. 455–462, 2004.

[33] M. Botts, G. Percivall, C. Reed, J. Davidson, "Sensor Web Enablement: Overview and High Level Architecture," The Open Geospatial Consortium whitepaper, 20 August 2008.

[34] A. Bröring, J. Echterhoff, S. Jirka, I. Simonis, T. Everding, C. Stasch, S. Liang, R. Lemmens, "New Generation SensorWeb Enablement," *Sensors*, 11 (3), pp. 2652–2699, 2011.

[35] T. Berners-Lee, J. Hendler, and O. Lassila,"The Semantic Web," *Scientific American,* March 2001.

[36] Resource Description Framework (RDF), http://www.w3.org/RDF/, приступ 01.04.2018.

[37] A. Hertel, J. Broekstra and H. Stuckenschmidt, "RDF Storage and Retrieval System," *In Handbook on Ontologies, S. Staab and R. Studer (editors), second edition*, Springer 2009, pp. 489-508., 2009.

[38] RDF 1.1 N-Triples, http://www.w3.org/TR/n-triples/, приступ 01.04.2018.

[39] Notation3 (N3): A readable RDF syntax, https://www.w3.org/TeamSubmission/n3/, приступ 01.04.2018.

[40] RDF/XML Syntax Specification, http://www.w3.org/TR/REC-rdf-syntax/, приступ 01.04.2018.

[41] Turtle - Terse RDF Triple Language, http://www.w3.org/TeamSubmission/turtle/, приступ 01.04.2018.

[42] RDF Vocabulary Description Language 1.0: RDF Schema, https://www.w3.org/TR/rdf-schema/, приступ 01.04.2018.

[43] OWL Web Ontology Language, https://www.w3.org/OWL/, приступ 01.04.2018.

[44] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/, приступ 01.04.2018.

[45] T. Berners-Lee,"Design Issues: Linked Data",World Wide Web Consortium (W3C) note, July 2006, www.w3.org/DesignIssues/LinkedData.html, приступ 01.04.2018.

[46] LinkedOpenData, https://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData, приступ 01.04.2018.

[47] H. Patni, C. Henson, and A. Sheth, "Linked Sensor Data", In *Proceedings of 2010 International Symposium on Collaborative Technologies and Systems*, 2010.

[48] P. Barnaghi, M. Presser, and K. Moessner, "Publishing Linked Sensor Data", *In Proceedings of the 3rd International Workshop on Semantic Sensor Networks (SSN), In conjunction with the International Semantic Web Conference (ISWC)* 2010, November 2010.

[49]J. F. Sequeda, O. Corcho, "Linked stream data: a position paper," In *Proceedings of the 2nd International Conference on Semantic Sensor Networks*, Volume 522, pp. 148-157, 2009.

[50] Comet, http://infrequently.org/2006/03/comet-low-latency-data-for-the-browser, приступ 01.04.2018.

[51] I. Fette and A. Melnikov, "The Websocket Protocol," IETF Internet draft, work in progress, Dec. 2011.

[52] HTML Canvas 2D Context, http://www.w3.org/TR/2dcontext/, приступ 01.04.2018.

[53] Web Services Description Language (WSDL), https://www.w3.org/TR/2007/REC-wsdl20-20070626/, приступ 01.04.2018.

[54] Simple Object Access Protocol (SOAP) 1.1, https://www.w3.org/TR/2000/NOTE-SOAP-20000508/, приступ 01.04.2018.

[55] Z. Shelby, K. Hartke, C. Bormann, "The Constrained Application Protocol (CoAP), RFC7252", https://tools.ietf.org/html/rfc7252, June 2014, приступ 01.04.2018.

[56] C. Gutwin, M. Lippold, and T. C. N. Graham, "Real-Time Groupware in the Browser : Testing the Performance of Web-Based Networking," in *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pp. 167–176, 2011.

[57] Message Queue Telemetry Transport, MQTT, http://mqtt.org/, приступ 01.04.2018.

[58] Advanced Messaging Quieing Protocol, https://www.amqp.org/, accessed 20-July-2016.

[59] OASIS, "OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0," Burlington, MA, USA, https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=amqp, приступ 01.04.2018.

[60] Extensible Messaging and Presence Protocol (XMPP), http://xmpp.org/, приступ 01.04.2018.

[61] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," http://www.rfc-editor.org/info/rfc6120, 2011, приступ 01.04.2018.

[62] S. Bendel, T. Springer, D. Schuster, A. Schill, "A Service Infrastructure for the Internet of Things based on XMPP," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2013 IEEE International Conference on,* pp. 385-388, 2013.

[63] M. Kirsche and R. Klauck, "Unify to bridge gaps: Bringing XMPP into the internet of things," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on,* pp. 455 –458, 2012.

[64] A. Hornsby and E. Bail, "µXMPP: Lightweight implementation for low power operating system Contiki," in *Ultra Modern Telecommunications Workshops, 2009. ICUMT '09. International Conference on,* pp. 1 –5, October 2009.

[65] XMPP client for mbed, https://developer.mbed.org/cookbook/XMPPClient, приступ 01.04.2018.

[66] Data Distribution Service (DDS), http://portals.omg.org/dds/, приступ 01.04.2018.

[67] Extensible Markup Language (XML), http://www.w3.org/XML/, приступ 01.04.2018.

[68] P.Waher and Y. Doi, Efficient XML Interchange (EXI) Format, Std. XEP-0322, 2013.

[69] T. Kamiya and J. Schneider, “Efficient XML Interchange (EXI) Format 1.0,” World Wide Web Consortium, Cambridge, MA, USA, Recommend. REC-Exi-20110310, 2011.

[70] Google Protocol Buffer, https://developers.google.com/protocol-buffers/, приступ 01.04.2018.

[71] JVM-Serializers Benchmark, https://github.com/eishay/jvm-serializers/wiki, приступ 01.04.2018.

[72] N. Fernández, J. Arias, L. Sanchez, D. Fuentes-Lorenzo, and O. Corcho, “RDSZ: An Approach for Lossless RDF Stream Compression,” In *Proceedings of the Extended Semantic Web Conference (ESWC), Volume LNCS 8465,* pp. 52–67, 2014.

[73] A. K. Joshi, P. Hitzler, and G. Dong, “Logical Linked Data Compression,” In *The Semantic Web: Semantics and Big Data, Volume LNCS 7882,* pp. 170–184. 2013.

[74] J. D. Fernández, M. A. Martínez-Prieto, C. Gutiérrez, A. Polleres, and M. Arias, “Binary RDF Representation for Publication and Exchange,” *Journal of Web Semantics,* Vol. 19, pp. 22–41, 2013.

[75] A. Kos, S. Tomažič, and A. Umek, "Evaluation of Smartphone Inertial Sensor Performance for Cross-Platform Mobile Applications," *Sensors* 16, vol. 16(4), 2016.

[76] IoT Developer Survey 2016, https://www.slideshare.net/IanSkerrett/iot-developer-survey-2016, приступ 01.04.2018.

[77] A. Botta, W. Donato , V. Persico , A. Pescapé, "Integration of Cloud Computing and Internet of Things: A Survey," *Future Generation Computer Systems,* vol. 56, pp. 684-700, March 2016.

[78] M. Díaz , C. Martín, , B. Rubio, "State-of-the-art, Challenges, and Open Issues in the Integration of Internet of Things and Cloud Computing," *Journal of Network and Computer Applications,* vol. 67, pp. 99–117, May 2016.

[79] J. Mineraud, O. Mazhelisb, X. Suc, S. Tarkoma, "A Gap Analysis of Internet-of-Things Platforms," *Computer Communications,* vol. 89-90, pp. 5–16, 2016.

[80] G. Gospavic, Z. Babovic, D. Mijatovic, M. Petrovic, “Implementacija savremenih rešenja za upravljanje distributivnim sistemima baziranim na iskustvima korisnika“, *41. Kongres o grejanju, hladjenju i klimatizaciji,* Beograd, Serbia, 1-3 December 2010.

[81] D. Mills, “IEEE 1588 Precision Time Protocol (PTP),” http://www.eecis.udel.edu/~mills/ptp.html, приступ 01.04.2018.

[82] D. Mills et al., “Network Time Protocol Version 4: Protocol and Algorithms Specification,” IETF RFC 5905, http://tools.ietf.org/html/rfc5905, June 2010, приступ 01.04.2018.

[83] S. Hadim, and N. Mohamed, “Middleware Challenges and Approaches for Wireless Sensor Networks,” *IEEE Distributed Systems Online*, vol. 7(3), 2006.

[84] S.R. Madden, M.J. Franklin, J.M. Hellerstein, and W. Hong, ”TinyDB: an acquisitional query processing system for sensor networks,” *ACM Transactions on Database Systems,* vol. 30(1) pp. 122-173, 2005.

[85] N. Guarino, “Formal Ontology in Information Systems,” *In Proceedings of the 1st International Conference,* Trento, Italy, June 6-8, 1998.

[86] Semantic Sensor Network XG Final Report, W3C Incubator Group Report 28 June 2011, https://www.w3.org/2005/Incubator/ssn/XGR-ssn-20110628/, приступ 01.04.2018.

[87] Semantic Sensor Network Ontology, https://www.w3.org/TR/vocab-ssn/, приступ 01.04.2018.

[88] Russomanno, D., Kothari, C., Thomas, O.,"Sensor ontologies: from shallow to deep models," System Theory, 2005. *SSST '05. Proceedings of the Thirty-Seventh Southeastern Symposium on,* pp. 107-112, March 2005.

[89] IEEE Suggested Upper Merged Ontology (SUMO), http://www.ontologyportal.org/, приступ 01.04.2018.

[90] The NASA SWEET Ontology, http://sweet.jpl.nasa.gov/ontology/, приступ 01.04.2018.

[91] XPath, http://www.w3.org/TR/xpath/all/, приступ 01.04.2018.

[92] XQuery, http://www.w3.org/TR/xquery/all/, приступ 01.04.2018.

[93] P. Bonnet, Gehrke, J., Seshadri, P.,"Towards sensor database systems," In Proceedings of Mobile Data Management, volume 1987, *Lecture Notes in Computer Science. Springer,* Hong Kong, January 2001.

[94] A. Kansal, S. Nath, J. Liu, F. Zhao, "SenseWeb: an infrastructure for shared sensing," *IEEE Multimedia,* vol. 14(4), pp. 8–13, 2007.

[95] Lewis, M. Cameron, D., Xie, S., Arpinar, B.,"ES3N: A semantic approach to data management in sensor networks," In *Semantic Sensor Networks Workshop (SSN06),* Athens, Georgia, USA, Nov. 2006.

[96] D. Pfisterer et al., ''SPITFIRE: Toward a semantic Web of things,'' *IEEE Communications Magazine,* vol. 49(11), pp. 40–48, Nov. 2011.

[97] D. L. Phuoc, "SensorMasher: publishing and building mashup of sensor data", In *Proceedings of the 5th International Conference on Semantic Systems (I-Semantics 2009),* 2009.

[98] Li, L., Taylor, K.,"A framework for semantic sensor network services," In *6th International Conference on Service Oriented Computing,* 2008.

[99] Calbimonte, J.P., Corcho, O., Gray, A.J.G., "Enabling Ontology-based Access to Streaming Data Sources", 9th International Semantic Web Conference (ISWC2010), Shanghai, China, November 2010.

[100] Gray Alasdair J. G. Gray et al., "A Semantic Sensor Web for Environmental Decision Support Applications," *Sensors*, vol. 11, pp. 8855-8887, 2011.

[101] Brenninkmeijer, C.Y., Galpin, I., Fernandes, A.A., Paton, N.W., "A semantics for a query language over sensors, streams and relations," In *BNCOD '08.,* pp. 87-99, 2008.

[102] Aberer, K., Hauswirth, M., Salehi, A., "A middleware for fast and flexible sensor network deployment, "In *Proceedings of 32nd International Conference on Very Large Data Bases VLDB, pp. 1199–1202, VLDB Endowment,* 2006.

[103] Calbimonte, J.P., Sarni, S., Eberle, J., Aberer, K.,"Xgsn: An open-source semantic sensing middleware for the web of things," In *7th International SSN Workshop,* 2014.

[104] V. Tsiatsis, A. Gluhak, T. Bauge, F. Montagut,J. Bernat, M. Bauer, C. Villalonga, P. Barnaghi, S. Krco, "The SENSEI Real World Internet Architecture,"

in Towards the Future Internet—Emerging Trends From European Research, IOS Press, Amsterdam, The Netherlands: IOS Press, pp. 247–256., 2010.

[105] C. Villalonga, M. Bauer, V. Huang, J. Bernat, and P. Barnaghi, "Modeling of sensor data and context for the real world internet," in *7th IEEE Workshop on Context Modeling and Reasoning (CoMoRea) at the 8th IEEE International Conference on Pervasive Computing and Communication (PerCom'10),* IEEE, 2010.

[106] M. Bauer, M. Boussard, N. Bui, J. De Loof, C. Magerkurth, S. Meissner, A. Nettstrater, J. Stefa, M. Thoma, and J. Walewski, "IoT Reference Architecture," in Enabling Things to Talk, Designing IoT solutions with the IoT Architectural Reference Model, Springer Berlin Heidelberg, pp. 163–211, 2013.

[107] J. Kiljander et al., "Semantic Interoperability Architecture for Pervasive Computing and Internet of Things," *IEEE Access,* vol. 2, pp. 856 - 873, 2014.

[108] R. Giaffreda, ''iCore: A cognitive management framework for the Internet of Things,'' in The Future Internet, A. Galis and A. Gavras, Eds., Springer, Heidelberg, Germany, pp. 350–352., 2013.

[109] Le Phuoc, D., Nguyen-Mau, H.Q., Parreira, J.X., Hauswirth, M.,"A middleware framework for scalable management of linked streams," *Journal of Web Semantics,* Vol. 16, pp. 42–51, 2012.

[110] D.L. Phuoc et al., "A Native and Adaptive Approach for Unified Processing of Linked Streams and Linked Data," *Proc. 10th Int'l Conf. Semantic Web,* pp. 370–388, 2011.

[111] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.P. Calbimonte, M. Riahi, K. Aberer, P.P. Jayaraman, A.B. Zaslavsky, I.P. Zarko, L. Skorin-Kapov, R. Herzog, "OpenIoT: Open Source Internet-of-Things in the Cloud," In: Podnar Žarko I., Pripužiæ K., Serrano M. (eds) Interoperability and Open-Source Solutions for the Internet of Things, *Lecture Notes in Computer Science,* Vol. 9001, Springer, pp.13-25, 2015.

[112] D. Puiu et al., "CityPulse: Large Scale Data Analytics Framework for Smart Cities," *IEEE Access,* vol. 4, pp. 1086 - 1108, 2016.

[113] Y. Qin, Q. Z. Sheng, and E. Curry,"Matching Over Linked Data Streams in the Internet of Things," *IEEE Internet Computing,* vol. 19(3), pp. 21–27, 2015.

[114] Luckenbach, T., Gober, P., Arbanowski, S., Kotsopoulos, A., Kim, K.,"TinyREST: A Protocol for Integrating Sensor Networks into the Internet," *In Proceedings of the REALWSN 2005,* pp. 1-5, 2005.

[115] EU FP6 SANY, https://cordis.europa.eu/project/rcn/79757_en.html, приступ 01.04.2018.

[116] European Space Agency http://www.esa.int/, приступ 01.04.2018.

[117] North52, https://52north.org/software/software-projects/sos/, приступ 01.04.2018.

[118] Henson, C., Pschorr, J. K., Sheth, A. P., Thirunarayan, K.,"SemSOS: Semantic Sensor Observation Service," in *Proceedings of the 2009 International Symposium on Collaborative Technologies and Systems (CTS 2009),* Baltimore, MD, 2009.

[119] The XML Linking Language (XLINK), http://www.w3.org/TR/xlink/, приступ 01.04.2018.

[120] The Geography Markup Language (GML), http://www.opengeospatial.org/standards/gml, приступ 01.04.2018.

[121] The Time Ontology in OWL (OWL-Time), http://www.w3.org/TR/owl-time/, приступ 01.04.2018.

[122] Shneidman, J., Pietzuch, P., Ledlie, J., Roussopoulos, M., Seltzer, M., Welsh, M., "Hourglass: An Infrastructure for Connecting Sensor Networks and Applications," *Harvard Technical Report TR-21-04.*, 2004.

[123] Liu, J., Zhao, F., "Towards semantic services for sensor-rich information systems," In *2nd International Conference on Broadband Networks,* pp. 44–51, 2005.

[124] Bouilet, E., Feblowitz, M., Liu, Z., Ranganathan, A.,Riabov, A.,Ye, F.,"A Semantics-Based Middleware for Utilizing Heterogeneous Sensor Networks," In: *DCOSS'07*, 2007.

[125] Huang, V., Javed, M.,"Semantic sensor information description and processing," In *Proceedings of the 2nd International Conference on Sensor Technologies and Applications,* 2008.

[126] Zafeiropoulos, A., Konstantinou, N., Arkoulis, S., Spanos, D.-E., Mitrou, N.,"A Semantic-Based Architecture for Sensor Data Fusion," In *Proceedings of the 2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM '08),* Washington, DC, USA, pp. 116-121, 2008.

[127] Wun, A., Petrovic, M., Jacobsen., H.A.,"A system for semantic data fusion in sensor networks," In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems (DEBS '07),* ACM, New York, USA, pp. 75-79, 2007.

[128] Levis, P., Gay, D., Culler, D.,"Active Sensor Networks," In NSDI, 2005.

[129] Gibbons, P. B., Karp, Ke, B., Nath, S., Seshan, S.,"Iris-Net: An architecture for a worldwide sensor web," *IEEE Pervasive Computing*, Volume 2, pp. 22–33, Oct.-Dec. 2003.

[130] Moodley, D., Simonis, I.,"New architecture for the sensor web: the SWAP framework," In *5th International semantic web conference, ISWC* 2006, Athens, GA, USA, 2006.

[131] The OWL-S: Semantic Markup for Web Services, http://www.w3.org/Submission/OWL-S/, приступ 01.04.2018.

[132] V. Milutinovic, M. Kotlar, M. Stojanovic, I. Dundic, N. Trifunovic, Z. Babovic, "DataFlow Systems: From Their Origins to Future Applications in Data Analytics, Deep Learning, and the Internet of Things", In DataFlow Supercomputing Essentials, Computer Communications and Networks. Springer, 2017.

[133] IoT-Lite Ontology, https://www.w3.org/Submission/iot-lite/, приступ 01.04.2018.

[134] H. Zhang, G. Chen, B. C. Ooi, K.-L. Tan, and M. Zhang, "In-memory big data management and processing: a survey," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 27(7), pp. 1920–1948, 2015.

[135] B. Shao, H. Wang, and Y. Li, "Trinity: A distributed graph engine on a memory cloud," in *Proceedings ACM SIGMOD International Conference Management Data*, pp. 505-516, 2013.

[136] A. Mauri, J. Calbimonte, D. Dell'Aglio, M. Balduini, M. Brambilla, E. D. Valle, and K. Aberer,"Triplewave: Spreading RDF streams on the web," In *the Semantic Web - ISWC 2016 - 15th International Semantic Web Conference*, Kobe, Japan, October 17-21, 2016.

[137] Y. Zhou, S. De, W. Wang, and K. Moessner,"Search techniques for the web of things: A taxonomy and survey," *Sensor*s, vol. 16(5), 600, 2016.

[138] Zoran Babović, "Skladištenje RDF podataka u bazama podataka," Seminarski rad iz predmeta Objektne i relacione baze podataka, Elektrotehnički fakultet, Univerzitet u Beogradu, 43p, 2014.

[139] Z. Kaoudi, I. Manolescu, "RDF in the clouds: a survey," *The VLDB Journal*, vol. 24(1), pp. 67-91, 2015.

[140] Chang, F., Dean, J., Ghemawat, S., Hsieh, W.C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., Gruber, R. E., "Bigtable: A distributed storage system for structured data, " In *Proceedings of the 7th Conference on USENIX Symposium on Operating Systems Design and Implementation* – vol. 7, pp. 205-218, 2006.

[141] Apache HBase, http://hbase.apache.org/, приступ 01.04.2018.

[142] Lakshman, A. and P. Malik, "Cassandra: a decentralized structured storage system," *Operating Systems Review,* vol. 44(2): pp. 35-40, 2010.

[143] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, and K. Taylor, "IoT-Lite: A lightweight semantic model for the internet of things and its use with dynamic semantics," *Personal and Ubiquitous Computing*, vol. 21(3), pp. 475-487, 2017.

[144] M. Ganzhaa, M. Paprzyckia, W. Pawlowski, P. Szmejaa, K. Wasielewska, "Semantic interoperability in the Internet of Things: An overview from the INTER-IoT perspective," *Journal of Network and Computer Applications,* vol. 81, pp. 111-124, 2017.

[145] QUDT Ontology, http://www.qudt.org/pages/HomePage.html, приступ 01.04.2018.

[146] K. Zeng, J. Yang, H. Wang, B. Shao, and Z. Wang, "A Distributed Graph Engine for Web Scale RDF Data, " *VLDB Endowment*, vol. 6(4), 2013

[147] W.-S. Ku, H. Chen, C.J. Wang, C. M. Liu, "Geo-Store: A Framework for Supporting Semantics-Enabled Location-Based Services," *IEEE Internet Computing*, vol. 17(2), 2013.

[148] M. Acosta, M.-E. Vidal, T. Lampo, J. Castillo, and E. Ruckhaus,"ANAPSID: An adaptive query processing engine for SPARQL endpoints," In *ISWC*, pp. 18–34, Springer, 2011.

[149] T. Neumann and G. Weikum, "RDF-3X: a RISC-style Engine for RDF," *In Proceedings of the VLDB Endowment*, 1(1):647–659, 2008.

[150] J. Broekstra, A. Kampman, and F. van Harmelen, "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema, " In *ISWC*, pp 54–68, 2002.

[151] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds, "Efficient RDF Storage and Retrieval in Jena2, " In *SWDB*, pp 131–150, 2003.

[152] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan, ”An Efficient SQL-based RDF Querying Scheme, “ In *Proceedings of VLDB Endowment,* pp 1216–1227, 2005.

[153] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, “Scalable Semantic Web Data Management Using Vertical Partitioning,” In *Proceedings of the 33rd International Conference on Very Large Data Bases, VLDB ’07*, pp 411–422, 2007.

[154] C. Weiss, P. Karras, and A. Bernstein, “Hexastore: Sextuple Indexing for Semantic Web Data Management,“ *In Proceedings of the VLDB Endowment*, 1(1):1008–1019, 2008.

[155] J. Huang, D. J. Abadi, and K. Ren, “Scalable SPARQL Querying of Large RDF Graphs,” *In PVLDB*, 4(11), 2011.

[156] N. Papailiou, I. Konstantinou, D. Tsoumakos, and N. Koziris, “H2RDF: Adaptive Query Processing on RDF Data in the Cloud,” *In WWW*, 2012.

[157] M. A. Bornea, et al., "Building an Efficient RDF Store Over a Relational Database," In *SIGMOD*, 2013.

[158] M. Atre, V. Chaoji, M. J. Zaki, and J. A. Hendler, "Matrix bit loaded: A scalable lightweight join query processor for RDF data," In *Proc. of WWW 2010*, pp. 41–50. ACM, 2010.

[159] P. Yuan, P. Liu, B.Wu, H. Jin, W. Zhang, and L. Liu,"TripleBit: a Fast and Compact System for Large Scale RDF Data," In *PVLDB*, vol. 6(7), 2013.

[160] Gurajada, S., Seufert, S., Miliaraki, I., Theobald, M.,"TriAD: A distributed shared-nothing RDF engine based on asynchronous message passing," In *SIGMOD* 2014.

[161] M. Hammoud, D. A. Rabbou, R. Nouri, S.-M.-R. Beheshti, and S. Sakr, "DREAM: Distributed RDF Engine with Adaptive Query Planner and Minimal Communication," *Proc. VLDB Endow.*, 8(6), pp. 654-665, 2015.

[162] Schatzle, A., Przyjaciel-Zablocki, M., Skilevic, S., Lausen, G., "S2RDF: RDF querying with SPARQL on spark," In *PVLDB*, vol. 9(10), pp. 804–815, 2016.

[163] R. Harbi, I. Abdelaziz, P. Kalnis, N. Mamoulis, Y. Ebrahim, and M. Sahli, “Accelerating SPARQL Queries by Exploiting Hash-based Locality and Adaptive Partitioning,” *The VLDB Journal,* pp. 1–26, 2016.

[164] G. Stefanoni, B. Motik, E. V. Kostylev, "Estimating the Cardinality of Conjunctive Queries over RDF Data Using Graph Summarisation," In *Proceedings of the 2018 World Wide Web Conference,* pp. 1043-1052, 2018.

8 Прилози

Прилог 1 – SPARQL упити за тестирање RDF складишта

SPARQL упит бр. 2 – селектовање температурних сензора који су измерели вредност изнад 75° фаренхајта (23° целзијуса).

```
prefix om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
prefix xsd:<http://www.w3.org/2001/XMLSchema#>

SELECT DISTINCT ?sensor
WHERE {
    ?sensor om-owl:generatedObservation ?observation .
    ?observation rdf:type weather:TemperatureObservation .
    ?observation om-owl:result ?result .
    ?result om-owl:floatValue ?value .
    ?result om-owl:uom weather:fahrenheit .

    FILTER(?value >= "75.0"^^xsd:float)
}
```

SPARQL упит бр 3. – просторно-временски-вредносни упит - селектовање сензора који су измерили температуру испод 65° фаренхајта (18,33° целзијуса) у периоду од 8 часова увече до 8 часова ујутру 8. и 9. августа 2004. године, апроксимативно на простору америчке државе Аризоне.

```
prefix om-owl:<http://knoesis.wright.edu/ssw/ont/sensor-observation.owl#>
prefix rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix weather:<http://knoesis.wright.edu/ssw/ont/weather.owl#>
prefix sens-obs:<http://knoesis.wright.edu/ssw/>
prefix xsd:<http://www.w3.org/2001/XMLSchema#>
prefix owl-time:<http://www.w3.org/2006/time#>
prefix wgs84:<http://www.w3.org/2003/01/geo/wgs84_pos#>

SELECT DISTINCT ?sensor
WHERE {
    ?sensor om-owl:generatedObservation ?observation .
    ?observation rdf:type weather:TemperatureObservation .
    ?observation om-owl:samplingTime ?time .
    ?time owl-time:inXSDDateTime ?xsdtime .
    ?observation om-owl:result ?result .
    ?result om-owl:floatValue ?value .
    ?result om-owl:uom weather:fahrenheit .
    ?sensor om-owl:processLocation ?location .
    ?location rdf:type wgs84:Point .
    ?location wgs84:lat ?latitude .
    ?location wgs84:long ?longitude .

    FILTER(?value <= "65.0"^^xsd:float)
    FILTER(?xsdtime >= "2004-08-08T20:00:00-
07:00^^http://www.w3.org/2001/XMLSchema#dateTime")
```

```
    FILTER(?xsdtime <= "2004-08-09T08:00:00-
07:00^^http://www.w3.org/2001/XMLSchema#dateTime")
    FILTER(?latitude >= "31.30"^^xsd:float && ?latitude <= "37.00"^^xsd:float)
    FILTER(?longitude >= "-114.60"^^xsd:float && ?longitude <=
    "-109.05"^^xsd:float) }
```

9 Биографија аутора

Зоран Бабовић је рођен 14.09.1973. године у Лазаревцу. Основне студије је завршио на Електротехничком факултету у Београду, смер Рачунарска техника и информатика, са просечном оценом 8,07 (осам и 7 од 100) и просеком оцена 8,65 из предмета са Катедре за рачунарску технику и информатику, одбранивши дипломски рад на тему: "МПЕГ 1-2 Мултиплексер" код професора Вељка Милутиновића. Докторске студије је уписао школске 2011/2012. године на модулу Рачунарска техника и информатика и положио је све испите са просечном оценом 10. Од 2006. године запослен је као истраживач приправник у Иновационом центру Електротехничког факултета Универзитета у Београду, а у јуну 2016. године је изабран у звање истраживач-сарадник. Од 2011. године је ангажован на пројекту Министарства просвете, науке и технолошког развоја (МПНТР), „ИИИ44006: Развој нових информационо-комуникационих технологија коришћењем напредних математичких метода, са применама у медицини, енергетици, е-Управи, телекомуникацијама и заштити националне баштине".

Зоран Бабовић је научно-истраживачко искуство започео 2003. године у сарадњи са немачким институтом IPSI Fraunhofer на пројектима у области мултимедије. Даље искуство је стекао учествовањем у пет пројеката финансираним од стране EU у оквиру шестог и седмог оквирног програма (FP6 и FP7) у периоду од 2007. до 2013. године и такође је учествовао на пет иновационих пројеката финансираним од МПНТР Републике Србије, као члан тима или као водећи истраживач такође у истом периоду. Пројекти припадају области сензорских мрежа, архитектуре рачунара, проналажења скривеног знања и електронске управе.

Први је аутор на два научна рада објављена у часописима са импакт фактором у категоријама М21 и М23, а такође је коаутор у још два рада објављених у часописима категоријама М22 и М23, затим два рада у часопису од националног значаја категорије М53, шест радова објављених на међународним и домаћим конференцијама категорије М33 и М63. Такође,

коаутор је три поглавља у научним монографијама издатих од издавача Springer категорије М14/М15.

Прилог 1.

Изјава о ауторству

Потписани Зоран Бабовић
Број уписа 5077/2011

Изјављујем

да је докторска дисертација под насловом

„Семантичка интеграција сензорских мрежа“

- резултат сопственог истраживачког рада,
- да предложена дисертација у целини ни у деловима није била предложена за добијање било које дипломе према студијским програмима других високошколских установа,
- да су резултати коректно наведени и
- да нисам кршио/ла ауторска права и користио интелектуалну својину других лица.

Потпис докторанда

У Београду, 31.05.2018.

Прилог 2.

Изјава о истоветности штампане и електронске верзије докторског рада

Име и презиме аутора Зоран Бабовић

Број уписа 5077/2011

Студијски програм Рачунарска техника и информатика

Наслов рада „Семантичка интеграција сензорских мрежа“

Ментор проф. др Вељко Милутиновић

Потписани

Зоран Бабовић

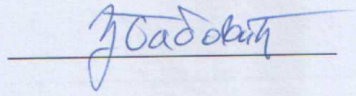
изјављујем да је штампана верзија мог докторског рада истоветна електронској верзији коју сам предао за објављивање на порталу **Дигиталног репозиторијума Универзитета у Београду**.

Дозвољавам да се објаве моји лични подаци везани за добијање академског звања доктора наука, као што су име и презиме, година и место рођења и датум одбране рада.

Ови лични подаци могу се објавити на мрежним страницама дигиталне библиотеке, у електронском каталогу и у публикацијама Универзитета у Београду.

Потпис докторанда

У Београду, 31.05.2018.



Прилог 3.

Изјава о коришћењу

Овлашћујем Универзитетску библиотеку „Светозар Марковић“ да у Дигитални репозиторијум Универзитета у Београду унесе моју докторску дисертацију под насловом:

„Семантичка интеграција сензорских мрежа“

која је моје ауторско дело.

Дисертацију са свим прилозима предао сам у електронском формату погодном за трајно архивирање.

Моју докторску дисертацију похрањену у Дигитални репозиторијум Универзитета у Београду могу да користе сви који поштују одредбе садржане у одабраном типу лиценце Креативне заједнице (Creative Commons) за коју сам се одлучио:

1. Ауторство
2. Ауторство – некомерцијално
3. Ауторство – некомерцијално – без прераде
4. Ауторство – некомерцијално – делити под истим условима
5. Ауторство – без прераде
6. Ауторство – делити под истим условима

(Молимо да заокружите само једну од шест понуђених лиценци, кратак опис лиценци дат је на полеђини листа).

Потпис докторанда

У Београду, 31.05.2018. _____