Магдалена Костоска

Интероперабилност на софтвер како сервис во пресметување во облак

Докторска дисертација

Тема	Интероперабилност на софтвер како сервис во пресметување во облак
Автор	Магдалена Костоска
Научна област	Информатички технологии
Ментор	проф. Д-р Марјан Гушев, редовен професор на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
	проф. Д-р Марјан Гушев, редовен професор на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
Членови на комисија	проф. Д-р Гоце Арменски, доцент на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
	проф. Д-р Иван Чорбев, вонреден професор на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
	проф. Д-р Боро Јакимовски, доцент на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
	проф. Д-р Сашко Ристов, доцент на Универзитетот „Св. Кирил и Методиј“, Факултет за информатички науки и компјутерско инженерство - Скопје
Датум на одбрана	_____
Датум на промоција	_____

Посветена на татко ми Штерјо кој одамна не е повеќе меѓу нас, а многу ќе се гордееше со оваа докторска дисертација

Благодарност

Најпрво сакам најмногу да му заблагодарам на мојот ментор проф. д-р Марјан Гушев кој ме водеше низ цел овој процес, ме насочуваше кон вистинскати идеи и насоки за истражување, неговото активно учество и поддршка во истражувањето и пишувањето на нашите објавени трудови. Точно знаеше да го процени потенцијалот на секоја моја идеа и да ми предложи свои во моментите кога не бев сигурна како понатаму. Се надевам и ќе ми биде многу мило ако го продолжиме нашето заедничко истражување во иднина.

Многу сакам да се заблагодарам и мојата мајка, сестра и Милан кои ме поддржуваа и ми помагаа на секој начин на кој можеа додека ја изработував оваа теза.

Голема благодарност и колегата Сашко Ристов за тое што ме мотивираше и ми помагаше, како и до Александар Доневски за неговата техничка помош во секое време.

Предговор

Истражувањето за оваа докторска дисертација е спроведено на Универзитетот „Свети Кирил и Методиј“, Факултетот за информатички науки и компјутерско инженерство во Скопје, Република Македонија. Во текот на мојот магистерски труд со наслов „e-Трезор - Трезорски Информационен Систем“ започнав со истражување на темата Интероперабилност на е-влада и генерално интероперабилност на големи системи. Ме заинтригира овој концепт и моќноста која ја нуди и во моментот кога требаше да се насочам кон своето истражување за докторат решив да направам синергија од овој концепт и тогаш релативно новиот, но исто така моќен, концепт на пресметување во облак, која нудеше можност за истражување во оваа насока.

Истражувањето започна со генерален преглед на оваа тема и насоки за истражување. Бидејќи се чинеше неизвесна можноста за интероперабилност на пресметувањето во облак, започнав со истражување и генерирање на адаптерски услуги во дадени домени. Во 2012 се појави првиот предлог за TOSCA стандардот кој треба да понуди портабилност (како дел од интероперабилност) и се чинеше како предизвик да се имплементира. Од тој момент натака својата истражувачка работа ја насочив во оваа насока и си го наметнав предизвикот добро да го истражам овој предлог стандард, да ги идентификувам неговите недостатоци, да разгледам негови можни подобрувања, да ја истражам неговата изводливост и конечно да креирам практично решение за да ја докажам таа изводливост. Со задоволство можам да кажам дека го совладав предизвикот кој си го наметнав.

Вовед

Појавата на концептот на пресметување во облак понуди многу опции за голем број корисници. Можноста за закуп на сервисите кои ги нуди (како

хардвер, софтвер итн..) многу значат за компании кои не се во можност да постават свои компјутерски центри. Дополнително можноста за скалирање и еластичност на тие сервиси, до навидум неограничени ресурси и можност за користење на тие ресурси во даден временски период потребен на корисникот и нивно ослободување нуди можност за користење во моментите кога сопствените ресурси не се доволни.

Сепак со појавата на голем број на понуди за сервиси од пресметување во облак се појави потребата од менување на операторот кој ги нуди овие сервиси, како потребата за соработка на повеќе сервиси од различни оператори. Со тоа се наметна и прашањето и потребата за интероперабилност и портабилност во оваа област, кои се денес активни насоки за истражување и работа. Моето истражување го насочив на најгорниот слој од сервисите кои ги нуди облакот: Софтвер како сервис (SaaS).

Опис на проблемот и целите

Често се врши споредба на пресметката во облак со моделот на достава на електрична енергија – и во двата случаи корисниците ги користат услугите на моделите, без потреба да има потреба да ги разбираат компонентите на уредите или инфраструктурата која е потребна за овозможување на услугата. Во овој контекст е првата цел на оваа докторска дисертација: да се креираат адаптери на дадени сервиси и се што ќе биде потребно за еден корисник е да знае кој адаптер треба да го избере за даден сервис да може да го користи, при услови да има веќе друг постоечки сервис (на ист начин на кој сега се користат адаптерите за електрична енергија – доколку знаете што имате за користење на услугата и знаете какво побарување има добавувачот ќе може да изберете соодветен адаптер). Со оглед на тоа дека во брз рок нема да има реални унифицирани решенија за интероперабилност на сервисите во Софтвер како сервис од страна на самите добавувачи во облакот, ова решение во голем обем би помогнало за решавање на проблемите на корисници на овие сервиси за кои се очекуваше дека ќе се појават во скор рок со оглед на брзината на развојот на моделот Пресметување во облак. Следната цел на овој докторски труд е насочена кон овозможување портабилност на апликации во облак, односно овозможување на модел со кој дадена апликација лесно ќе биде мигрирана во облак или пренесена од еден облак во друг.

Основните прашања за истражување на оваа дисертација се:

- Дали е можно моделирање на интероперабилност за Пресметување во облак?
- Ако е можно моделирање на интероперабилност за Пресметување во облак, како се врши?
- Дали е можно моделирање на интероперабилност за Пресметување во облак со помош на адаптери?

- На кои познати технолошки и методолошки пристапи за Софтвер како сервис кај Пресметување во облак и ќе одговорат на сите предложените адаптер?
- Дали е можно моделирање на портабилност на апликации?
- Дали е изводлив моделот на портабилност на апликации во облак?
- Каде е применлив моделот на портабилност на апликации во облак?
- Дали е исплатлив (во однос на време и перформанси) моделот на портабилност на апликации во облак?

Опсег на дисертацијата

Опсегот на оваа дисертација е:

- Анализа на предложени модели за интероперабилност на различните слоеви во облакот Инфраструктура како сервис (IaaS), Платформа како сервис (PaaS) и Софтвер како сервис (SaaS) и нивна евалуација;
- Испитување и анализа на дел од познати софтверски сервиси во облак и моделирање адаптери за овие сервиси за да се овозможи нивна интероперабилност и
- Моделирање на портабилност за апликации во облак

Методи

За да се обработат прашањата за истражување од областа на предложени модели за интероперабилност на различните слоеви во облакот разгледани се сите актуелни научни предлози од оваа тема, истите се поделени и детално разгледани според типот на интероперабилност кој го нудат и финално е поставена формална рамка за оценување на овие предлози и е извршена нивна евалуација.

За да се обработат прашањата за истражување од областа на моделирање на адаптери за познати сервиси во облак, спроведени се експерименти и креирани се прототипови кои ја демонстрираат интероперабилноста на Софтвер како сервис (SaaS) со користење на адаптери.

За истражувањето од областа на креирање на нов модел за портабилност на Софтвер како сервис (SaaS) земен е предлог моделот на TOSCA и истиот е дополнет и проширен. Спроведен е практичен експеримент во тестна околина која е изготвена за претходно истражување [107]. со кој се докажува изводливоста и применливоста на овој модел и изведена е евалуација во однос на потребно време и перформанси на овој модел, за да се разгледа неговата исплатливостза понатамошно користење.

Содржина на дисертацијата

Оваа дисертација се состои од 7 глави: 1) Основни концепти пресметување во облак, интероперабилност и модели на интероперабилност, 2) Преглед, анализа и евалуација на предложени модели за интероперабилност во облак, 3) Адаптери како средство за моделирање интероперабилност на Софтвер како сервис во облак, 4) TOSCA базиран модел за портабилност на апликации поставени на Платформа како сервис, 5) Применливост и користење на P-TOSCA моделот, 6) Резултати од евалуација на P-TOSCA моделот и 7) Заклучни согледувања.

Во Глава 1 се прикажани основните концепти и дефиниции за потребите на оваа дисертација. Таа ги опишува основните концепти и архитектурата на облакот, моделите за поставување и слоевите на сервиси. Во оваа глава се дадени и основните дефиниции за интероперабилност, нејзините карактеристики, модели и применливост. На крај се даваат дефиниции за интероперабилност и портабилност на Пресметување во облак, како и сценарија овозможување на овие концепти.

Глава 2 го анализира и обработува предложени и објавени научни модели за интероперабилност на различните слоеви во облакот, врши нивна категоризација според пристапот и типот на интероперабилност кој го нудат. Даден е преглед на секој предложен модел поединечно и се поставува формална рамка за оценување на овие предлози и се врши нивна евалуација.

Глава 3 го обработува обработува концептот на адаптер и користење на програмирачки интерфејси на апликации (API-а) за пристап и преземање на информации. Дава опис на типовите на интерфејсите кои актуелно се користат, прикажува како се моделира адаптерско решение и дава примери на моделирани адаптери.

Глава 4 го разгледува предложениот модел на TOSCA, го анализира и ги истакнува неговите слабости и се дава проширување и дополнување на моделот. Се прикажува нов модел во вид на Платформа како сервис (PaaS) за овозможување на портабилност на апликации, се дава негов генерален преглед како и детална архитектура.

Глава 5 ја прикажува применливоста и изводливоста на моделот за портабилност врз различни типови на софтверски архитектури. Дополнително дава преглед за кои типови на апликации не е применлива платформата.

Глава 6 ги прикажува резултатите од евалуацијата на моделот. За таа цел се креирани повеќе сценарија и секое од нив е евалуирано на две различни околини за облак: OpenStack и Eucalyptus.

Последната Глава 7 дава детален преглед на резултатите добиени во оваа докторска дисертација, опширно ги прикажува примените на резултатите и дава преглед на планирана идна работа.

Главни резултати

Резултатите добиени од оваа од истражувањата за оваа докторска дисертација се објавени во рецензирани конференции. Најважните резултати се во областите на анализа на интеоперабилност во облак, генерирање на адаптерски решенија во облак и моделирање на портабилност на апликации во облак.Главните резултати се елаборирани во следните секции.

Евалуација на предложени модели за интероперабилност на Пресметување во облак

Како дел од истражувањето за оваа докторска дисертација се испитувани предлог моделите за овозможување на интероперабилност на Пресметување во облак [73][65, 68]. За таа цел е извршен преглед и анализа на моделите: Унифициран интерфејс за облак / Брокер за облак, Платформа за оркестрација на облак / Оркестрациски слој, Нацрт за Интероблак, Интерфејс за управување со инфраструктура на облак и Отворен интерфејс за пресметување во облак. Формирана е рамка за евалуација која се базира на т.н. индикатори, со кои се доделуваат вредности за дадени карактеристики на предложените модели/стандарди. Во оваа рамка се разгледуваат следните карактеристики: сервисен модел за кој се однесува моделот/стандардот, статусот на документацијата, статусот на реализацијата и поддршката на предлог моделот/стандардот.

Покрај овие модели како предлози за стандарди дополнително се анализирани и други предлог стандарди: Интерфејс за управување со податоци во облак, IEEE P2301 (Профили за облак) и IEEE P2302 (Интероблак) [63].

Пристапот кој е употребен за оваа евалуација е базиран на претходни искуства за евалуација на модели на интероперабилност и стандарди [69, 103, 104, 52, 55, 109].

По евалуацијата на секој модел/стандард посебно заклучено е дека стандардот Отворен интерфејс за пресметување во облак има најдобри оцени, а воедно веќе има свои имплементации за дел од платформите за облак. Според оценките овој стандард најверојатно ќе биде прифатен од повеќето платформи и понудувачи. И стандардот Интерфејс за управување со податоци во облак има добри оцени, но му недостасуваат повеќе имплементации за различни платформи на облак. На стандардот Интерфејс за управување со инфраструктура на облак му недостасува имплементација. На предлог моделите и стандарди Брокер за облак и Оркестрациски слој им недостасува поддршка. Стандардите IEEE P2301

(Профили за облак) и IEEE P2302 (Интероблак) сеуште се во развој и немаат ни предлог документација.

Моделирање на интероперабилност на Софтвер како сервис со користење на адаптери

Кога е потребно да се овозможи соработка и интероперабилност меѓу повеќе сервиси често се случува интеракциите меѓу сервисите да не се компатибилни. Овие некомпатибилности не дозволуваат семантичка интероперабилност меѓу сервисите. Еден често користен пристап за справување со вакви ситуации е употребата на адаптери кои ќе вршат конверзија на барања во соодветен побаруван формат. Со оглед на тоа дека моделирање на интероперабилност на Софтвер како сервис е возможно даден домен, креирани се повеќе предлози и решенија со користење на софтверски адаптери за да се овозможи соработка и размена на податоци. Притоа е разгледана употребата на стандардниот Адаптер шаблон, како и шаблонот Скалабилен адаптер.

Првото решение е за користење на софтверски адаптери за креирање интероперабилност на календари овозможени од повеќе различни популарни понудувачи на услуга: Google Calendar, Microsoft Live и Facebook. Притоа се анализирани различни типови на стандарди кои се користени од овие понудувачи како и нивните програмирачки интерфејси на апликации. Генерирано е решение кое нуди интеграција и размена на податоци од дадените типови на календари и со тоа е овозможена интероперабилност во овој домен. Самото решение претставува SOAP сервис. Со менувањето на интерфејсите и на начинот на размена на податоци, и софтверскиот адаптер е соодветно менуван за да се задржи функционалноста [74].

Второто решение е за користење на софтверски адаптери за креирање интероперабилност на услуги за складирање овозможени од повеќе различни популарни понудувачи на услуга: Dropbox, Google Drive и SkyDrive (OneDrive). Генерирано е решение во вид на веб апликација кое нуди интеграција и размена на складирано податоци од различни услуги и со тоа е овозможена интероперабилност преку посредник во овој домен [62].

Третата анализа е предлог за овозможување интероперабилност меѓу различни типови на софтвер кои се користат од страна на универзитети. Извршена е анализа на постоечки нормативи и стандарди и разгледувана е нивната примена во различни сценарија. Како заклучок се дадени предлог стандарди за употреба кои делумно ќе го надминат недостатокот на интероперабилност. Така за системите за управување со универзитети е предложена употреба на SCORM и IMS Enterprise стандардите, за системите за управување со учење е предложена употреба на SCORM, QTI и IMS Enterprise стандардите и за системите за електронско тестирање

е предложена употреба на SCORM и QTI стандардите. Со примена на наведените стандарди може да се овозможи размена на информации и соработка на наведените типови на системи [71, 53].

Моделирање и дефинирање на портабилност на апликации во облак

За моделирање на портабилност на апликации во облак се истражуваат можности за модели и различни пристапи кои вклучуваат употреба на стандарди, генерирање на помошни решенија со апстрактни јазици и користење на семантички технологии. За детална анализа е избран предлог моделот Тополошки и оркестрациски сервиси за апликации (Topology and Orchestration Services for Applications - TOSCA). Извршена е детална анализа на можностите за тополошко моделирање [58] како и дадена е предлог рамка за платформа за користење на овој модел и опишани се генералните функционалности кои што би требало да се поддржат [72].

За истражување на примената и можностите на моделот за TOSCA извршено е истражување за креирање на тополошки модел на iKnow апликацијата - систем за управување со Универзитет [60]. Системот е избран поради неговата комплексност и специфичен начин на примена. Креирана е опсежна тополошка спецификација на системот и дефиниран е начин на примена.

Проширување на TOSCA моделот за портабилност и креирање на околина за примена

При креирање на околина за употреба на TOSCA моделот воочени се некои битни недостатоци, како и дополнителни потребни дефиниции за да се осигура изводливост. Од таа причина е креиран нов моделот P-TOSCA кој претставува проширување и менување на TOSCA со нови дефиниции. Развиена е целосна околина за употреба на овој модел на повеќе различни системи на облак и експериментално е докажана примената на P-TOSCA моделот во реална средина со користење на проширени спецификации за различни типови на апликации и со тоа е докажана примената на проширениот модел во реална средина [70].

Моделирање на Н-слојни и Сервисно ориентирани апликации со P-TOSCA

P-TOSCA моделот е употребен за дефиниција на Н-слојни и Сервисно ориентирани апликации. Притоа се креирани спецификации и архиви и истите се тестирани за апликации со дво-слојни дистрибуирано архитектури, апликации со три-слојни архитектури и апликации со сервисно ориентирани архитектури [59, 64, 67].

Евалуација на практичното решение за примена на моделот за портабилност

Спроведена е евалуација на развиената околина во однос на време и перформанси за да се утврди исплатливост на моделот. За таа цел генерирани се повеќе сценарија вклучувајќи и извршена е споредба на перформанси меѓу стандардна мануелна миграција, миграција со користење на моделот од сопствена околина во околина на облак и автоматизирана миграција со користење на моделот меѓу различни околини на облак. Генералниот заклучок е дека користењето на автоматизирана миграција и пренос е неколкукратно побрзо од мануелното [66].

Дополнително е извршена евалуација на безбедноста на примената на овој модел, како и на креираната околина тргнувајќи од резултатите добиени во [105].

Применливост на резултатите

Покрај важните резултати кои ги споменавме претходно, вреди да се спомене и нивната применливост.

Автоматизирана миграција меѓу облаци

Со користење на P-TOSCA моделот и неговата имплементација за различни облаци [70] се овозможува автоматизирана миграција на апликациите поставени на еден облак во друг со користење на сервисите на платформата.

Независност во избор на платформа за облак

Главниот придонес на оваа дисертација се базира врз независноста на клиентот од изборот на облак за поставување на својата апликација поради можност за лесна миграција и промена на понудувачот на услуги во облак. Притоа земени се во предвид предизвиците и ризиците од безбедноста на предложената архитектура при нејзината примена во различни околини [106] како и поедноставување на можноста за воспоставување на систем за управување со безбедност на информации [102].

Намалување на времето за миграција и промена на облак

Со користење на P-TOSCA моделот[70] значително се намалува времето потребно за поставување на апликација во облак, како и времето потребно за промена на понудувач на услуги во облак. Добиените експериментални резултати покажуваат повеќекратна заштеда на време [66].

Стандардизирано опишување на топологија на апликација и потребните операции и својства за облак

Со користење на P-TOSCA моделот [70] овозможуваме стандардизиран опис на дадена апликација во однос на нејзината топологија како и дефинирање на потребните операции и својства за примена на апликацијата во облак. Со примена на P-TOSCA моделот е предвидено дефинирање на спецификација за скалабилното и еластично e-Assessment решение [101]. Целта е полесно изведување на масивни, скалабилни и еластични курсеви како Архитектура и организација на компјутери, посебно поради задоволството на студентите за користење на електронски алатки [3]. Моделот ќе биде применет и за други апликации кои се реализирани како производ на истражувања [61, 54]

Градење на адаптери за интероперабилност на Софтвер како сервис

Со дефинираната методологија за генерирање на адаптери се овозможува генерирање на решенија за интероперабилност на Софтвер како сервис услуги од даден домен [62].

Скопје, *Магдалена Костоска*
Јуни 2014

Extended abstract

The PhD research was realized at the Ss. Cyril and Methodius University, Faculty of Information Sciences and Computer Engineering in Skopje, Macedonia. During the work on my Master's thesis titled "E-Treasury - Treasury Information System"I started researching the subject Interoperability of e-government and interoperability of large systems in general. I was intrigued by this concept and power that it offers. So at the moment when I needed to focus my research for doctoral thesis I decided to perform synergy of this concept and the relatively new at the time, but also powerful, concept for Cloud computing, which was presenting a research opportunity.

Researching started with general overview of this subject and research directions. Since at that time the opportunity to enable interoperability seemed uncertain I directed my research in research and generation of adapter services for given service domains. In 2012 the first proposal for TOSCA specification was presented and the goal of this specification was to offer portability (as part of operability). TOSCA appeared as challenge for implementation. From that moment on my research took another direction and imposed myself the challenge to do in-depth research of this proposed standard, to identify its disadvantages, to look at his possible improvements, to research its feasibility and finally to create practical solution to prove that feasibility. I'm pleased to say that I did conquer the challenge I have imposed.

Background

The concept of Cloud computing offers a lot of options for many users. Possibility for leasing for the offered services (like hardware, software etc.) means a lot for companies that doesn't have opportunity to set their own computer centers. Additionally, this concept offers scalability and elasticity of the services to seemingly unlimited resources and possibility for usage of those resources when needed and their release afterwards.

Along with the introduction of a large number of services for Cloud computing another necessity appeared: the possibility to change clod provider and the need for two cloud providers to collaborate and exchange information. Given these facts the question of cloud portability and interoperability was promoted and it still represents an active area for research and development. I have focused my research on the top layer of the Cloud computing stack of services: Software as a Service (SaaS).

Problem Description and Objectives

The concept of Cloud computing if often compared to the model of electricity supply since in both cases the users exploit the services of the model without the need to understand the underlying structure, components and the devices used by the service. In this context is the first goal of this thesis: to create adapters for given services so the user can exploit the service in the required format, given the service exist. This concept is similar to the usage of electrical adapters. Given the state of Cloud computing interoperability in the moment a general solution for SaaS interoperability will not be available for a while. The adapter solution solves the problem at this moment. The second goal of this thesis is to provide portability of applications in the cloud, more precisely to create a model that will enable automated migrations of applications in the cloud and automated transfer of applications from one cloud to another.

The main research questions of this thesis are:

- Is it possible to model Cloud computing interoperability?
- Is it possible to model Cloud computing interoperability with usage of adapter's solutions?
- Which SaaS technological and methodological approaches can be covered by adapter's solutions?
- Is it possible to model application portability?
- Is the application portability model feasible?
- Where can be applied the application portability model?
- Does the application portability model offers beneficial performances and time savings?

Scope

The scope of this thesis is:

- Analysis and evaluation of proposed Cloud computing interoperability models for all the layers of cloud computing service stack (IaaS, PaaS and SaaS);

- Research, analysis and adapter solutions modeling for part of common used cloud applications and
- Cloud applications portability modeling

Methods

To address the research questions from the area of cloud computing interoperability a comprehensive and in-depth analysis is performed of all proposed models and standards. Each model is analyzed by which layer of the service stack the model addresses, the current state of the documentation, the current state of deployment and the support by cloud providers. Each of these characteristics is used as an indicator for new evaluation framework and all the models are evaluated using this framework.

To address the research questions from the area of adapter models for services interoperability experiments to demonstrate interoperability are carried out on the created prototypes.

To address the research questions from the area of application portability model implementations of the model for different cloud framework are deployed in controlled testing environment described in [107] and each of the implementation is tested given number of times for different use cases in order to evaluate the time performance of the model.

Content of the Thesis

This thesis consists of seven chapters: 1) Basic concepts of cloud computing, interoperability and cloud interoperability and portability, 2) Overview, analysis and evaluation of proposed cloud computing interoperability models and standards, 3) Adapters as solution for SaaS interoperability modeling, 4) TOSCA based Portability Model for PaaS Hosted Applications, 5) Application and usage of the P-TOSCA model, 6) Evaluation results of P-TOSCA model and 7) Conclusions.

Chapter 1 presents the basic concepts and definitions for the purpose of this thesis. It describes the basic concepts and architecture of cloud computing, deployment models and service layers stack. It defines interoperability, describes types of interoperability and how it can be deployed. Finally it defines cloud computing interoperability and portability and describes use cases that cover those concepts.

Chapter 2 gives overview and analysis of proposed and published models for cloud interoperability. In this chapter an also an evaluation framework for cloud interoperability models is exposed.

Chapter 3 defines the concept of software adapters, gives overview of different API types, describes modeling of adapter solution for SaaS applications and shows adapter modeling in three examples.

Chapter 4 given overview of the TOSCA standard and identifies weakness of this specification. In this chapter a new model is described: P-TOSCA that represents an extension of the TOSCA standard. It also shows architecture for implementation of this model in form of Platform as a Service.

Chapter 5 describes the application and usage of the P-TOSCA model for different architectural styles. It also gives overview of application types where the model cannot be applied.

Chapter 6 shows the results from P-TOSCA model evaluation. It defines use cases and each of the use cases is evaluated in two cloud frameworks: OpenStack and Eucalyptus.

The final Chapter 7 gives detailed description of the obtained results, extensive description of the applicability of the results and presents future work.

Main Results

The results obtained within this PhD thesis research were published on reviewed conferences. The most important results are in the areas of evaluation cloud interoperability, implementation of adapter-approach cloud interoperability and modeling cloud portability. The main results are elaborated in the following sections.

Evaluation of proposed cloud computing interoperability models and standards

As a part of the research of this doctoral thesis a comprehensive and in-depth analyzes of proposed cloud computing interoperability models is performed [73, 65, 68]. The analysis includes the following models: Unified Cloud Interface/Cloud Broker, Enterprise Cloud Orchestration Platform / Orchestration layer, Blueprint for the Intercloud, Cloud Infrastructure Management Interface (CIMI) and Open Cloud Computing Interface (OCCI). Aside these models three more proposed standards are also included in the analysis: Cloud Data Management Interface (CDMI), IEEE P2301 (Cloud Profiles) and IEEE P2302 (Intercloud) [63]. In order to evaluate the models an evaluation frameworks introduced.The framework defines indicators for evaluation of suggested models. The indicators define numerical values for given different characteristics of the models like the status of the documentation, the status of the implementation, the support of the model by cloud providers and

the service layer that the model addresses. The approach used for the performed evaluation is based on previous evaluation frameworks experience [69, 103, 104, 52, 55, 109].

The evaluation results have shown highest score for the Open Cloud Computing Interface (OCCI) model, which already have implementations for some cloud platforms. This shown that the OCCI model will be most likely accepted as a standard by the cloud vendors. The Cloud Infrastructure Management Interface (CIMI) has also shown significant results but lacks more implementations for more cloud platforms. The Cloud Infrastructure Management Interface (CIMI) also lacks implementation. The Unified Cloud Interface/Cloud Broker and Enterprise Cloud Orchestration Platform / Orchestration layer models lack support. The proposed standards IEEE P2301 (Cloud Profiles) and IEEE P2302 (Intercloud) are still in development and haven't published draft documentations yet.

Adapter-approach modeling for SaaS interoperability

When is needed to enable collaboration among more services it often happens for the services to use different interfaces or to use different communication protocols. On commonly used approach in these situations is to use adapters in order to achieve semantic interoperability. In order to achieve SaaS interoperability for given domains we have performed analysis of the API types that are commonly used by the services and we suggest two types of modeling: using of the classical Adapter design pattern in cases when the services needed to communicate implement similar functionalities and data and use of extended Scalable adapter pattern when the services needed to communicate have partial overlap functionalities and data (i.e. part of the functionalities of a given service are also covered in a another service, while other part of the functionalities in other service and we need to provide interoperability of the given service with the two other services) [62]. Given this modeling recommendations and implementations for interoperability are developed for some representative cloud applications and domains.

The first implementation is the usage of simple software adapters to create interoperability of popular calendars services like Google Calendar, Microsoft Live and Facebook. For this process different calendar standards and provider APIs are analyzed. The created implementation offers integration and exchange of the given calendar services. The solution is realized as service that uses SOAP protocol and contains adapters for each of the services. The adapters communicate with the REST services of the providers by exchange of JSON messages. During the implementation process some provider's API have changed so some of the adapters were modified [74].

The second solution is the usage of software adapters to enable interoperability of storage services of popular providers like Dropbox, Google Drive and

OneDrive (ex. SkyDrive). This solution is implemented as web application that offers common interface for the storage services. The solution contains adapter that communicate with the provider's APIs using REST services and JSON messages. This solution offers a single interface point where the user can upload, update and delete files from any of the services as well as to migrate files between services [62].

The third study is recommendation for enabling interoperability among different university services. For that purpose an analysis of the used standards in this area is given and various use cases are described. The electronic services of Ss. Cyril and Methodius University and the Faculty of Computer Science and Engineering are used as a referent point, where adapters are used at to moment to enable collaboration among these systems. In order to enable partial interoperability usage of the following standards is suggested: SCORM and IMS Enterprise standards for University management systems, SCORM, QTI and IMS Enterprise standards for Learning management systems and SCORM and QTI standards for E- Assessment systems [71, 53].

Modeling portability for cloud applications

Various approaches for modeling cloud application portability like usage of standards, development of proprietary solutions and usage of semantic technology and ontologies are researched. The Topology and Orchestration Services for Applications (TOSCA) standard is used for further detailed analysis of the possibilities for topology definition [58]. A platform framework describing the required functionalities for TOSCA specification processing is described [72].

To examine the possibilities and opportunities of TOSCA for topological modeling a specification for the topology of the iKnow application (University management system) is created [60]. The iKnow system is chosen due to the complex and specific topology.

Extension of the TOSCA portability model and architecture to process the model

During the process of platform creation for implementation for the TOSCA model weaknesses and ambiguities of the standard are observed: lack of precise definition of the Properties element of the Node Template element and lack of definition of security policies and access rights. Also it is ambiguous to define plan with usage of BPMN and BPEL languages and by that this standard doesn't offers mechanism for proper application configuration and exchange of information between created nodes, like usage of the IP address

(dynamically allocated by the cloud) of one instance (node in the topology) for configuration properties of other instance (another node).

The new model, P-TOSCA represents an extension of TOSCA model with the following definitions:

- specifying the *external namespace* of ServerProperties
- specifying the *initial number of instances* child element to the ServerProperties element
- extending the ServerProperties element of Node Template with *ServerIP-Address* element
- specifying the *external namespace* of ScriptArtifactProperties
- extending the Properties element of Artifact Template with *InputParameters* element
- extending the Properties element of Node Template with *ServerSecurity-Properties* element
- introducing *XML defined plan*

To implement and test the model PaaS architecture is developed and implemented for two cloud frameworks: OpenStack and Eucalyptus. The architecture consists of three main modules:

- *Interface module* serves as interface to the client. It consists of a web application for direct communication with the user and web services that can be used by the user or other P-TOSCA PaaS instance.
- *P-TOSCA core module* processes and executes the specification. It contains P-TOSCA Model processor, Artifact executor, cloud communicator and Archive processor. The P-TOSCA module also uses external libraries.
- *Database module* contains history of user activities.

Using the implementations of the architecture a successful deployment is performed for several applications specified with the P-TOSCA model [70].

Modeling of N-Tier and SOA applications with P-TOSCA

The usage of the P-TOSCA model is researched for N-Tier and Service-Oriented Applications. For that reason specifications and archives are created and tested for applications with two-tier distributed architectures, applications with three-tier architectures, as well as service-oriented applications [59, 64, 67].

Evaluation of P-TOSCA portability model and P-TOSCA PaaS architecture implementations

In order to evaluate the time performance of the implementations of P-TOSCA model several use cases and their variations are created. The use cases include standard manual migration, initial cloud migration using the P-TOSCA model and migration of application from one cloud to another using the P-TOSCA model. For each of the use cases series of trials are performed and execution times are measured. The same process is executed in the cloud environments: OpenStack and Eucalyptus. For each use case the average execution time is calculated and a comparison of average time for the use cases is performed. The evaluation shown multiple time savings using the P-TOSCA model in comparison of manual migration [66]. Additionally a security evaluation is performed of the model and the implementation is performed given the previous security evaluations [105].

Applicability of the Results

Besides the important results that were mentioned previously, it is important to mention the applicability of these results.

Automated cloud applications portability

Usage of the P-TOSCA model and the implementation of the model for different cloud frameworks [70] provides automated cloud applications portability for applications deployed using the P-TOSCA PaaS instances.

Independence for cloud provider selection

The main contribution of the thesis is the independence for cloud provider selection, since the usage of the P-TOSCA model and implementations doesn't allow vendor lock-in and enables easy migration and vendor change for the hosted applications. In this process the challenges and risks of the proposed security architecture are taken into account [106] as well as the possibility to set information security management system [102].

Decrease of time needed for migration to cloud and change of cloud provider

Usage of the P-TOSCA model and its PaaS implementation offers significant reduction of time needed to migrate applications to cloud, as well as to migrate the application from one cloud provider to another. The performed experiments confirm multiple reduction of time [66].

Standardized application topology definition and operations for the topology in the cloud

Usage of this of the P-TOSCA is planned to create specification for scalable and elastic e-Assessment cloud solution [101]. The goal is to provide massive, scalable and elastic online courses for courses like Computer Architecture and Organization where the students expressed satisfaction for using e-Learning Tools [3]. The model will be also applied to other applications that are realized as a result of research [1, 61, 54]

Building adapters for SaaS interoperability

The defined methodology for enabling interoperability using the adapter-approach offers building adapters not only for SaaS services with similar functionalities (the standard adapter pattern approach), but also for SaaS services with only partial overlap of functionalities (the extended scalable adapter pattern approach) [62].

Skopje, *Magdalena Kostoska*
June 2014

Содржина

Слики

Табели

Алгоритми и кодови

Глава 1
Основни концепти

Преглед Пресметување во облак е воспоставен концепт за користење на сервиси кои нудат ресурсите за пресметувања. Овој концепт нуди две основни правила: „користи колку и кога имаш потреба“ и „плати само тоа што го користиш“ што им овозможува услуги на корисници кои не се во состојба или не сакаат да инвестираат во купување на сопствени компјутерски ресурси. Концептот на интероперабилност не е нов концепт. Овој концепт претставува способност два или повеќе хетерогени елементи да разменуваат информации меѓу себе и да ги употребуваат разменетите информации. Од друга страна концептот на интероперабилност на Пресметување во облак е релативно нов и е активна тема на истражување. Во оваа глава ќе се даде дефиниција и објаснување на сите наведени концепти.

1.1 Пресметување во облак

Оваа секција ги прикажува основните концепти за пресметување во облак.

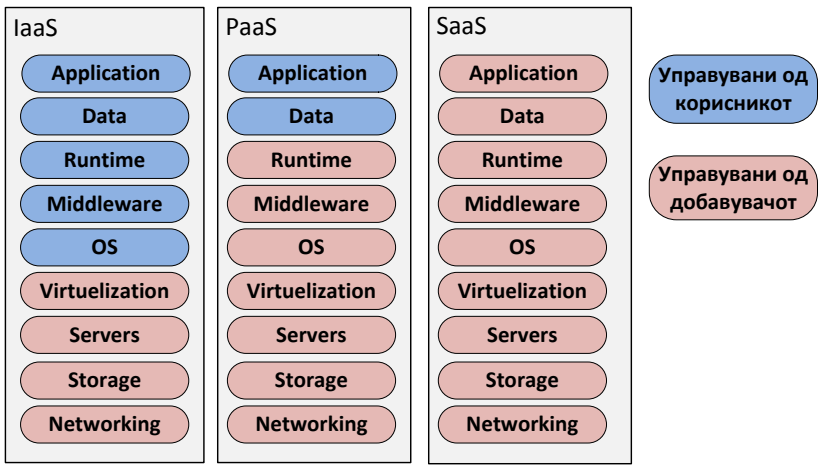
1.1.1 Што е пресметување во облак?

Според NIST Пресметување во облак е *"модел за овозможување на сеприсутен, удобен, и на-барање мрежен пристап до едно заедничко множество на компјутерски ресурси кои може да се конфигурираат (на пример, мрежи, сервери, складирање, апликации и услуги) и кои можат да бидат брзо овозможени и ослободени со минимални напор за управување или интеракција со давателот на услугата"*[87].

Пресметување во облак е составено од пет основни карактеристики: самопослужување на барање, широк мрежен пристап, содржи множество на ресурси, нуди еластичност и се врши мерење на сервисите.

1.1.2 Сервисни модели во облак

Услугите во облакот се достапни во три главни слоеви (сервиси): Инфраструктура како сервис (IaaS), Платформа како сервис (PaaS) и Софтвер како сервис (SaaS). Слика. 1.1 прикажува поедноставено објаснување на трите слоја.



Слика 1.1 Споредба на трите сервисни модели IaaS, PaaS и SaaS: Адаптирано од [54]

Дополнително во литературата се предложува воведување и на други слоеви како *Data Storage-as-a-Service* (DaaS) [41], *Communication as a Service* (CaaS) [41] итн.

1.1.3 Модели на поставување облак

Според дефинициите на NIST [87] постојат четири основни модели за поставување: 1) јавен, 2) приватен, 3) хибриден облак и 4) на заедница.

Кај приватниот модел на поставување на облак ресурсите на инфраструктурата ги користи само една организација. Организацијата не мора да ја поседува, управува или оперира инфраструктурата, тоа може да се прави и од страна на трето лице.

Кај јавниот модел на поставување на облак ресурсите на инфраструктурата се јавни и може да се користат од било кое лице или организација.

Кај хибридниот модел на поставување на облак претставува композиција од две или повеќе инфраструктури на облак (јавни и приватни).

Кај моделот на заедница ресурсите на инфраструктурата ги користи само една специфична заедница.

1.2 Интероперабилност

Интероперабилноста не може униформно да биде дефинирана и во литературата можат да се најдат многу различни дефиниции кои варираат од технолошки аспекти, рамки за развој кои можат да бидат широки или да се однесуваат само за некои детали до стандарди. Генерално дефиницијата на интероперабилност зависи од контекстот на примена.

Следуваат неколку генерални дефиниции на интероперабилност од повеќе организации (IEEE,NIST,UKOLN).

Дефиниција 1 *Интероперабилност е способност на систем или продукт да работи со други системи или продукти без дополнителен напор од страна на клиентот/потрошувачот. Интероперабилноста е возможна со примена на стандарди [43].*

Дефиниција 2 *Интероперабилност е способност на две или повеќе мрежи, системи, уреди, апликации или компоненти да разменуваат информации на безбеден и ефективен начин, без малку или никаква непријатност за клиентот и да можат веднаш да ги употребуваат разменетите информации [89].*

Дефиниција 3 *Интероперабилноста може да се разгледува како тековен процес во кој се обезбедува управувањето на дадени системи, процедури и култура на една организација да ги зголеми до максимум можностите за размена и повторна употреба на информации [112].*

Интероперабилноста вклучува многу области со свои карактеристики [112] :

- *Техничка интероперабилност* - развој на стандарди за комуникација, транспорт, и претставување.
- *Семантичка интероперабилност* - употребата на различни термини за да се опишат слични концепти може да предизвика проблеми во комуникација, извршување на програми и пренос на податоци.

- *Политичка/Хумана интероперабилност* - одлука да се направат ресурсите пошироко достапни носи импликации за организации, нивните вработени и крајните корисници.
- *Интероперабилност меѓу општества и заедници* - има зголемена потреба за пристап до информации од широк ранг на извори и заедници.
- *Интернационална интероперабилност* - кога се работи со други земји има варијации во стандарди, комуникациски проблеми, јазични бариери, разлики во комуникациски стилови и недостаток на заедничка основа.

1.2.1 Видови на интероперабилност

Во повеќето познати рамки за интероперабилност овој поим се разгледува на три нивоа [76]:

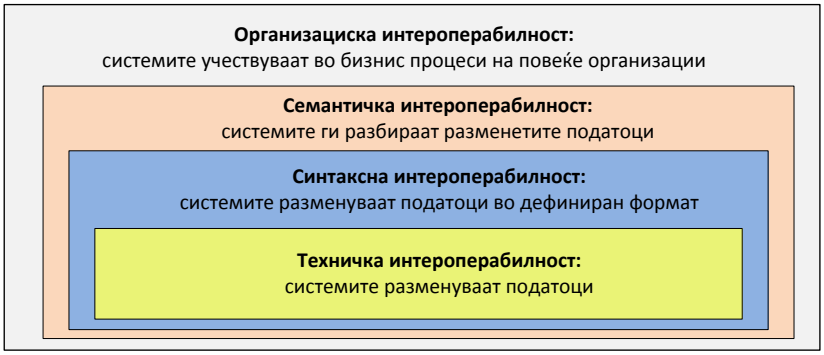
Техничка интероперабилност која вклучува стандарди и протоколи. Ова ниво *ги покрива техничките прашања за поврзување на компјутерски системи и сервиси. Ги вклучува главните аспекти како отворени интерфејси, интерконекциски сервиси ,интеграција на податоци и средишен(помошен) слој, преставување на податоци и размена, пристапност и безбедносни сервиси* [113]. Според ETSI техничката интероперабилност *вообичаено се асоцира со хардверски/софтверски компоненти, системи и платформи кои овозможуваат комуникација меѓу две машини. Овој вид на интероперабилност е често насочен кон (комуникациски) протоколи и инфраструктура која е потребна за овие протоколи да функционираат* [75].

Синтаксна интероперабилност која *е вообичаено асоцирана со форматите на податоците* [75].

Семантичка интероперабилност која *осигурува дека точното значење на разменетата информации е разбирливо од страна на било која друга апликација која не била иницијално развиена за таа намена. Семантичката интероперабилност овозможува системите да ги комбинираат добиените информации со други информациски ресурси и да ги обработат со соодветното значење* [113].

Дополнително со литературата се опишува и **организациска интероперабилност** според која се овозможува системите да учествуваат во бизнис процеси на повеќе организации. [113].

Приказ на сите типови на интероперабилност е даден на Слика. 1.2



Слика 1.2 Типови на интероперабилност и нивна поврзаност

1.2.2 Примена на интероперабилност

Примената на интероперабилноста во било кој домен најчесто со реализира со дефинирање и примена на стандарди. Генерално целта на интероперабилноста и стандардите е иста - да овозможат размена и соработка на компјутерски сервиси. Стандардите ја дефинираат протокол со чија примена се овозможува сите добавувачи на сервис кој го имплементираат стандардот да нудат исто структурирани податоци и ист начин на размена на информации, без разлика на внатрешната организација и начинот на имплементација на сервисот.

Од друга страна се случува доминантни понудувачи на сервиси да го наметнат својот протокол како де-факто стандард и да ги принудат останатите понудувачи да се подредат.

1.3 Интероперабилност и портабилност на Пресметување во облак

Интероперабилност на Пресметување во облак дозволува различни сервиси на различни облаци да обработуваат податоците со заедничка спецификација и дозволува едноставна размена и употреба на податоци меѓу различни инфраструктури на облак и употреба на разменетите податоци и сервиси за да можат ефективно да функционираат заедно. Од друга страна, **Портабилност на Пресметување во облак** дозволува едноставно употреба на податоци и сервиси од еден облак на друг за два или повеќе вида на инфраструктури на облак [88].

Табела 1.1 Дефиниција на интероперабилност и портабилност на Пресметување во облак за даден контекст и ниво

Активност	**Контекст**	**Ниво**
Интероперабилност	Управување	IaaS
	Платформа	PaaS
	Апликација	SaaS
Портабилност	Платформа	IaaS (само компоненти)
		IaaS (image)
	Апликација	PaaS
		IaaS
	Податоци	SaaS
		SaaS

Интероперабилност и портабилност на Пресметување во облак не се еднозначни и зависат од контекстот во кој се разгледуваат, па така побаруваат понатамошно додефинирање соодветно од примената. Табела. 1.1 дава преглед на различни нивоа на анализа.

1.3.1 Интероперабилност на Пресметување во облак

Интероперабилност во облак се разгледува и дефинира на ниво на сервисен модел и услуга, па така може да се разгледува интероперабилност на апликации, платформи и управување.

Интероперабилност на ниво на управување во облак се дефинира на ниво на Инфраструктура на сервис (IaaS) и подразбира едноставно и стандардизирано управување со инфраструктурите од различни типови на системи на облак. Управувањето вклучува инстанцирање и контрола на виртуелни машини, овозможување и откривање на мрежни карактеристики, поставување и менување на безбедносни правила итн...

Овој тип на интероперабилност е најдобро дефиниран. За овој тип е поставена основна таксономија на изрази прикажани на Слика. 1.3 [117, 98, 100]:

- Механизам за пристап - дефинира како еден сервис во облак може да се пристапи од страна на корисници и/или развивачи на софтвер,
- Виртуелни ресурси - достава на сервиси како комплетен софтверскиот стек од инсталација на виртуелна машина,
- Мрежа - адресирање и API,
- Складишта - управување и организација на складиштата,
- Безбедност - автентикација, авторизација, кориснички сметки и енкрипција,
- Service-Level Agreement - архитектура, формат, мониторирање и цели,

- Останато.

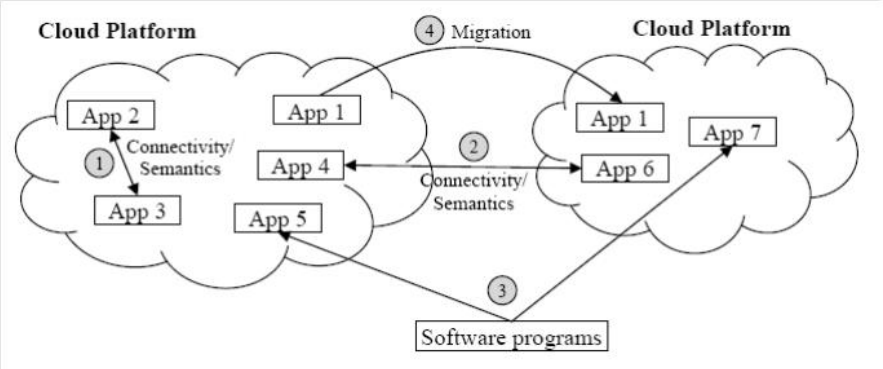
Слика 1.3 Таксономија на интероперабилност на IaaS: Адаптирано од [117]

Интероперабилност на ниво на платформи во облак се дефинира на ниво на Платформа на сервис (PaaS) и подразбира едноставна размена на податоци и сервиси меѓу различни платформи поставени на различни инфраструктури на облак и ефективно искористување на разменетите податоци без вложување дополнителен труд од страна на корисникот.

Интероперабилноста на ниво на апликации во облак се дефинира на ниво на Софтвер како сервис (SaaS) слојот и подразбира едноставна размена на податоци и сервиси меѓу различни апликации поставени на различни платформи и инфраструктури на облак и ефективно искористување на разменетите податоци без вложување дополнителен труд од страна на корисникот. Дополнително овој тип на интероперабилност може да се разгледува од страна на различни апликациски домени.

Интероперабилноста на ниво на апликации за прв пат се поставува од страна на Kumar и останати [77] во 2010. Според овој труд интероперабилноста на SaaS слојот може да биде разгледувана во четири категории (како што е прикажано на Слика 1.4):

- Интероперабилност меѓу апликации во ист облак,
- Размена на информации и повикување на операции на апликации низ различни околини за пресметување во облак,
- Софтверски програми кои поврзуваат повеќе различни околина на облак и интегрираат податоци и апликации низ облаците на унифициран начин и
- Миграција на апликација во облак од една околина на облак во друга.



Слика 1.4 Типови на интероперабилност меѓу апликации во облак [77]

1.3.2 Портабилност на Пресметување во облак

И портабилноста може да се разгледува и дефинира на ниво на сервисен модел и услуга, па така може да се разгледува портабилност на податоци, апликации и платформи.

Портабилност на податоци во облак може да се разгледува како повторно искористување на податоци:

- низ различни апликации во облак (SaaS) - односно како пренос и лесно повторно искористување на податоци од една во друга апликација во облак,или
- како пренос и повторно користење на податоци преку различни типови на складишта во облак (DaaS).

Портабилност на апликации во облак може да се разгледува како олеснето повторно искористување на апликациите низ:

- различни платформи (PaaS) - односно апликација развиена на една платформа (PaaS) да биде едноставно пренесена и повторно искористена на друга платформа, или
- инфраструктури (IaaS) во облак - односно апликација поставена на една инфраструктура (IaaS) на даден тип на систем на облак едноставно да биде пренесена и повторно искористена на друга инфраструктура и друг тип на систем на облак.

Портабилност на платформи во облак може да се разгледува како олеснето повторно искористување на платформите преку

- пренос на компонентите на платформата и нивна повторно употреба на друга инфраструктура (IaaS) или

Табела 1.2 Сценарио 1: Копирање на податочни објекти меѓу добавувачи на услуги во облак [4]

Актери	Клиент на облак, Добавувач на услуги во облак 1, Добавувач на услуги во облак 2, Транспортен агент
Цели	Копирање на податочни објекти од добавувач 1 во добавувач 1 на иницијатива на клиент
Сценарио	Клиентот се взаемно се автентицира со добавувачот 1 и започнува комуникација. Од добавувач 1 клиентот може да пристапи на други системи преки Интернет. Клиентот ги одредува идентификаторите на објектите ко сака да ги копира од добавувач 1 кај добавувач 2. Со користење на команди преку добавувач 1 клиентот се автентицира кај добавувач 2. Клиентот ја одредува локацијата кај добавувачот 2 каде што треба да се копираат објектите (може ќе треба да креира нова локација). За секој објект што треба да се копира: го симнува објектот на виртуелна машина на добавувачот 1, го праќа кај добавувачот 2, ги брише копираните податоци од виртуелната машина кај добавувачот 1.

- целосен пренос (image) на платформата на друга инфраструктура (IaaS) во облак.

1.3.3 Сценарија за интероперабилност и портабилност на Пресметување во облак

Како дел од соработката на NIST со Standards Acceleration to Jumpstart Adoption of Cloud Computing (SAJACC) Bager и останати дефинираат пет сценарија за интероперабилност и портабилност на Пресметување во облак[4]:

1. Копирање на податочни објекти меѓу добавувачи на услуги во облак, прикажано во Табела. 1.2
2. Динамичко праќање и распоредување на операции на IaaS облак, прикажано во Табела. 1.3
3. Трансформација на нов облак од податочниот центар во јавен облак, прикажано во Табела. 1.4
4. Мигрирање на апликации кои користат редици на чекање, прикажано во Табела. 1.5
5. Мигрирање на (стопирана) виртуелна машина од еден облак на друг, прикажано во Табела. 1.6

Табела 1.3 Сценарио 2: Динамичко праќање и распоредување на операции на IaaS облак [4]

Актери	Клиент на облак, Добавувач на услуги во облак 1, Добавувач на услуги во облак 2, ..., Добавувач на услуги во облак n
Цели	Повикување на операции на најефективните облаци одредени од страна на множество на правила на клиентот кои се евалуираат во време на извршување
Сценарио	Клиентот сака да изврши задача на облак кој нуди најдобри перформанси, има најмала цена и е најдоверлив. Од моментот кога клиентот отвара сметка кај секој добавувавач има записи за цените на услугите и ветените перформанси и достапност. Клиентот формира мала задача, ја извршува еднаш или повеќе пати на секој добавувач и ги сортира добавувачите според достапност, точност на излезот на задачата и перформансите.

Табела 1.4 Сценарио 3: Трансформација на нов облак од податочниот центар во јавен облак [4]

Актери	Клиент на облак, Добавувачи на услуги во облак, Брокер за управување со облак
Цели	Одржување на побарувaните нивоа на услуги за хостиран процес на податочен центар на агенција со динамичко алоцирање/деалоцирање пресметувачки или ресурси за складирање според моменталните побарувања за услуга
Сценарио 1	Клиентот подготвува и одржува виртуелни машини или складишта со дадена големина кај добавувачот дизајнирани за да ги задоволат потребите на клиентот. Клиентот поставува процес за мониторирање на оптовареноста на ресурсите поставува одредена граница за трансформација (bursting). Горната граница управува со започнување на нов процес кај добавувачот за да се справи со зголемената оптовареност. Долната граница управува со процес за справување со намалена оптовареност.
Сценарио 2	Клиентот рачно алоцира и деалоцира ресурси според известувањата за оптовареност.
Сценарио 2	Брокерот за управување со облак ја мониторира оптовареноста и границата и алоцира и деалоцира ресурси на добавувачот со употреба на интерфејс овозможен од добавувачот.

Табела 1.5 Сценарио 4: Мигрирање на апликации кои користат редици на чекање [4]

Актери	Клиент на облак, Добавувачи на услуги во облак 1, Добавувачи на услуги во облак 2, Брокер за управување со облак
Цели	Миграција на постоечки редици и нивни пораки од еден добавувач на друг
Сценарио	Клиентот сака да мигрира редица и нејзините пораки од добавувач 1 кај добавувач 2. И двата добавувачи имаат договорено минимално множество на атрибути на пораки, атрибути на редици и операции над редици за да може да се извршат миграциски активности. Клиентот издава команда на брокерот да ја мигрира редица X од добавувач 1 во редица Y кај добавувач 2. Брокерот издава команда на добавувачот 2 преку неговото API да креира редица Y. Брокерот издава команда на добавувачот 1 преку неговото API да ја стопира редицата X. Брокерот издава команда на добавувачот 1 да пристапи до пораките од редица X и команда на добавувачот 2 да креира идентични објекти во редицата Y според договорот. Добавувачот 2 издава команда за старт на редицата Y и го известува клиентот.

Табела 1.6 Сценарио 5: Мигрирање на (стопирана) виртуелна машина од еден облак на друг [4]

Актери	Клиент на облак, Добавувачи на услуги во облак 1, Добавувачи на услуги во облак 2, Брокер за управување со облак
Цели	Да се мигрира произволна виртуелна машина без проблем од добавувач 1 на добавувач 2
Сценарио	Клиентот издава команда да се стопира виртуелната машина (ВМ) кај добавувачот 1. Добавувачот 1 креира конфигурациски документ за стопираната ВМ кој вклучува апстрактен опис на хардверот употребуван од ВМ. Клиентот го доставува конфигурацискиот документ до добавувачот 2 и побарува соодветна околина и притоа по потреба доставува дополнителни дефиниции кои добавувач 2 не може да ги идентификува од конфигурацискиот документ и ако складиштето на ВМ е перзистентно ги копира податоците на складиште на добавувачот 2. Ако не е перзистентно се справува на поинаков начин. Клиентот издава команда на добавувачот 2 да ја стартува ВМ.

Глава 2
Преглед, анализа и евалуација на предложени модели и стандарди за интероперабилност

Преглед Интероперабилност на пресметување во облак (Cloud Computing Interoperability) сервисите е атрактивна тема за истражување во која се ангажирани голем број на научници, организации, работни групи и овозможувачи на услуги во облак. Објавени се многу различни пристапи и можни решенија, но не постојат многу прифатени стандарди или модели досега. Оваа глава претставува преглед на најважителните објавени модели за интероперабилност на пресметување во облак и се дискутираат на нивните можности и предизвици. Презентираниот заклучок се однесува на растечкиот тренд во користењето на пресметување во облак (cloud computing) и недостигот од видливи резултат да се постигне интероперабилност.

2.1 Вовед

Постојат различни перцепции на терминот интероперабилност на пресметување во облак дефинирани од различни точки на погледи [92] [36] [95].

Овој поим може да биде наведен како способноста на апликациите да работат во различни облаците да ги споделат податоци, пренос на апликации на друго решение за пресметување облак или можноста апликациите да имаат иста функционалност и опции во различни платформи или решенија за пресметување во облак. Исто така не се исклучени преносливоста на податоците, управување и миграција меѓу различните решенија на пресметување во облак. Интероперабилност може да се дефинира на секое слој на сервиси на пресметување во облак: Инфраструктура како сервис (IaaS), Платформа како сервис (PaaS) и Софтвер како сервис (SaaS)

Во оваа глава ќе биде употребена дефиницијата дадена од Enterprise Interoperability Science Base (EISB) речникот [5].

Дефиниција 4 (Интероперабилност на пресметување во облак) *Интероперабилност на сервиси за пресметување во облак е способноста на сервисите да бидат во можност да работат заедно со различни облак услуги и провајдери и други апликации или платформи кои не се зависни од облак. [5]*

[94] е едно од првите истражувања кој сугерира потреба за интероперабилност на пресметување во облак и опишува можните сценарија за овозможување на интероперабилност. Како што пресметувањето во облак станува се повеќе широко користена технологија, интероперабилноста е анализирана од повеќе истражувачки заедници. Сепак, не постои единствено решение.

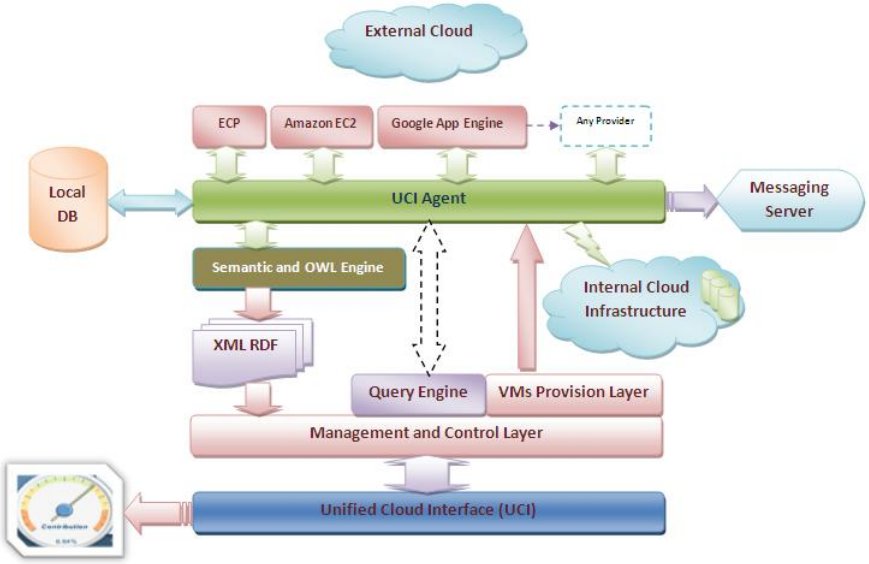
2.2 Преглед на модели и предлог стандарди за интероперабилност на пресметување во облак

Различни модели целат на различни сервиси од сервисниот стек на облакот: [30] се однесува на сите слоеви во облакот, додека [11], [6] и [16] целат само на IaaS слојот. Во продолжение ќе ги опишеме овие модели.

2.2.1 Унифициран интерфејс за облак / Брокер за облак (UCI)

Проектот за Унифициран интерфејс за облак (The Unified Cloud Interface Project) има за цел да креира отворен и стандардизиран интерфејс за облак за различни апликациски интерфејси на провајдери на облак [30]. Овој модел на Унифициран интерфејс за облак (брокер за облак) е дискутиран во [94]. Основната идеја е "*да се направи апстракциски слој кој е независен од било кој апликациски интерфејс на провајдер на облак, платформа или инфраструктура*". Унифицираниот интерфејс за облак треба да се создаде апликациски интерфејс за други интерфејси на облак, да послужи како заеднички интерфејс, да обезбеди спецификација и шема за интеграција со други модели на управување и размена на информации за управување и се однесува на Инфраструктура како сервис (IaaS) и Платформа како сервис (PaaS) слоевите. Овој модел сугерира користење на семантички веб технологии и јазици за веб онтологии. Преглед на моделот е прикажан на Слика 2.1.

Овој пристар е предложен од страна на непрофитната организација Cloud Computing Interoperability Forum (CCIF). За жал некои од најголемите компании го отфрлија CCIF пристапот, па малку е веројатно дека овој модел ќе биде широко користени.

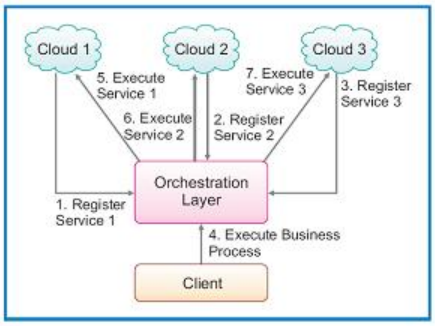


Слика 2.1 Архитектура на Унифициран интерфејс за облак [30]

2.2.2 Платформа за оркестрација на облак / Оркестрациски слој (Cloud Orchestration)

Извор на идејата за Платформа за оркестрација на облак(Enterprise Cloud Orchestration Platform) се базира на идејата Интернет да се презентира како мрежа на мрежи. Во овој модел различни облак провајдери може да ги регистрираат своите услуги во облак во рамките на слојот за оркестрација, слично на објавување на веб сервиси со Universal Description, Discovery and Integration (UDDI) [11].[94] сугерира " textit Оркестрацискиот слој може динамички да избере и поврзе сервиси врз основа на критериуми / алгоритми кои ги утврдуваат најдобрите сервиси во облак за одредена работа врз основа на фактори како највисоки перформанси, најниската цена или други барања како што е наведено од страна на клиентот ". Пример за повикување на три различни услуги обезбедени од различни провајдери во облак е прикажан на Слика 2.2.

Покрај стандардните безбедносни прашања за пресметување во облак, овој модел има многу фактори кои треба да се решат: ограничена поддршка на платформа на бараната услуга, справување со доцнења и латентност кои се должат на големите побарувања за перформанси на оркестрацискиот слој и големите трошоци за транспорт на големи податоци.



Слика 2.2 Оркестрација на облак [94]

За жал овој модел е, исто така, не се прифаќаат од страна на највлијателните провајдери на услуги во облак.

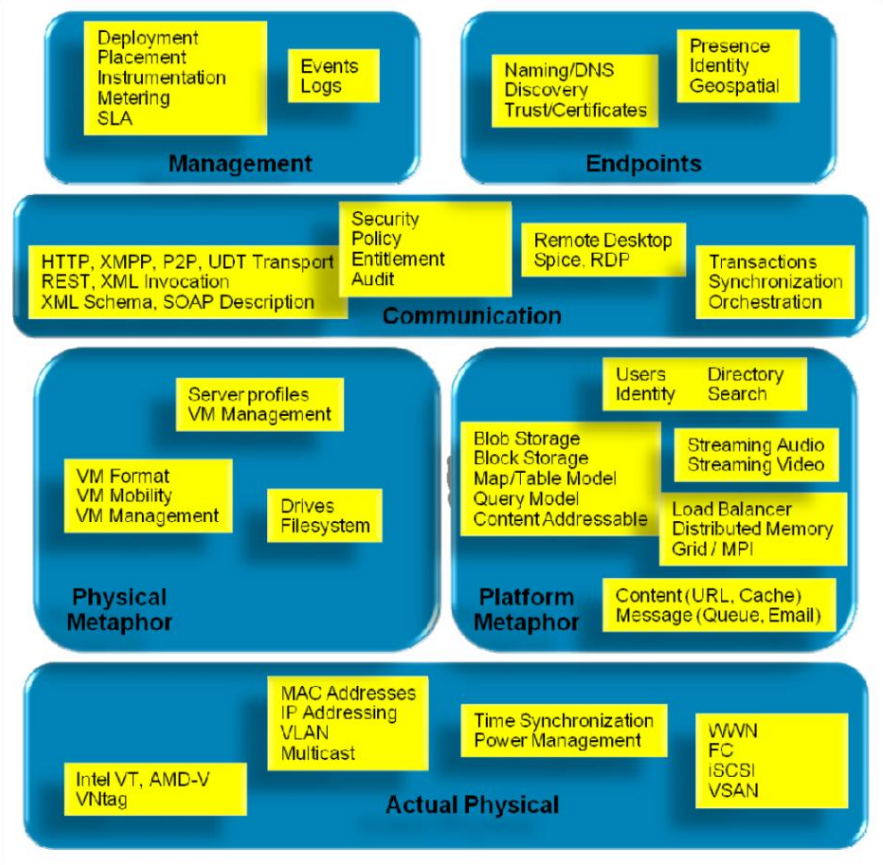
2.2.3 Нацрт за Интероблак (InterCloud)

Решението InterCloud преставува федерација на облаци [94]. [6] разгледува различни сценарија на интероперабилност на пресметување во облак вклучувајќи ги и интероперабилност, интер-облак протоколи и формати за овозможување на сценаријата. Опишани се два типа на сценарија: сценарија кои вклучуваат употреба на физичка метафора и употреба на апстрактен метафора. Физичката метафора вклучува сервери, дискови, мрежни сегменти, итн..., додека апстрактната метафора вклучува функции за складирање, редици за пораки (message queue), е-мејл функции, multicast функции, итн...

Првиот тип на сценарија е за инстанцирање и мобилност на виртуелни машини (ВМ). Тие вклучуваат трансакции за мобилност на ВМ, безбедна комуникација меѓу облаците, транспорт на ВМ и формати за инстанцирање на ВМ.

Вториот тип на сценарија е за интероперабилност и федерацијата на податочни складови. Тоа вклучува пренос на податочни складови од еден облак провајдер на друг и се претпоставува безбедна комуникација и транспорт меѓу облаците [6].

[6] сугерира употреба на два вида на облак, еден користење хипервизори од VMware и друг користење на хипервизори со отворен код како Xen и KVM од RedHat. "Интероблак протоколите"се тестирани на овие облаци. Слика 2.3 ја покажува архитектурата на Интероблак стандардите. Предмет на понатамошни истражувања и проширување се множества на веќе воспоставените протоколите.



Слика 2.3 Архитектура за Интероблак стандарди [6]

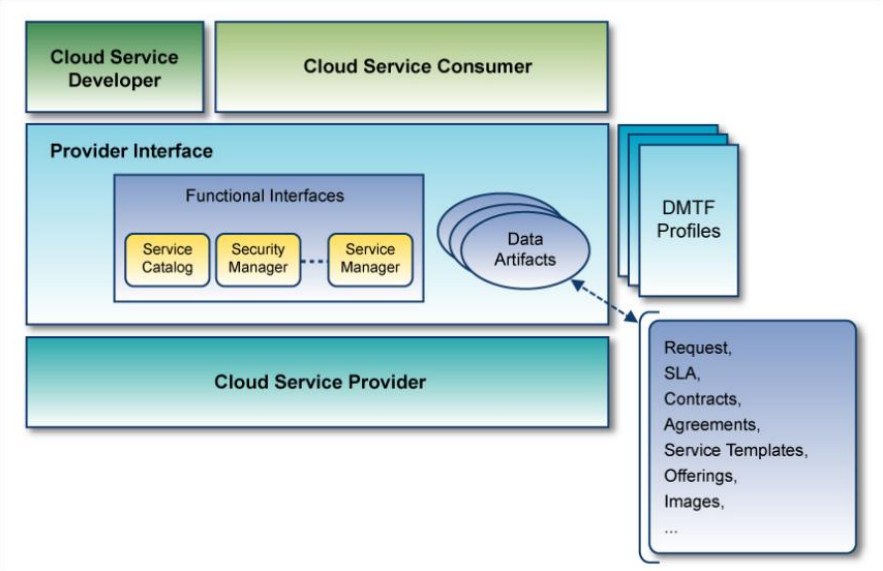
Ова истражување покажува само насоки за понатамошна стандардизација и тоа е само појдовна точка. Тоа бара многу на идната работа и комуникација помеѓу постоечките провајдери на облак.

2.2.4 Интерфејс за управување со инфраструктура на облак (CIMI)

Во 2009 година Distributed Task Management Force е формираше група посветена на одговори на потребата за отворени стандарди за управување за пресметување во облак. Групата е наречена "Инкубаторот за Отворени Стандарди во Облак"(Open Cloud Standards Incubator) и нивната цел е да се развие множество на информативни спецификации за управување со

ресурсите во облак [16]. Нивната цел е IaaS интероперабилност. Работата на оваа група е ветувачка бидејќи некои од најголемите понудувачи на услуги во облак како AMD, Cisco, Citrix, EMC, HP, IBM, Intel, Microsoft, Novell, Red Hat, Savvis, Sun Microsystems, и VMware се дел од оваа група.

Од 2009 до 2010 година оваа група има создадено документи за нивната визија за интероперабилни облаци [17], архитектура за управување со облак [18] и сценарија и описи на интеракции за управување со облак [26]. Оттогаш тие работат на Интерфејс за управување со инфраструктура на облак (Cloud Infrastructure Management Interface - CIMI) моделот.



Слика 2.4 Референтна архитектура за сервиси во облак [17]

Оваа група во своите документи ги формализира предизвиците за управување со облак, дава преглед на сценарија за интеракција со користење интероперабилни стандарди за облак, дефинирани животен циклус на услуга во облак и создаде Референтна архитектура за сервиси во облак [17]. Слика 2.4 ја прикажува предложената Референтна архитектура за сервиси во облак.

Овој стандард дефинира логички модел за управување со ресурси во IaaS слојот. Моделот користи HTTP RESTfull протокол за управување со интеракцијата помеѓу давателите на сервиси и корисниците. Пораките што се форматирани со користење или Java Script Object Notation (JSON) или eXtensible Markup Language (XML). Неговата цел е да им овозможи преносливост меѓу облак имплементации кои ја поддржуваат овој спецификација и да се овозможи интероперабилност помеѓу потрошувачот и повеќе провајдери [17].

Подетална дефиниција на предложената референтна архитектура вклучува модели на ресурс, безбедносна архитектура, барања на високо ниво и примери за протоколи [18]. Оваа група, исто така, дефинира сценарија за управување со животниот циклус на облак услуги и податочни артефакти кои се користат во наведените случаи [26].

CIMI ги моделира основните ресурси како машини, складишта и мрежи и нивниот домен во еден облак со поделена администрација. секој од ресурсите се претставува со множество на клуч/вредност парови со што се дефинира конфигурацијата, операциите или односите меѓу ресурсите. Употребува различни типови на парови како boolean, dateTime, duration, integer итн. Ресурсите се организирани во следните каталози:

- Влезна точка на облак (Cloud Entry Point) - влезна точка за наоѓање ресурси и можности,
- Ресурси на машини (Machine Resources) - ресурси за пресметувачка инфраструктура,
- Ресурси на волумени (Volume Resources) - ресурси на инфраструктура за складови,
- Мрежни ресурси (Network Resources) - ресурси за мрежна инфраструктура,
- Системски ресурси (System Resources) - односи меѓу машини, складови и мрежа и
- Ресурси за надгледување (Monitoring Resources) - следење на развојот на операциите, мерење и следење на статусот на други ресурси.

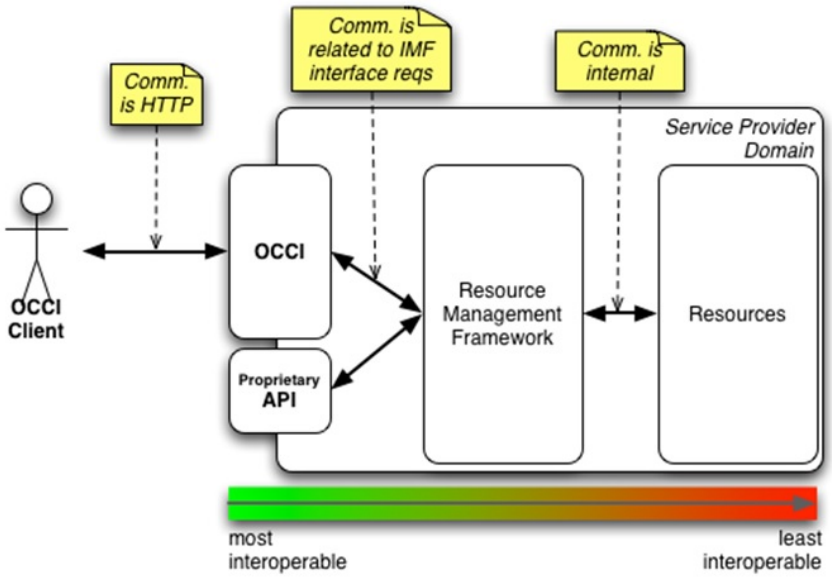
Работата на оваа група е се уште во тек, развој тече бавно.

2.2.5 Отворен интерфејс за пресметување во облак (OCCI)

Отворениот интерфејс за пресметување во облак (Open Cloud Computing Interface - OCCI) е развиен од страна на Open Grid Forum. Овој протокол претставува API за сите видови на задачи за управување. Главната цел на овој стандард е иста како CIMI - да се овозможи интероперабилност. OCCI претставува кориснички сервис кој нуди пристап до внатрешниот управувачкиот слој на провајдерот. Првично ова API е наменето "за развој на интероперабилни алатки за заеднички задачи, вклучувајќи распоредување, автономниот скалирање и следење". Идејата на овој стандард е да претставува унифицирано API за управување за сите провајдери. Слика 2.5 го покажува поставувањето на OCCI во инфраструктурата на провајдерот [93].

Основниот модел на OCCI го дефинира начинот на кој ресурси се претставени и како тие може да се манипулираат преку имплементација на OCCI за рендерирање и како може да бидат претставени од страна UML

дијаграм на класа. Ресурс може да биде виртуелна машина, корисникот, итн.



Слика 2.5 Поставување на OCCI во инфраструктурата на провајдерот [93]

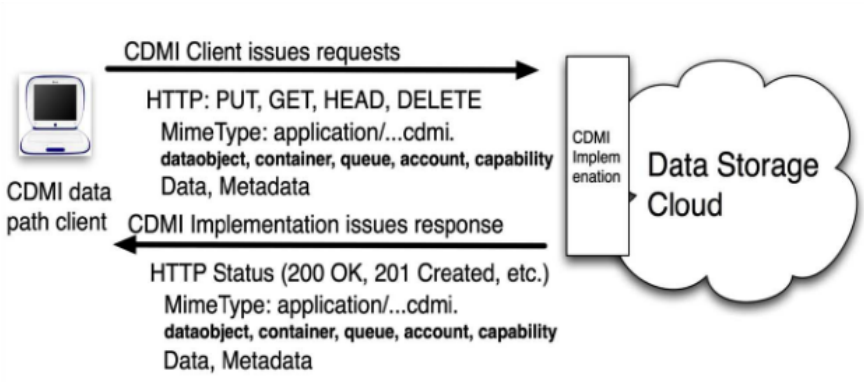
Овој протокол исто така ја опишува OCCI инфраструктурата, која покажува како имплементација на OCCI може да се моделира и имплементира IaaS API-а за создавање и управување со ресурсите. Дополнително може да се врши интеракција со основниот модел преку RESTful OCCI API.

Овој протокол веќе нуди имплементации за OpenStack, OpenNebula, Eucalyptus и др.

2.2.6 Интерфејс за управување со податоци во облак (CDMI)

Интерфејс за управување со податоци во облак (Cloud Data Management Interface - CDMI) е развиен од страна на Storage Networking Industry Association (SNIA) и сега претставува ISO стандард (ISO/IEC 17826:2012) [110]. Овој протокол дефинира функционален интерфејс кој апликациите ќе можат да го користат за да креираат, достапат, ажурираат и бришат податочни елементи во облак, т.е. овозможува управување со Податоч-

ни складишта како сервис (Data Storage-as-a-Service - DaaS) слојот во облак. Може да биде употребен и за управување со домени, безбедносен пристап, информации за употреба и наплата итн... Користи RESTful проколок (кога е возможно) како интерфејс. Користи и мета-податоци за управување со големи количини на податоци со различни побарувања. Слика 2.6 го покажува го прикажува основниот тек на податоци со користење на овој протокол.



Слика 2.6 Размена на податоци со користење на CDMI [110]

2.2.7 IEEE P2301 (Профили за облак)

Овој стандард е во процес на изработка и неговата цел е да се развие водич за профили на портабилност и интероперабилност за облак. Идејата е да советува учесници во екосистемот на облак (како понудувачи на системи за облак, понудувачи на услуги во облак и корисниците) за избор на стандарди и понуди во области како што се интерфејси на апликации, интерфејси за преносливост, интерфејси за управување, интерфејси за интероперабилност, формати на датотеки и конвенции за операции. Овој стандард треба да креира повеќе логички профили за да прикажат различни особености на облак [44].

2.2.8 IEEE P2302 (Интероблак)

Овој стандард е во процес на изработка и неговата цел е да дефинира топологија, функции и управување на интероперабилност меѓу облаци

Табела 2.1 Преглед на клучни карактеристики на предложени модели и стандарди за интероперабилност

Модел/Предлог стандард	**Сервисен модел**	**Статус на документација**	**Статус на реализација**	**Поддршка**
Брокер за облак	IaaS	Предлог документација	Демо реализација со Amazon EC2 и Enomaly ECP	Нема поддршка
Оркестрациски слој	IaaS, PaaS, SaaS	Достапна одобрена документација	Реализација од страна на Cordys, RightScale и CSC	Нема поддршка
Интероблак	IaaS	Нема документација	Нема	Нема поддршка
Интерфејс за управување со инфраструктура на облак	IaaS	Предлог документација	Нема	Има поддршка
Отворен интерфејс за пресметување во облак	IaaS	Достапна одобрена документација	Реализација со OpenStack, OpenNebula, Eucaliptus	Има поддршка
Интерфејс за управување со податоци во облак	DaaS	Достапна одобрена документација	Реализација за Amazon S3	ISO стандард
IEEE P2301 (Профили за облак)	N/A	Во изработка	Нема	N/A
IEEE P2302 (Интероблак)	IaaS	Во изработка	Нема	N/A

и федерација. Тополошките елементи вклучуваат облаци, размени (кои посредуваат меѓу различни управувања на облаци) и gateways (кои посредуваат во размена на податоци меѓу облаци). Функционалните елементи вклучуваат namespaces, пораки, онтологии за ресурси и инфраструктура за доверба. Елементите за управување вклучуваат регистрација, независност од локација, итн... Не е цел на овој стандард да ги разгледува внатрешните операции на облак ниту дополнителни хибридни имплементации на облак [45].

2.3 Евалуација

Во овој дел се врши евалуација на сите дадени модели и предлог стандарди. Табела. 2.1 нуди преглед на најбитните карактеристики на опишаните решенија.

2.3.1 Методологија на евалуацијата

Методологијата која ја воведуваме употребува категоризација според сервисниот модел и домени на статусот на: документација, реализација и поддршка. За да може да се изврши споредба и евалуација на предлог моделите и стандардите за интероперабилност се воведува оценување на секоја од категориите. Оцените варираат од 0 - која претставува најлоша оцена, до 3 - која преставува најдобра оцена. Оваа склала на оценување се воведува бидејќи сите предложени категории во овој момент содржат по четири типа на вредности кои може да се мапираат во оваа скала. Со понатамошен развој на предложените модели и нивно постепено прифаќање би се вовеле и повисоки оцени како 4.

Во првата категорија на оваа методологија се оценува на за кој *сервисен модел* се однесува моделот/стандардот, па така предлозите кои се однесуваат на најниското ниво од сервисниот модел се оценуваат со најголема оцена 3, а тие на највисоко ниво со најмала оцена - 1 (поради зависноста на погорните ниво од подолните). Дополнително тие кои се протегаат на повеќе нивоа ја добиваат оцената според најниското ниво кое го вклучуваат.

Во втората категорија на оваа методологија се оценува *статусот на документацијата*, па така:

- Достапна одобрена документација се оценува со 3,
- Предлог документација се оценува со 2,
- Документација во изработка се оценува со 1,
- Отсуство на документација се оценува со 0.

Во третата категорија на оваа методологија се оценува *статусот на реализацијата*, па така:

- Реализација од страна на најчесто користени понудувачи на услуги во облак се оценува со 3,
- Реализација од страна на понудувачи на услуги во облак се оценува со 2,
- Демо реализација се оценува со 1,
- Отсуство на реализација се оценува со 0.

Во третата категорија на оваа методологија се оценува *поддршката* на предлог моделот/стандардот, па така:

- Поддршка од страна на најчесто користени понудувачи на услуги во облак се оценува со 3,
- Поддршка од страна на понудувачи на услуги во облак се оценува со 2,
- Поддршка од страна на само еден или два понудувачи на услуги во облак се оценува со 1,
- Отсуство на поддршка се оценува со 0.

Табела 2.2 Резултати од евалуација на предложени модели и стандарди за интероперабилност

Модел/Предлог стандард	**Сервисен модел**	**Документација**	**Реализација**	**Поддршка**	**Вкупно**
Брокер за облак	3	2	1	0	6
Оркестрациски слој	3	3	2	0	8
Интероблак	3	0	0	0	3
Интерфејс за управување со инфраструктура на облак	3	2	0	3	8
Отворен интерфејс за пресметување во облак	3	3	3	3	12
Интерфејс за управување со податоци во облак	2	3	2	2	9
IEEE P2301 (Профили за облак)	0	1	0	0	1
IEEE P2302 (Интероблак)	3	1	0	0	1

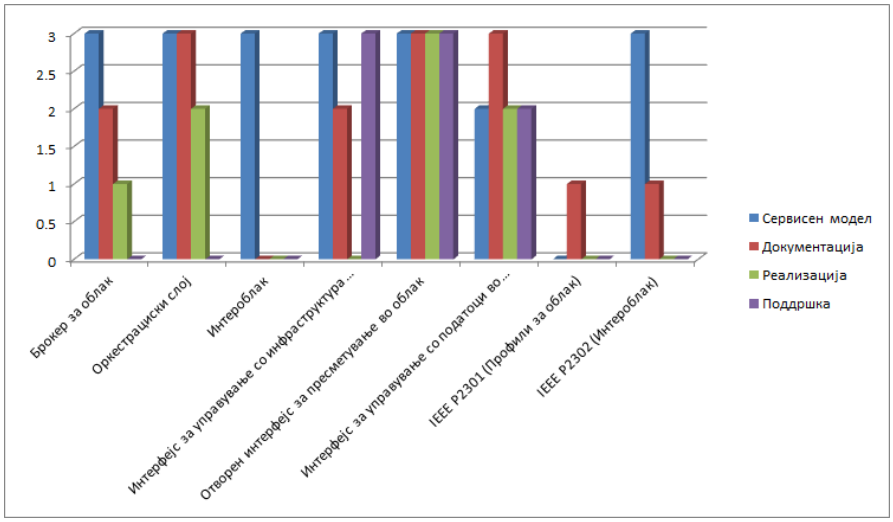
2.3.2 Резултати

За секој од дадените предлози за модел или стандард е извршено оценување според опишаната методологија. Резултатите по секоја категорија посебно и збирните резултати се прикажани во Табела. 2.2.

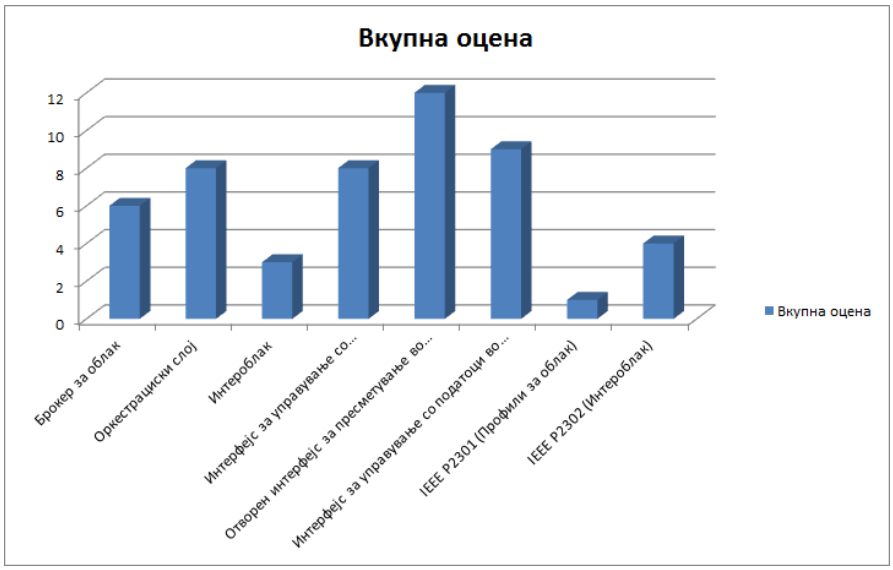
Столбестиот дијаграм на Слика. 2.7 дава споредба на оцените на секој од предложените модели/стандарди по оценуваните категории. Од дијаграмот јасно се забележува дека Отворен интерфејс за пресметување во облак има највисоки оцени во сите можни категории. Дополнително се забележува дека Интерфејс за управување со инфраструктура на облак е на добар пат и преостанува последната фаза - имплементација, додека на Оркестрациски слој моделот без разлика на документацијата и имплементацијата му недостасува поддршка. Столбестиот дијаграм на Слика. 2.8 дава споредба на вкупните оцени на секој од предложените модели/стандарди.

2.4 Главен придонес

Користење на пресметување во облак се зголемува, особено во областа на инфраструктура и софтвер во облак како сервис (IaaS и SaaS). Постојат неколку пристапи и истражувачки иницијативи тврдејќи напредок во



Слика 2.7 Споредба на оцените на секој од предложените модели/стандарди по оценуваните категории



Слика 2.8 Споредба на вкупните оцени на секој од предложените модели/стандарди

создавање на интероперабилност во областа на IaaS. Меѓутоа не постои доказ за напредок за поставување на интероперабилност во областа на останатите слоеви (PaaS и SaaS).

Два од предложените модели на ниво на IaaS (OCCI и CIMI) веќе имаат свои имплементации, како и поддршка од дел од понудувачите на услуги во облак и постои голема веројатно да бидат прифатени како стандарди од повеќето понудувачи на услуги во облак. Овие два модели воедно имаат најдобар резултат од евалуацијата.

Глава 3
Адаптери за моделирање интероперабилност на Софтвер како сервис

Преглед Во оваа глава се разгледува концептот на воведување на софтверски адаптери како средство за моделирање интероперабилност на Софтвер како сервис во облак. Се дава преглед на концептот на софтверски адаптер и се дава преглед на користење на програмирачки интерфејси на апликации (APIa). Се прикажува примена на два софтверски шаблона за изработка на софтверски адаптери за овозможување семантичка и организациска интероперабилност на Софтвер како сервис во облак. Примената на моделот се покажува врз познати типови на Софтвер како сервис.

3.1 Вовед

Концептот на софтверски адаптер не е нов и многу често се применуваат во реалните апликации. Во оваа глава се разгледува концептот на воведување на софтверски адаптери за моделирање интероперабилност на Софтвер како сервис.

Кога е потребно да се овозможи соработка и интероперабилност меѓу повеќе сервиси често се случува интеракциите меѓу сервисите да не се компатибилни. Возможно е сервисите да имаат различни интерфејси (signature incompatibilities) или пак различна логика за комуникација (protocol incompatibilities) [24]. Овие некомпатибилности не дозволуваат семантичка интероперабилност меѓу сервисите. Генерален проблем е тоа што секој производител се обидува да наметне свој стандард, наместо да се воведе стандард кој сите би го почитувале. Затоа и воведувањето на адаптер како решение овозможува соработка на сервиси од различни производители. Еден често користен пристап за справување со вакви ситуации е употребата на адаптери кои ќе вршат конверзија на барања во соодветен побаруван формат.

Понатаму во оваа глава се дава преглед на користење на програмирачки интерфејси на апликации (APIа), кои дефинираат интерфејси за користење на дадени сервиси. Се прикажува употреба на два софтверски шаблони: адаптер шаблон и проширен скалабилен адаптер шаблон за моделирање на семантичка и организациска интероперабилност на Софтвер како сервис во облак. Се прикажува примената на шаблоните врз познати типови на Софтвер како сервис.

3.2 Концепт на програмирачки интерфејси на апликација (API)

Програмирачки интерфејс на апликација (API) претставува спецификација дадена од производителот на софтверот и ја дефинира интеракцијата на софтверот со други алатки или софтвери. Изработката и понудата на API-то преставува желба и одговорност на производителот.

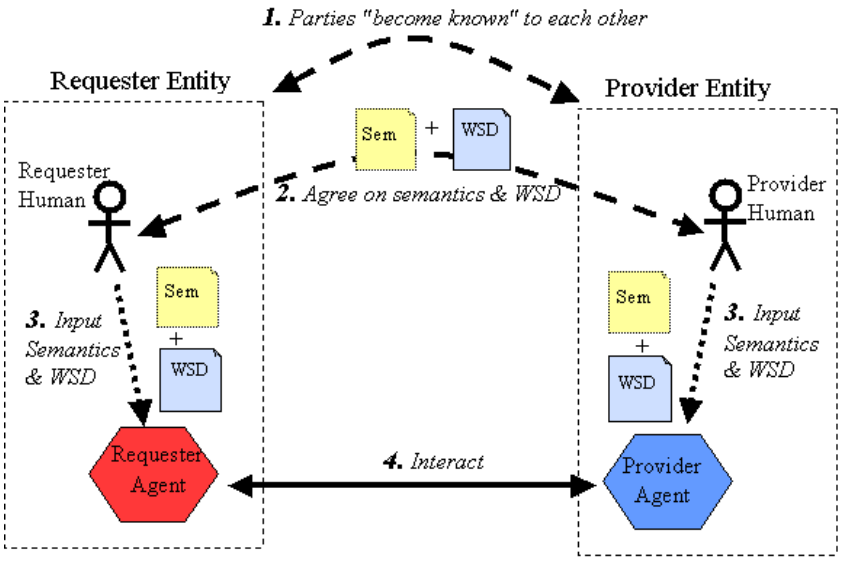
Програмирачките интерфејси на апликациите можат да се најдат во многу различни форми како стандарди, библиотеки или документации на производителот. Генерално постојат неколку типа:

- Веб сервиси,
- Веб сокети,
- Библиотеки и
- Повик на оддалечен објект.

Веб сервиси:

Веб сервисите претставуваат софтвер кој овозможува комуникација со останати машини преку мрежа со размена на пораки и користење на HTTP (или пак HTTPS) протоколот. За негово идентифицирање се користи URI (Uniform Resource Identifier). Форматот за размена на податоци е предефиниран и до тоа се овозможува соработка. При комуникација со веб сервиси се доставува т.н. барање (HTTP request) и сервисот враќа соодветен одговор (HTTP response). Самата структура на барање и одговор е предефинира и стандардно содржи заглавје во кое се дадени различни типови ја мета-податоци и содржи тело во кое се сместуваат податоци и доколку треба да се транспортираат објекти се врши нивна серијализација. [115, 114] Механизмот за размена на пораки се документира со WSD (Web Service Description) напишан во WSDL и го дефинира форматот на пораката, типовите на податоци, протоколите за транспорт и форматот за серијализација. Дополнително специфицира една или повеќе мрежни локации за повик на агентот кој овозможува трансфер. Постојат повеќе начини на кои може да се употреби еден сервис. Слика. 3.1 ги прикажува основните чекори: 1) побарувачот и овозможувачот се "запознаваат"; 2) се "согласуваат"за описот на сервисот и семантиката според која ќе се води интеракцијата; 3) агентите на побарувачот и

овозможувачот ја користат договорената семантика и WSD и 4) агентите на побарувачот и овозможувачот разменуваат пораки.



Слика 3.1 Генерален процес на користење на веб сервис [115]

Најчести користени типови на податочни формати се XML и JavaScript Object Notation (JSON).

Најчести користени типови на веб сервиси се:

- SOAP,
- XML-RPC и
- JSON-RPC и
- REST.

SOAP претставува стандард кој овозможува рамка за пакување и размена на XML пораки и притоа за пренос можат да се користат повеќе протоколи како HTTP, SMTP, FTP, RMI/IIOP или други [115]. XML-RPC е постар протокол кој употребува специфичен XML формат за размена на податоци. JSON-RPC е сличен на XML-RPC, но податоците ги дефинира во JSON формат. За разлика од генералниот архитектурен стил на веб сервиси, REST преставуваа поорганичувачки стил во кој агентите овозможуваат униформна семантика за интерфејси и дополнително интеракциите "немаат состојба т.е. значењето на пораката не зависи од состојбата на интеракцијата [25].

Веб сокети:

WebSocket преставува протокол за комуникација преку TCP поврзување. Овој протокол овозможува двонасочна комуникација помеѓу клиентот кој извршува недоверлив код во контролирана околина на оддалечени компјутери кои се избрани во комуникациите за дадениот код. Безбедносниот модел кој се користи за ова е стандардниот безбедносен модел кој најчесто се користи од страна на веб прелистувачите. Протоколот се состои од отворањето ракување проследено со додавање заглавје на основната поракапреку TCP слојот [49]. WebSocket API дефинира API за користење на WebSocket протоколот[116].

Библиотеки:

Библиотеките се чест случај на API. Содржат дефиниција и имплементација на функции дадени од производителот на софтверот и корисникот треба да ги референцира или импортира.

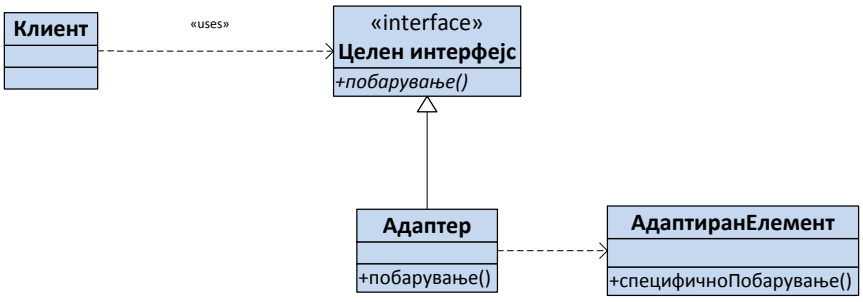
Повик на оддалечен објект (RPC):

Овој тип на програмирачките интерфејси на апликации најчесто користи протокол за повик на оддалечени објекти и се имплементира со локални прокси објекти кои ги претставуваат оддалечените објекти. Пример за ваков тип на протоколи се CORBA и .NET Remoting.

3.3 Моделирање на интероперабилност со адаптери

3.3.1 Адаптер шаблон

Еден стандарден пристап за моделирање на адаптер е користење на адаптер шаблонот [32]. Овој шаблон овозможува адаптирање, т.е. користење на функционалностите на еден елемент и нивно адаптирање во соодветна форма. Структурата на овој шаблон е прикажана на Слика. 3.2.



Слика 3.2 Структура Адаптер шаблонот (адаптирано од [32])

Друг добар пристап за моделирање на интероперабилност со помош на адаптери преставува дизајнирачкиот шаблон: Скалабилен адаптер [39]. Иницијално овој шаблон е дефиниран за да овозможи интероперабилност меѓу различни алатки за образование, но истиот може да се употребува и за креирање на адаптер за Софтвер како сервис во случај кога е потребна синхронизација на дел од податоците низ дел од дадените околини.

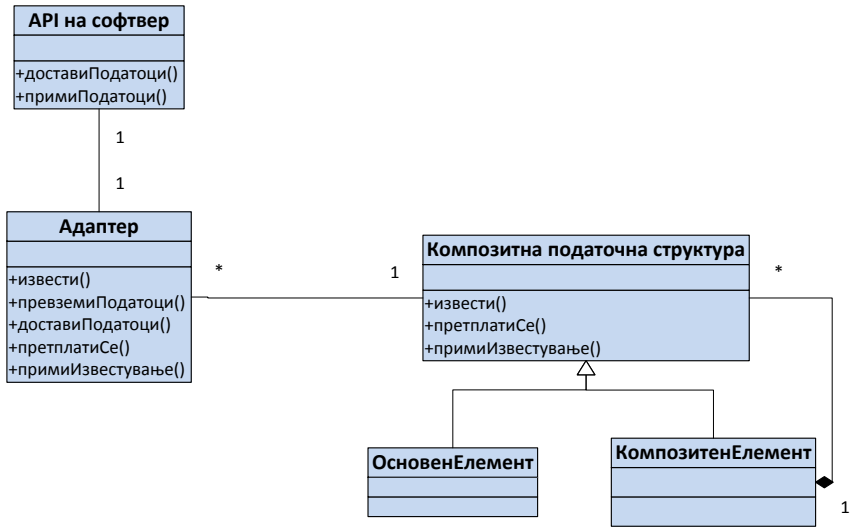
3.3.2 Модифициран скалабилен адаптер шаблон

Шаблонот Скалабилен адаптер преставува софтверска архитектура која се користи за да овозможи интероперабилност меѓу едукациски алатки со генерирање на мали адаптери за секоја различна околина за учење. Преставува спој од шаблоните адаптер, композитен шаблон и Publisher-Subscriber шаблонот. Овој шаблон го решава проблемот кога различни околини треба да соработуваат и потенцијално секоја околина е заинтересирана за само дел од податоците на друга околина. Бидејќи не може однапред да се одреди кои податоци ќе и бидат потребни на една апликација, потребно е да се дизајнира флексибилно и скалабилно решение. Со ова решение секој од адаптерите може да пристапи до дел од податоците на својата околина за учење и може да разменува податоци со останатите адаптери. Структурата на овој шаблон вклучува повеќе елементи:

- Постоечки околини - која може да доставуваат и примаат податоци преку своето API,
- Адаптери - кои овозможуваат интерфејс (преку APIто) на дадена околина за учење и воедно содржат методи за известување на промени кон своите претплатници и преземање на податоци од постоечка околина,
- Композитна податочна структура - која обезбедува пристап до произволни податочни елементи со користење на дрво структура за елементите
- Основен елемент - податочен елемент кој е лист во структурата на податочни елементи и
- Композитен елемент - кој е јазол во структурата на податочни елементи и може има (т.е. содржи) други податочни елементи (било основни или композитни)

Овој шаблон има ограничување за комуникација во насока од адаптерот кон композитната податочна структура. Во случајот на адаптери за Софтвер како сервис оваа комуникација не е применлива, истиот се дополнува со комуникација во двете насоки. Приказ на модифицираната структура на овој шаблон за користење на различни софтверски сервиси е даден на Слика. 3.3.

Адаптер шаблонот се користи за креирање на едноставни посредници меѓу услуги од ист тип, односно кога треба да се овозможи соработка



Слика 3.3 Структура на Скалиран адаптер шаблонот

и размена на податоци меѓу сервиси кои имаат многу слични функционалности и доста слични податоци. Изменетата верзија на Скалабилен адаптер шаблонот се користи кога треба да се овозможи посредување и комуникација на сервиси кои што имаат делумно преклопување на функционалностите и типовите на податоци.

3.4 Примена на модел на адаптер кај познати типови на Софтвер како сервис

Во овој дел се прикажува примената на адаптерскиот пристап за моделирање интероперабилност за три познати сервиси: сервиси за календари, сервиси за складирање и универзитетски сервиси.

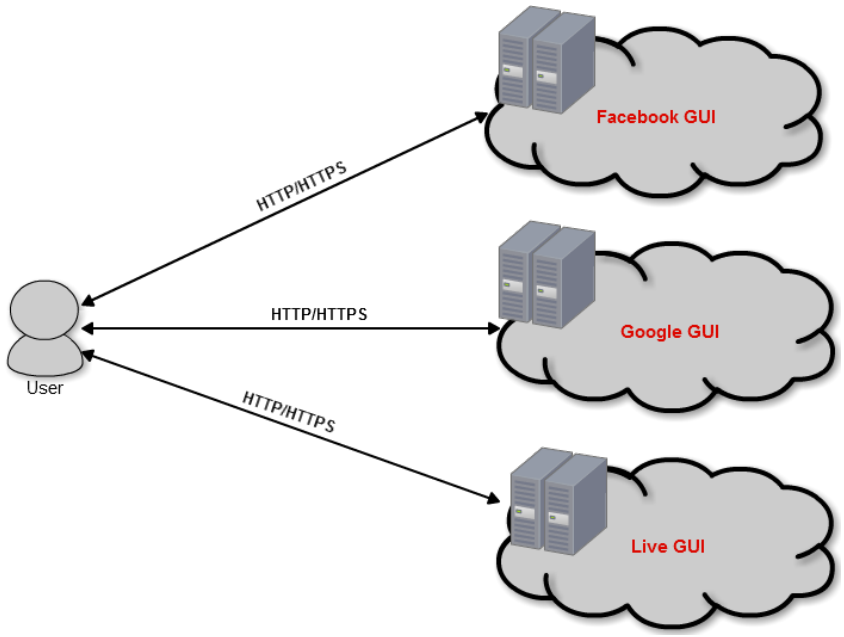
3.4.1 Сервиси за календари

Денес речиси секој човек, кој користи електронски услуги, исто така, користи календари понудени од различни провајдери. Нивната главната цел е да се закаже или сподели информација за состаноци, активности или настани. Многу често за луѓето користење на ваков вид на сервиси

е удобно и често сме зависни од користењето на овој електронски сервис како потсетник и организациска алатка.

Многу услуги за календари се нудат на пазарот и типичен Интернет корисник почнува да користи (или е принуден да ги користи) нови календари за различни намени (нова работа позиција, нова група, итн.) Проблемот се јавува кога во одреден момент мора да се проверат неколку услуги за календари, со цел да се синхронизираат сите настани или активности (дури и оние кои не се напишани во календари) и да се избегне преклопување или пропуштање на настан.

Типична употреба на повеќе календари е прикажана на Слика. 3.4 каде корисникот се поврзува на повеќе познати сервис (со употреба на нивниот соодветен интерфејс) со цел да го провери распоредот. Овие активности може да се исцрпувачки.



Слика 3.4 Поврзување на корисник на повеќе календари преку нивниот интерфејс

3.4.1.1 Протоколи и програмирачки интерфејси на апликации за календари

Потребата за стандарди за складирање на настани од календар и нивно споделување произлезе во 1996 година и се должи на голем број производи за електронски календари и нивното ограничување за размена на

информации само меѓу корисници во рамките на истиот систем. Работната група на Internet Engineering Task Force (IETF) истакна три клучни области за идна стандардизација: формат за размена , протокол интероперабилност и контрола на пристап [14]. Оваа група генерираше неколку стандарди:

- Internet Calendaring and Scheduling Core Object Specification (iCalendar),
- iCalendar Transport-Independent Interoperability Protocol (iTIP) и
- iCalendar Message-Based Interoperability Protocol (iMIP).

Денес iCalendar е широко употребуван стандард и се употребува од голем број на познати продукти за календари како Google Calendar [33], Apple iCal [13], Microsoft Outlook [82], IBM Lotus Notes [42] итн. Со користење на овој протокол датотеките за календари се споделуваат и разменуваат со користење на WebDav сервер кој нуди RESTful интерфејс.

Internet Calendaring and Scheduling Core Object Specification (iCalendar) стандардот дефинира "*формат на податоци за преставување и размена на информации за календари и распореди како што се настани, листа на обврски, записи во дневник, и информации зафатеност на термин, независно од која било поединечна календарска услуга или протокол*"[29].

Дефинира MIME тип на содржина и овозможува размена со користење на неколку стандардни протоколи за транспорт (како HTTP и SMTP), различни податочни системи и повеќе типови на транспорт и комуникација. Овозможува мапирање на типот на содржината за извршување на операции со календар [29].

Web Distributed Authoring and Versioning (WEBDAV) протоколот специфицира "*збир на методи, заглавија и HTTP/1.1 типови на содржина за управување со својства на ресурси, креирање и управување со колекции на ресурси, управување со namespace и заклучување на ресурси*"[27].

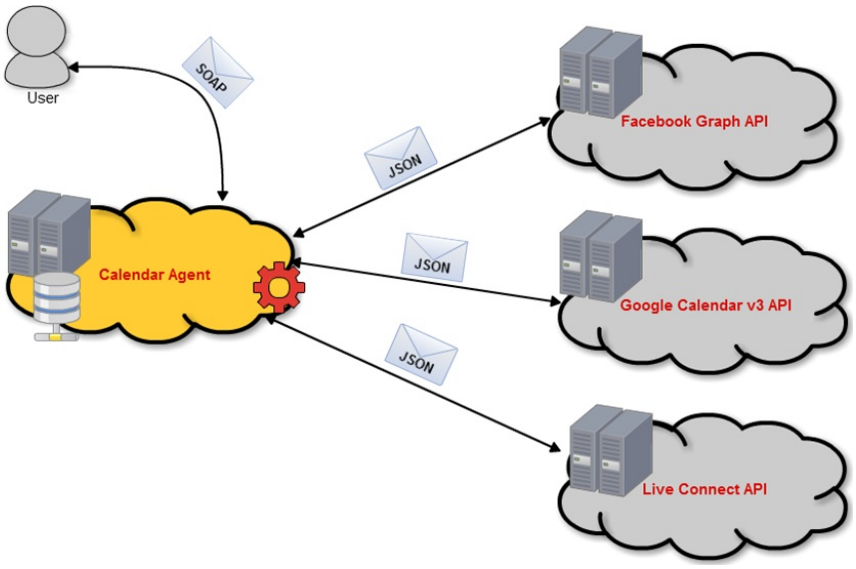
Calendaring Extensions to WebDAV (CalDAV) дефинира "*проширувања на (WebDAV) протоколот за спецификација на стандардизиран начин за пристап, управување и споделување на информации за календари и распореди базирани на iCalendar формат*"[28].

CalDAV протоколот овозможува три главни својства: одржување, пребарување и безбедност на календари [22].

3.4.1.2 Преглед на адаптерски агент за календарски сервиси

Адаптерскиот агент работи со три платформи на познати провајдери: Facebook, Google и Microsoft. За нашето истражување имаме развиено прототип користење Јава ЕЕ. Податоци за веб сервисите се чуваат во MySQL база на податоци. Календарот агент е конструиран со RPC стил веб сервис со користење на SOAP пораките за комуникација. Податоци во пораките се структурирани како JSON стринг поради чиста структура која ја нуди, а со цел да се направи веб сервис лесен за употреба и да за

интеграција. Целосен преглед на размената на податоци е прикажан на Слика. 3.5.



Слика 3.5 Преглед на размената на податоци на календар адаптерскиот агент

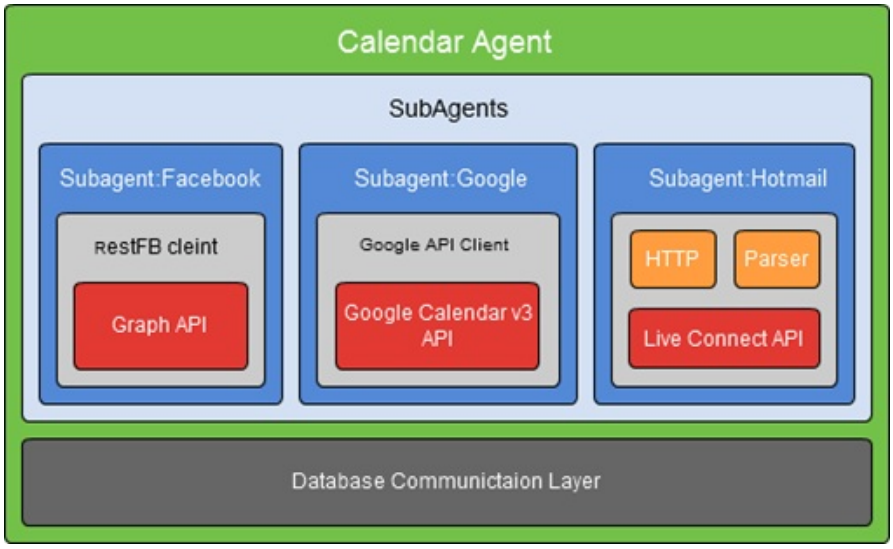
Агентот се имплементира со употреба на фасада дизајн шаблонот и интегрира и завиткува три други приватни агенти кои не можат се пристапат од надвор. Секој внатрешен агент е доделен да извршува задачи на дадена платформа и за ги чува резултатите во базата на податоци. Клиентите може да го пристапат само интерфејсниот агент со достава на автентикациски детали во една JSON порака. Пораката мора да биде SLL криптирана, а содржината зависи од задачата што треба да се изврши. Ако задачата успешно се изврши резултатот се сместува во базата на податоци и се праќа назад до корисникот преку JSON стринг.

Слика. 3.6 ја прикажува архитектурата на решението. Инстанци од приватните агенти се генерираат со користење на приватни фабрички методи. Секој внатрешен агент ја имплементира целосната логика за извршување задача за дадената платформа. Со тоа се овозможува скалабилност на решението.

Не се ограничува бројот на кориснички сметки по клиент. Регистрацијата на секоја сметка побарува валиден токен за пристап кој се зачувува во базата на податоци на соодветниот агент.

Следните функции се имплементираат со ова решение (подетално опишани во [74]):

- Пренос на настани,
- Креирање на настани,



Слика 3.6 Артитектура на календар адаптерскиот агент

- Манување на настани и
- Бришење на настани.

3.4.2 Сервиси за складирање

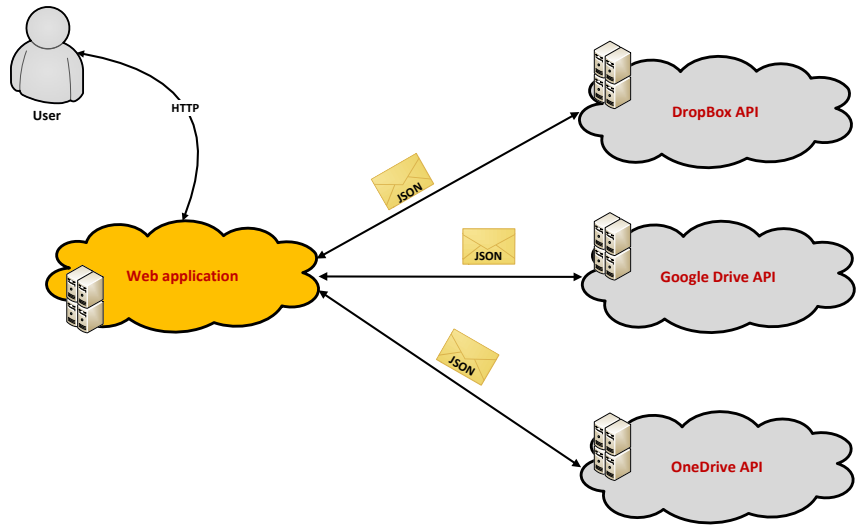
Денес голем број на луѓе користат сервиси за складирање и споделување на своите датотеки. Најчесто овие сервиси имаат ограничување во големината на складирањето, па често се доведуваме во ситуација целосно да го искористиме капацитетот на еден склад и да ни се потребно мигрирање на дел од документите во друг склад. Понекогаш ги дистрибуираме документи низ различни складови и имаме потреба од нивно префрлање од еден склад во друг. За таа цел, во рамки на ова истражување, се покажува имплементација која преставува адаптер за размена на информации и датотеки меѓу Dropbox [19], Google Drive [34] и OneDrive (SkyDrive) [83]. Идејата е да се овозможи заеднички интерфејс преку кој ќе може да се манипулира датотеките на било која корисничка сметка од дадените услуги.

3.4.2.1 Преглед на програмирачки интерфејси на апликации за складирање

Овој проект разменува информации со програмирачки интерфејси на апликации (APIa) за складирање на Dropbox, Google Drive и SkyDrive (OneDrive). Наведените сервиси користат RESTful програмирачки интерфејси. Со овие интерфејси може да се користат стандардните HTTP методи, како PUT, GET и POST, а кај дел од нив и HEAD и DELETE методите. Google Drive нуди XML и JSON API [35]. Dropbox и OneDrive нудат JSON API [20, 84]. Сите услуги нудат OAuth 2.0 автентикација.

3.4.2.2 Преглед на адаптерска веб апликација за сервиси за складирање

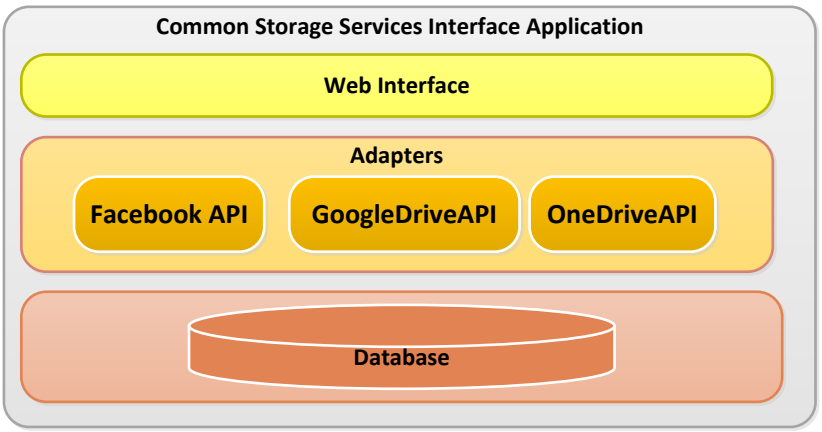
Адаптерската веб апликација работи со три платформи на познати провајдери: Dropbox, Google и Microsoft. Податоци за веб сервисите се чуваат во MS SQL база на податоци. Изработена е со употребена на .NET технологијата. Веб апликацијата комуницира со REST интерфејсите на сервисите со размена на JSON пораки. Целосен преглед на размената на податоци е прикажан на Слика 3.7.



Слика 3.7 Преглед на размената на податоци на апликацијата со сервисите за складирање

Агентот во својата имплементација содржи три други адаптери кои се доделуваат да извршуваат задачи на дадена платформа. Клиентите може да го пристапат само до интерфејсот на апликацијата и доставуваат автентикациски детали за дадена корисничка сметка на сервиси и истите се доставуваат како JSON порака до соодветниот сервис. Ако задачата успешно се изврши резултатот се добива соодветен токен со кој понатаму може да се листаат, менуваат, бришат или додаваат нови датотеки за дадената корисничка сметка.

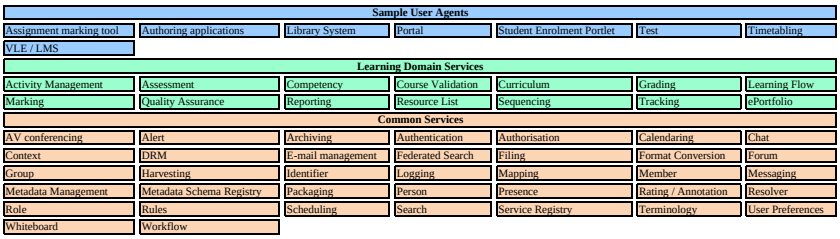
Слика. 3.8 ја прикажува архитектурата на решението. Инстанци од приватните адаптери се генерираат со користење на приватни фабрички методи. Секој внатрешен адаптер ја имплементира целосната логика за извршување задача за дадената платформа. Со тоа се овозможува скалабилност на решението. Дополнително не се ограничува бројот на кориснички сметки по клиент.



Слика 3.8 Архитектура на апликацијата за интегриран интерфејс на сервисите за складирање

3.4.3 Универзитетски системи

Во овој дел се врши анализа на можностите за интероперабилност меѓу основните апликации кои нудат различни видови на универзитетски сервиси. Стандардно на еден универзитет се користат повеќе софтверски продукти кои не ретко вклучуваат систем за управување со универзитетот, систем за управување со учење и систем за електронско тестирање.



Слика 3.9 The E-Learning Framework[111]

Овие системи се често развиени од различни производители и се имплементираат посебно и не се внимава на интероперабилноста [2]. Дополнителен предизвик се честите надградби или промени на овие системи.

Ние вршиме анализа на можните интеракции и соработка меѓу наведените системи, како и на постоечките стандарди кои може да се применат за да се овозможи автоматизирана интеракција. Главната цел е справување со прашањата на интероперабилност и да се анализира, ако тоа е можно, да се оди и чекор понатаму: да се воспостави интероперабилна околина, па еден универзитет да може да го промени видот на услугата, или провајдерот на софтвер, или и двете, без било каква загуба или штета.

Како пример на системи за анализа во оваа студија се земени системите кои се употребуваат на нашиот универзитет и факултет: iKnow системот за управување со универзитет [46], Moodle системот за управување со учење [85] и E-test системот за електронско тестирање [37]).

3.4.3.1 Преглед на типови на универзитетски сервиси и стандарди

Системите за управување со учење овозможуваат средина каде што предавачите комуницирате со учениците и студентите со испораката на материјали за учење, оценување на домашни задачи и други релевантни активности. Овој тип на систем ги следи на сите активности поврзани со процесот на учење. Дополнително овозможува на дискусии, форуми, итн... Функционалности на овие видови на системи се разликуваат и може да вклучуваат многу елементи. Еден сеопфатен пример на функционалности кои овие системи може да ги имаат е рамката за е-учење, прикажана на Слика. 3.9. Оваа рамка е развиена преку координирани напори на Британскиот комитет за здружени информациски сервиси (Joint Information Services Committee - JISC) и Министерството за образование, наука и обука на Австралија [111]. Оваа рамка дефинира 59 различни функционалности поделени во три главни групи: Пример на кориснички агенти, сервиси од домен на учење и заеднички служби.

Најпопуларен стандард за системите за управување со учење е Sharable Content Object Reference Model (SCORM). Претставува збир на технички стандарди и спецификации создавајќи елементи од онлајн материјали за учење и обука кои може да се споделуваат од различни системи [108]. Таа е развиена од групата за Напредно дистрибутер учење на Министерството за одбрана на САД. SCORM генерално референци од постојните стандарди и елементи на IEEE, AICC и Information Management Standard (IMS) спецификациите.

Системите за електронско тестирање и оценување се одделени од системите за учење и нудат можност за дефиниција и администрација на прашањето, создавање на тестови, испорака на тестирање, извршување на тест, евалуација, оценување и управување со корисниците и резултатите [50].

Добро дефиниран стандард за овој тип на системи е IMS QTI спецификацијата, која овозможува размена на податоци за содржина, тестови и резултати [48]. Создаден е од страна на IMS Global Consortium. Овој стандард опишува податочен модел за презентација на податоци за прашања и тестирања и нивните соодветни резултати. Спецификација содржи голем број на документи: водич за имплементација, тест за оценување, информации модел за елементи и секции, метаподатоци и податоци за користење, извештаи за резултати, водич за интеграција, поврзување со XML и водич за миграција.

Во нашата анализа за системите за управување со универзитет вклучуваме подмножество на функционалности за ваков тип на системи: студентски служби, академска евиденција која ги следи на сите релевантни информации за студентите (вклучувајќи ги и нивните досиеја и датотеки), како и финансиски и други административни работни процеси кои се вклучени меѓу студентите и Универзитетот. Системот кој го користиме во нашата анализа е iKnow, кој го користи Универзитетот Св. Кирил и Методи.

iKnow овозможува размена на електронски информации меѓу повеќе учесници: студенти, професори, администрација, универзитетската управа и Министерството за образование и наука. Се состои и две централни компоненти: сервиси за упис и централни студентски сервиси. Сервисите за упис нудат волшебник кој ги води кандидатите низ процесот на упис, форми за рачен внес на податоци за кандидатите, обработка на податоците на кандидатите, модул за рангирање и објавување на резултатите за упис. Централните студентски сервиси вклучуваат модул за администрација на универзитетот и факултетите, модул за студентски програми и распореди, модул за студентски активности, модул за персонална идентификација и контрола на пристап, модул за електронска наплата и модул за миграција на податоци од претходни системи.

3.4.3.2 Интеракции меѓу универзитетски сервиси

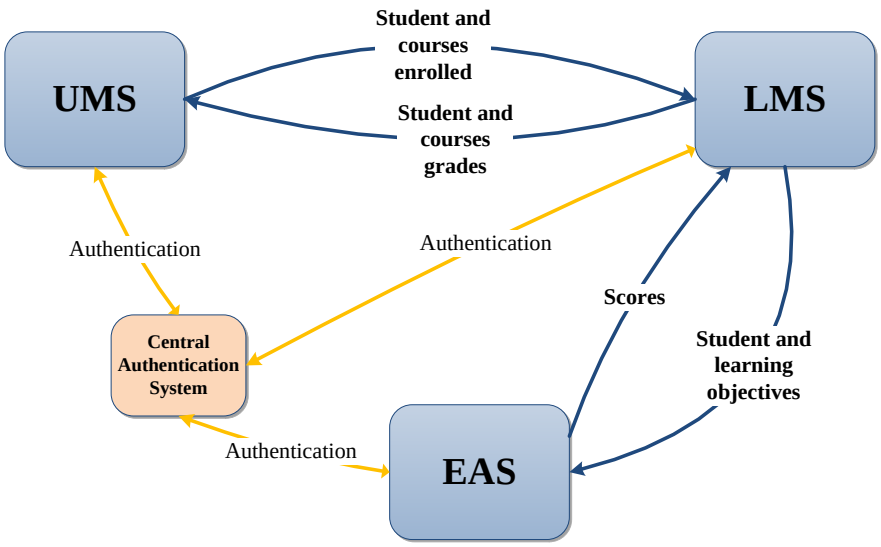
Потребата за размена на информации варира, во зависност од соодветните имплементации на сервисите и нивните функционалности. Слика. 3.10 прикажува едно можно сценарио за размена на податоци и информации меѓу системите.

Интеракцијата од системите за учење кон системите за електронско тестирање се состои од пренос на информации за наставна програма, цели за учење и студенти. Во спротивната насока системите за електронско тестирање доставуваат резултати за оцени од спроведените тестирања.

Интеракцијата од системите за управување со универзитетот кон системите за учење се состои од пренос на информации за студентите и предметите кои ги имаат запишано. Во спротивната насока системите за учење доставуваат записи за резултати и оцени на студенти и по потреба соодветни извештаи.

Во ова сценарио не постои директна интеракција меѓу системите за управување со универзитетот и системите за електронско тестирање.

Дополнително е прикажан централен автентикациски систем за трите вида на сервиси.



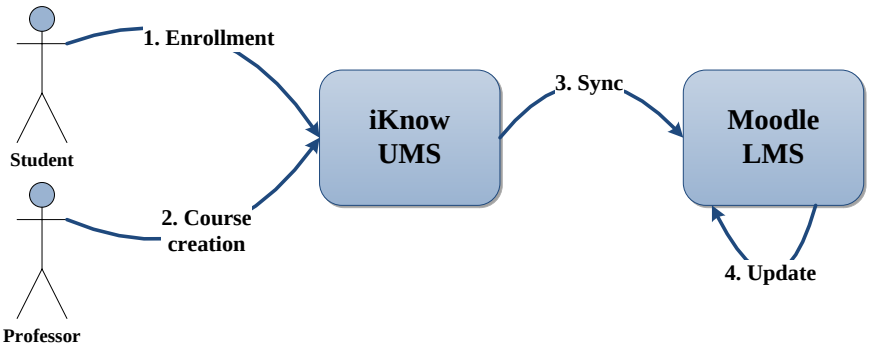
Слика 3.10 Interaction among LMS, EAS and UMS systems

3.4.3.3 Сценарија за интеракција и дискусија

Трансфер за информации за запишани студенти и предмети:
За оваа цел на нашиот системите за управување со универзитетот беше развиен специјален модул за да се овозможи овој трансфер. Целта е да овозможи трансфер на студентите кои запишале семестар и предмети во Moodle системот, како и да се креираат соодветни курсеви за предметите и да се внесат нивните одговорни наставници. Се користи можноста на Moodle за користење на екстерна база на податоци. Овој модул употребува синхронизација за внес/измена на податоци во регуларни временски периоди.

Ова сценарио е прикажано на Слика. 3.11 и се состои од следните чекори :

1. На почетокот на секој семестар студентите во iKnow го запишуваат семестарот избираат предмети кои ќе ги слушаат тој семестар;
2. Во текот на упис на семестарот секој професор има опција (вградена во iKnow) автоматски да креира Moodle курс за предмет на кој е доделен или да креира еден Moodle за повеќе слични или исти предмети во iKnow (посебно дефинирани од различни причини);
3. Синхронизацијата кон Moodle се извршува во регуларни временски интервали; и
4. Moodle курсот автоматски се креира и сите студенти кои го имаат запишано курсот, како и одговорните наставници, се додаваат на курсот со соодветните пермисии.



Слика 3.11 Сценарио: Трансфер за информации за запишани студенти и предмети

Оваа интеракција не е стандардизирана, па при промена на било кој од системите ќе бидат потребни адаптации. Подобрување во оваа комуникација би било користењето на IMS Enterprise стандардот [47]. IMS Enterprise Information Model е дизајниран да поддржи интероперабил-

ност на следните елементи од бизнис процесите: Одржување на податоци за персонален профил, управување со групи, управување со упис и процесирање на финални резултати. потенцијален проблем е замена на некој од системите со нов систем кој не го поддржува IMS Enterprise стандардот.

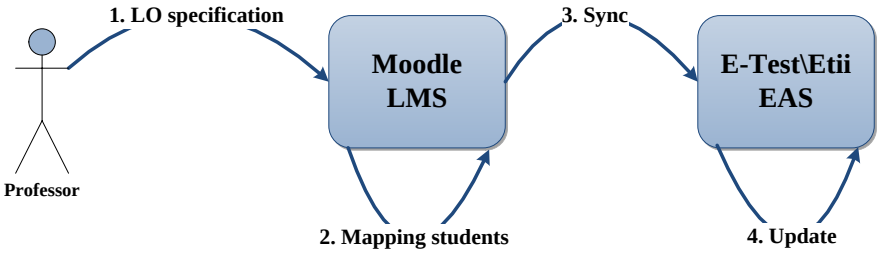
Друго дополнително подобрување на овој процес би било креирање на SCORM пакети за учење за секој курс и нивно импортирање во Moodle при креирањето на курсот.

Трансфер за информации од системот за учење кон системот за електронско тестирање:

Во процесот на креирање на тест потребен е да се специфицираат целите за учење кои треба да бидат вклучени. Идејата на ова сценарио е да се достави потребна спецификација за тест од системот за учење кога ќе биде потребан активност за оценување. Креираните тестови треба да бидат достапни само на оние студенти кои го следат курсот. Евалуацијата на резултатите и оцената може да биде специјално дизајнирана т.ш. може да вклучи само студенти кои задоволуваат одредени предуслови за да се тестираат (како на пр. минимален број на поени од тест X, доставена домашна бр. Y итн...).

Ова сценарио е прикажано на Слика. 3.12 и се состои од следните чекори :

1. На почетокот на семестарот професорот дефинира цели на учење за секој курс во Moodle;
2. Moodle содржи евиденција од сите студенти и нивните курсеви;
3. Синхронизацискатa активност се извршува на секој час од страна на Moodle и се доставуваат информации за целите на учење и запишаните студенти; и
4. Системот за електронско тестирање ги ажурира целите на учење и листата на запишани студенти за активираните курсеви.



Слика 3.12 Сценарио: Трансфер за информации од системот за учење кон системот за електронско тестирање

Ова сценарио на интеракција е генерално тешко за имплементирање бидејќи побарува автоматизирано креирање на прашања. Една можност е употреба на семантика и онтологија за секоја цел на учење и нивна употреба за креирање на прашања. Ова сугерира дека онтологијата е веќе креирана и достапна. Јованов и Гушев [51] вршат истражувања на оваа тема и веќе е креира почетен модел.

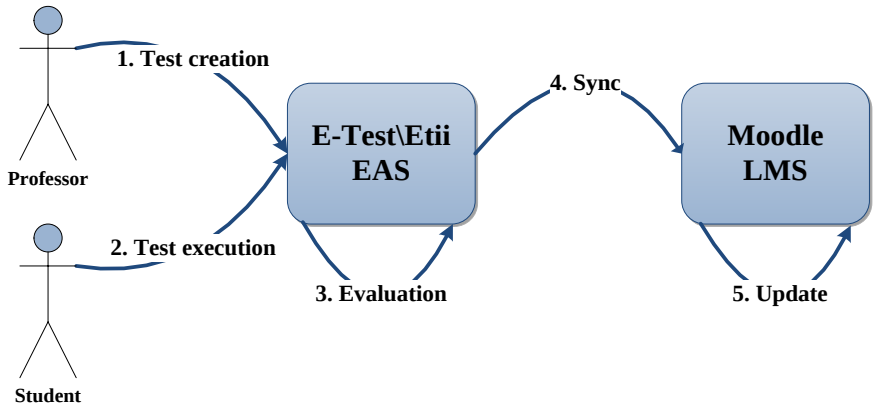
Ова сценарио не е имплементирано во нашите системи.

Трансфер резултати и оцени од системот за електронско тестирање кон системот за учење:

Целта на ова сценарио е да се овозможи автоматски пренос на студентските резултати од e-Test во Moodle за секоја активност на оценување (тест).

Ова сценарио е прикажано на Слика. 3.13 и се состои од следните чекори :

1. Инструкторот креира стратегија и закажува тест;
2. Студентот одговара на прашања од тестот и системот за електронско тестирање автоматски ги евалуира резултатите;
3. Се започнува синхронизирачка активност со која соодветните резултати од тестирањето од e-Test се доставуваат во Moodle;
4. Синхронизирачката активност се активира на секој час и Moodle ги ажурира добиените податоци;



Слика 3.13 Сценарио: Трансфер резултати и оцени од системот за електронско тестирање кон системот за учење

Иницијално ова сценарио беше реализирано со креирање на посебна активност во Moodle со која се превземаа резултатите од e-Test. Оваа активност побаруваше идентификатор на тестот и со ги превземаше резултатите. Компонентата требаше да биде изменета со новите верзии на

Moodle и побаруваше често одржување. Во моментов се користи алтернативно решение со користење на стандарди помошни документи за експортираните податоци (во csv, таб-одвоени податоци и слични формати) да се вчитаат во Moodle.

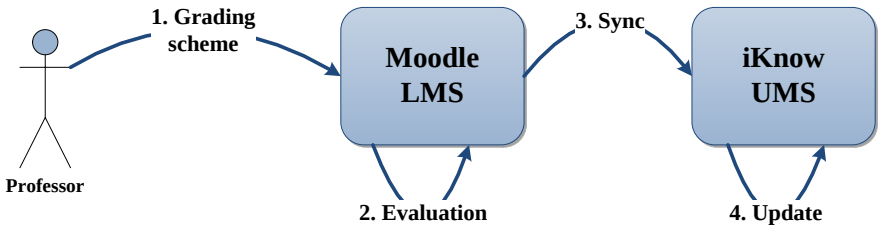
Оваа интеракција не е стандардизирана, иако употребува податоци за кои постојат стандардизирани претставувања. Подобар избор е употреба на QTI стандардот. Генерален проблем е ако еден од двата система не го поддржуваат QTI стандардот. Дополнително потребно е да се специфицира секој тест со колкав процент влегува во финалната оцена.

Трансфер на резултати за предмет од системот за учење кон системот за управување со универзитет:

Идејата на ова сценарио е да овозможи автоматски трансфер на оцени од системот за учење кон системот за управување со универзитет. За ова да биде возможно соодветниот професор треба да дефинира критериум за оценување за секој предмет (секоја активност со колкав процент влегува во финалната оцена) како и сите резултати од оценувања да бидат достапни во системот за учење. Дополнителен административен предизвик во нашата земја е побарувањето за пријава на испит во сесија и достава на потпишани пријави за секоја оцена на студент.

Ова сценарио е прикажано на Слика. 3.14 и се состои од следните чекори :

1. Професорот ги дефинира активностите за оценување по предмет, критериумот за оценување и условите за положување во Moodle;
2. Moodle содржи ажуриран запис со резултатите од сите активности за оценување за секој студент и пресметува оцена според дефинираниот критериум и услови;
3. Синронизирачка активност се стартува на Moodle LMS и ги доставува сите релевантни информации до iKnow UMS; и
4. Синхронизирачката активност на iKnow ги ажурира оцените на студентите за предметите.



Слика 3.14 Сценарио: Трансфер на резултати за предмет од системот за учење кон системот за управување со универзитет

Табела 3.1 Препорачани стандарди за системи за овозможување на делумна интероперабилност

Систем / Стандард	SCORM	QTI	IMS Enterprise
Систем за управување со универзитет	X		X
Систем за управување со учење	X	X	X
Систем за електронско тестирање	X	X	

Слично како и претходното сценарио оваа интеракција се реализира со употреба на алтернативен метод (со користење на помошни датотеки). Резултатите се експортираат во помошен excel документ и се импортираат во iKnow.

Оваа интеракција може да биде подобрена и стандардизирана со користење на IMS Enterprise стандардот.

3.4.3.4 Дискусија за интеоперабилност на универзитетски сервиси

Презентираните сценарија не ги вклучуваат сите можни интеракции, ако и сите можни размени на податоци. За некои од сценаријата не постојат применливи стандарди или стандардите не се доволни.

Формална спецификација на размена на информации меѓу системите за учење и тестирање вклучува најразлични параметри како информации за студент, информации за предмети, спецификација за цели за учење, резултати од тестови, извештаи за евалуација и оценување, итн... QTI стандардот специфицира интероперабилност за прашања и тестови, но е доволен за соочување со сите предизвици, како и SCORM, кој дефинира стандард за размена на материјали за учење.

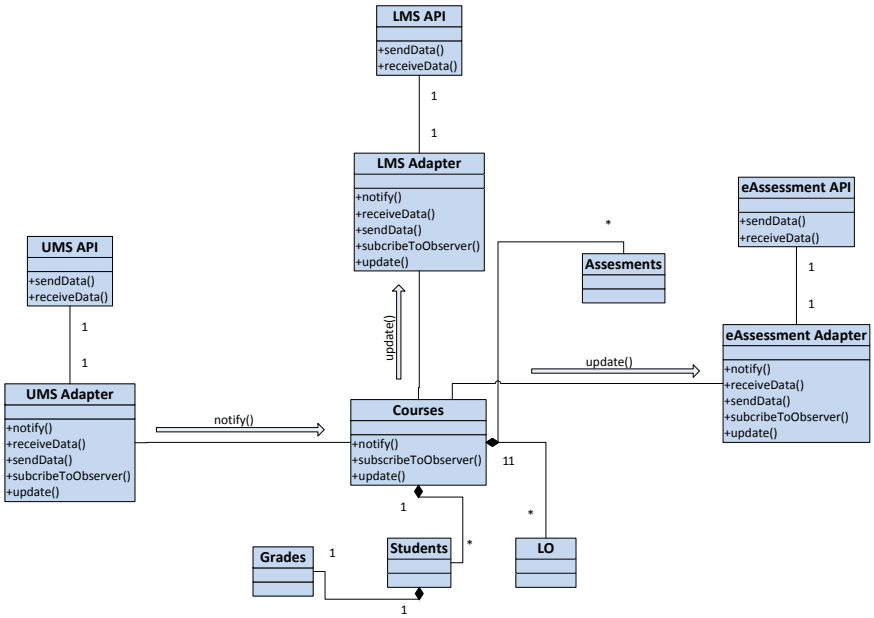
Размената на информации меѓу системите за управување со универзитет и за учење вклучува најмалку информации за упис, предмети, студенти и академски резултати. Возможна е примена на IMS Enterprise стандардот.

За да се овозможи делумна интероперабилност меѓу системите опишани во сценаријата секој од системите треба да поддржува одредени стандарди. Табела. 3.1 барем кои стандарди треба да бидат поддржани. Не ги опфаќа сите идентификувани предизвици, но ги покрива основните предизвици за размена на информации за учење.

3.4.3.5 Примена на проширен скалабилен адаптер за универзитетски сервиси

Размената на податоци меѓу универзитетските сервиси е пример на размена на податоци меѓу хетерогени системи кои меѓу себе разменуваат

само дел од податоците кои ги содржат, и размената на податоци не е иста меѓу сите системи, односно некои податоци се разменуваат само меѓу два системи, некои меѓу сите. Од таа причина проширениот скалабилен адаптер шаблон е добро решение за оваа интеракција. Во овој случај секој од податоците кои се разменуваат се моделира како компонента и интеракцијата меѓу адаптерите на сервисите се ограничува само за дадени компоненти. Пример на извршување на првото сценарио е дадено на Слика. 3.15. Во овој случај креираната листа на курсеви кои се означени за креирање се предава од адаптерот на универзитетскиот систем и притоа истата се пропагира до системите за управување со учење и тестирање.



Слика 3.15 Примена на проширен скалабилен адаптер за универзитетски сервиси

3.4.3.6 Заклучок за интероперабилност на универзитетски сервиси

Областа на е-учење нуди многу добро утврдени стандарди, додека областа на електронско оценување е сеуште во развој. Во овој дел е претставен нашиот пристап во дефинирањето интероперабилност предизвици меѓу системите за учење, електронско тестирање и универзитетско управување. Како пример се коментираат предизвиците со кои сме се соочиле и

некои решенија. Ние, исто така, даваме препораки за тоа кои стандарди може да се користат со цел да се обезбеди делумна интероперабилност. Сепак, сите интеракции меѓу овие системи е потребно прецизно да се дефинираат и многу малку број на достапни системи ги поддржуваат сите наведените стандарди или пак стандардите не се секогаш применливи.

3.5 Главен придонес

Генерално моделирање на интероперабилност на Софтвер како сервис не е возможно. Возможно е стандардизирање на програмирачки интерфејси на апликациите (API), но со тоа се овозможува само техничка интероперабилност, т.е. само размена на податоци, без вистинска соработка на сервисите. Моделирање на одреден домен на Софтвер како сервис е возможно и не ретко зависи од примената на стандарди во тој домен или пак од протоколот кој го наметнува доминантниот добавувач на сервис од тој домен. Возможно моделирање на интероперабилност со користење на Адаптер шаблонот за креирање на едноставни посредници меѓу услуги од ист тип, додека изменетата верзија на Скалабилен адаптер шаблонот може да се користи кога треба да се овозможи посредување и комуникација на сервиси кои што имаат делумно преклопување на функционалностите и типовите на податоци.

Глава 4
P-TOSCA модел за портабилност на апликации кај Платформа како сервис

Преглед Со цел да се реши проблемот на портабилност во облак OASIS работи на нов стандард - TOSCA, со кој може да се опише портабилна верзија на една апликација. Во оваа глава се дава преглед на недостатоците на TOSCA моделот и се воведува P-TOSCA моделот кој претставува проширување на TOSCA стандардот. Се дава и имплементација - нова Платформа како Сервис (PaaS), која овозможува софтверските решенија опишани во дадениот модел лесно да бидат применливи и преносливи во било кој систем на облак.

4.1 Вовед

Постоечките провајдери на услуги во облак сеуште не нудат автоматска пакување и преносливост на апликации. Оваа ситуација создава заклучување на апликациите во еден провајдер, од една страна, или тешкотии во миграцијата , особено за оние корисници кои имаат сложени и големи решенија.

Со созревањето на технологијата за пресметување во облак се појавува потребата за стандарди. Во моментов, во областа на портабилност во облак, два стандарди се во развој: Open Virtualization Format (OVF) од Distributed Management Task Force (DMTF) [15] и Тополошки и оркестрациски сервиси за апликации (Topology and Orchestration Services for Applications - TOSCA) од OASIS. OVF ја дефинира портабилноста на формати на виртуелизација и виртуелни машини. TOSCA ја дефинира портабилноста преку креирање на спецификација на услуги и апликации, која е пренослива, со која лесно се управува и е применлива во секое облак решение во било кој провајдер.

4.2 Преглед на релевантна литература

4.2.1 Пристапи за моделирање на портабилност на сервиси во облак

Најновите истражувања во поглед на преносливост на услуги во облак на секое ниво од слоевите, се справуваат со проблемот од три различни перспективи:

- Една перспектива е **употребата на стандардите** за да се постигне оваа задача. Во оваа насока истражувањата на Lewis [78] и Zhangи останати [118] се обидуваат да ги идентификуваат тековните предизвици, напорите и опции. Во овие истражувања авторите дискутираат за моменталните ситуации поврзани со стандардите и предлоуваат насоки за понатамошен развој, но не специфицираат конечно решение. Galan и останати [31] продложуваат проширување на отворениот стандард OVF, ни нивното решение е ограничено само на витруелните елементи.
- Втората перспектива рагледува **употреба на други комерцијални решенија** како апстракциско-ориентиран (abstraction-driven) пристап, каде што апстрактни јазици се користат за да се специфицираат решенија [99] или да се создаде слој за апстракција на складишта (Storage Abstraction Layer - CSAL) [40].
- Третата перспектива се состои од друг пристап кој се фокусира на искористување на **семантички технологии** да се создаде семантичка интероперабилност рамка [81], автоматски да се анализираат API-јата на понудувачот на услуга во облак [12] или да се користи mOSAIC за оваа намера [96, 86, 97].

OASIS воведува нов стандард за пресметување во облак наречен Спецификација за топологија и оркестрација апликации во облак (Topology and Orchestration Specification for Cloud Applications - TOSCA). TOSCA техничкиот комитет ја објави првата верзија на стандардот за *"интероперабилен опис на апликации и инфраструктура на сервиси во облак, релации (врски) помеѓу делови на услугата, и оперативното однесување на овие услуги (на пример, примена, исклучување и сл.)"*[91]. Ова иницираше ширење на истражувачките активност во различни насоки, како IBM напорите за спроведување на овој стандард [9, 21], создавање на средина за извршување [7] која во моментот ги нуди императивна обработка на CSAR и општ модел [72], алатки за моделирање [57, 8] и обид да се создаде овој тип на спецификација [79].

4.2.2 TOSCA

Овој дел претставува краток преглед на јазикот, структурата и употребата на TOSCA.

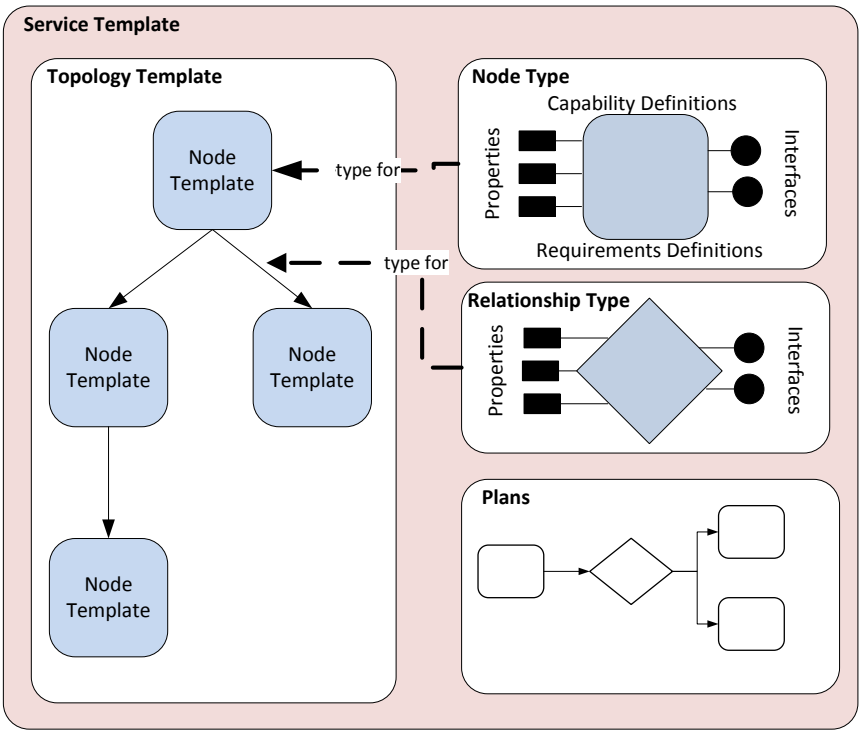
Тополошки и оркестрациски сервиси за апликации (Topology and Orchestration Services for Applications - TOSCA) е спецификација развиен од страна на OASIS, не-профитабилна компанија, и од ноември 2013 претставува OASIS стандард. Идејата зад TOSCA е да им овозможи на преносliв и интероперабилен опис на апликација со користење на јазикот предвидени со оваа спецификација, како и управување во текот на целиот животен циклус на апликацијата [91].

Оваа спецификација се базира на отворени стандарди како XML Schema 1,0 за да се опише услуга/апликација и BPEL 2.0 и BPMN 2,0 за управување со услуга независно од провајдерот и системот на облак. Во верзија 1.0, сегашниот стандард, услуги и нивното управување се дефинирани со користење на документот за сервисен шаблон (Service Template). TOSCA јазикот е проширлив и им дозволува користење на атрибути и елементи од други XML namespaces.

Верзијата 1.0 на TOSCA OASIS стандардот специфицира јазик кој поддржува дефиниција на мета-модел на услуги и управување со животниот циклус. Централниот дел на TOSCA спецификација е сервисниот шаблон, прикажан на Слика . 4.1, и се состои од две логички делови: шаблон за топологија (Topology Template) и планови (Plans). Шаблонот за топологија ја опишува топологија структурата на услугата/апликацијата преку опис на тополошки елементи и односите помеѓу нив. Плановите го дефинираат управување со животниот циклус на услугата/апликацијата опишани од страна на шаблонот за топологија.

Шаблонот за топологија се состои од шаблони за јазли (Node Template) (кои ги претставуваат елементите на апликацијата) и шаблони за релации (Relationship Template) (кои ги опишуваат врските меѓу елементи). Секој шаблон за јазли содржи референца до (т.е. е од тип на) тип на јазол (Node Type), која дава опис на секој тип на јазол за неговите способности, побарања, својства и интерфејси (операции). Еден тип на јазол може да биде референциран од повеќе шаблони за јазли. Шаблонот за релации ги дефинира односите помеѓу шаблоните за јазли, со наведување на типот на врска, извор и цел на врската и може да има дополнителни параметри. Извор на врската е побарување дефинирано за еден шаблон за јазол, и целта е способноста дефинирана за друг шаблон за јазол. Како и шаблоните за јазли, така и шаблоните за релации содржат референца (т.е. се од тип на) тип на релација (Relationship Type). TOSCA поддржува и композиција на сервисни шаблони.

Намената на планови е да се извршуваат соодветни акции за дадени јазли (или типови на јазли) и врските меѓу нив (или врските меѓу типовите на јазли) и нивните интерфејси (операции) и се дефинирани со користење BPMN и BPEL јазици.



Слика 4.1 TOSCA сервисен шаблон (адаптирано од [91])

За автоматизирана примена на апликација TOSCA вклучува дефиниција на артефакти од два вида: за имплементација и за примена. Дефинициите и артефактите за дадена апликација се пакуваат во zip архива наречена Архива на сервис во облак (Cloud Service Archive - CSAR). Секоја CSAR архива мора да содржи поддиректориум наречен TOSCA-Metadata (која мора да содржи TOSCA.meta датотека) и треба да содржи директориум Definitions (која содржи документ или документи за дефиниција на апликација).

4.3 Недостатоци на TOSCA

Бидејќи TOSCA е сеуште во почетна фаза на развој, сегашната верзија на овој стандард нема прецизна дефиниција (или воопшто не поддржува дефиниција) на некои елементи потребни за автоматски да се извршат некои животни процеси на апликацијата.

Првиот елемент кој бара понатамошна попрецизна дефиниција е Properties елементот на Node Template елементот. Овој елемент е наменет за да се опишат некои својства на јазлите, во зависност од нивниот тип. Овој елемент е поставена како комплексен тип на тековната верзија на XML schema-та на TOSCA и не е дефиниран, иако дава информации за некои својства од суштинско значење, како на пр. хардверска дефиниција на ниво/елемент (како број на процесори, диск големина и големината на меморијата) или порта која ја користи веб сервер. Во почетниот TOSCA пример [90] во овој елемент се прецизно вклучени својства за ниво/апликација, но генералната конструкција на овој елемент овозможува дефиниција на сопствени својства и ова, во иднина, може да биде тешко да се толкува.

Друг недостаток на оваа спецификација е дефиницијата на безбедносни политики и права на пристап. Првично беше планирано тие да се поставуваат со помош на планови, но во најновата верзија на документот е јасно наведено дека *TOSCA не наметнува употреба на било кој специфичен механизам или технологија за автентикација на клиенти*[91].

За било која апликација управувањето со животниот циклус е од суштинско значење. TOSCA се потпира на BPMN и BPEL јазиците за опис на процесите од животниот циклус. Бидејќи овие јазици се јазици со општа намена, тие не нудат поддршка за создавање на плановите за управување. Обид во оваа насока е во тек [56], но без комерцијална околина за поддршка на овој напор, резултатите се уште не се видливи.

Потенцијален проблем при примената на TOSCA спецификација се случува кога еден слој/апликација не е јавен (не е наведен во DNS или на друго место и облак провајдерот доделува динамична IP адреса) и се наоѓа на посебен сервер, а други нивоа/апликации на други сервери зависат од ова приватно ниво/апликација и им е потребна нивната локација. TOSCA моделот не нуди механизам за управување и информација со локацијата на нивоа/апликации. Една можност да се надмине оваа ситуација е користење на влезни параметри во Plan објектот, но сепак овој механизам е двосмислен и бара дополнително појаснување.

4.4 Нови дефиниции и проширувања на TOSCA

Со цел да се овозможи целосно автоматизирано управување со животниот циклус на една апликација се проширува дефиницијата на TOSCA со нови елементи:

- се специфицира употреба на *надворешен namespace* за ServerProperties
- се специфицира употреба на *initial number of instances* дете-елемент на ServerProperties елементот
- се проширува ServerProperties елементот на Node Template со *Server-IPAddress* елементот

- се специфицира употреба на *надворешен namespace* за ScriptArtifactProperties
- се проширува Properties елементот на Artifact Template со *InputParameters* елемент
- се проширува Properties елементот ба Node Template со *ServerSecurityProperties* елемент
- воведување на *XML дефиниран план*

Следува детално објаснување.

Во својства на Node Template се вклучува назад оригиналниот ServerProperties елемент со стандардните елементи, како бројот на процесори, диск големина и големината на меморијата. Покрај тоа се вклучува и NumberInstances елемент за да се дефинира потребниот број на инстанци за даденият тип јазол. Важен проширување на својствата е елемент кој нарекува ServerIPAddress. Овој својство не се поставува од страна на TOSCA архитектот (или ако е поставено, тоа ќе биде игнорирано). Ова својство се поставува од страна на TOSCA платформа која ја применува спецификацијата. Код. 4.1 прикажува пример за дефиниција на проширените својства за NodeTemplate.

```
<NodeTemplate id="LinuxVm"
              name="LinuxVm for Tomcat"
              type="Server">
        <Properties>
          <ServerProperties>
              <NumCpus>1</NumCpus>
              <Memory>1024</Memory>
              <Disk>10</Disk>
              <InitialNumInstances>
                    2
              </InitialNumInstances>
              <ServerIPAddress>
                    194.149.135.5
                    <!-- This property
                          will be ignored -->
              <ServerIPAddress>
          </ServerProperties>
        </Properties>
...
</NodeTemplate>
```

Код 4.1 Пример за дефиниција на проширен ServerProperties елемент во NodeTemplate

Исто така се користи и да се проширува елементот Properties за ArtifactTemplate каде TOSCA архитектот да додадете влезен параметар на артефактот кој може да биде својство на јазол, во случај конфигурацијата на апликацијата да треба да користи некои својства јазол (како локација ниво/апликација, која ќе се добие од страна на ServerIPAddress својството на ниво/апликација). Кодот. 4.2 прикажува пример на дефиницијата на влезен параметар за артефакт.

```
<ArtifactTemplate id="appconfscript" type="ScriptArtifact">
    <Properties>
        <ScriptArtifactProperties>
                <ScriptLanguage> sh
                </ScriptLanguage>
                <PrimaryScript>
                     scripts/MyApp/configure.sh
                </PrimaryScript>
                <InputParameters>
                   <InputParameter
                     nodeTemplateId="LinuxMySql"
                     property="ServerIPAddress"/>
                </InputParameters>
        </ScriptArtifactProperties>
    </Properties>
    ...
</ArtifactTemplate>
```

Код 4.2 Пример за дефиниција на влезен параметар на артефакт

Додаден е дополнителен елемент во елемент за својства на Node Template: ServerSecurityProperties. Овој елемент ги дефинира општите својства за безбедност на ниво/апликација, преку дозволени протоколи и порти. Кодот. 4.3 покажува пример дефиниција на овој елемент.

```
<NodeTemplate id="TomcatWebServer"
        name="TomcatWeb Server"
        type="TomcatWebServer">
    <Properties>
        <ServerSecurityProperties>
            <ServerSecurityProperty>
                    <protocol>TCP</protocol>
                    <port>80</port>
            </ServerSecurityProperty>
        </ServerSecurityProperties>
    </Properties>
</NodeTemplate>
```

Код 4.3 Пример за употреба на ServerSecurityProperties во Node Template

Значително е променет елементот Plan од тековната верзија и е заменет со план дефиниран во XML, со цел да им овозможи на основена обработка на планови. Plan елемент се состои од Node Template-и и нивни операции (поставени во дефинициите) потребни за да се изврши дадена акција. Планот елемент, исто така, содржи услови кои се прошируваат да бидат други план елементи и со тоа се задаваат предуслови за даден план елемент. Операции се извршуваат секвенцијално според редоследот на описот, и ако за одреден план постојат предуслови сите потребни планови како предуслов се извршуваат рекурзивно. Кодот. 4.4 покажува пример дефиниција на овој елемент.

```
<Plan id="InstallApplication"
```

```
    <NodeTemplateOperations>
      <NodeTemplateOperation ref="TomcatWebServer">
          <Operation name="install"/>
          <Operation name="configure"/>
      </NodeTemplateOperation>
      <NodeTemplateOperation ref="SimpleApp">
          <Operation name="install"/>
          <Operation name="configure"/>
      </NodeTemplateOperation>
    </NodeTemplateOperations>
</Plan>
```

Код 4.4 Пример на Plan елемент со XML

4.5 Ново предложено решение

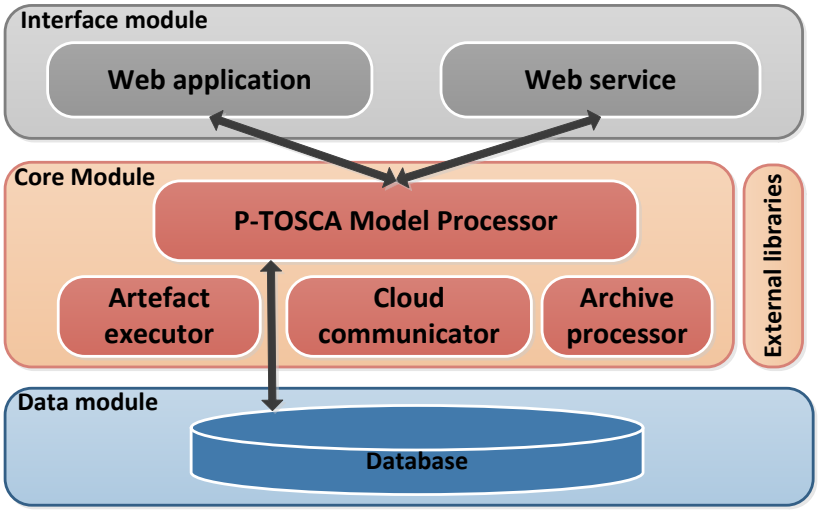
4.5.1 P-TOSCA базирана Платформа како Сервис (PaaS)

Општа идеја на Платформата како Сервис за P-TOSCA моделот е клиенти да имаат единствена точка во било која околина на облак каде што тие може да ги постават своите CSAR архиви за нивна обработка и платформата да ги обработи архивите, да ги креира дефинираните инстанци и да нуди извршување на дефинирани планови.

Платформата е поставена на Ubuntu, но лесно може да се префрли на било кој тип на Linux оперативен систем бидејќи не зависи од било Ubuntu специфични функционалности и тоа е лесно се конфигурира. Слика. 4.2 ја прикажува архитектурата на P-TOSCA платформата. Сите елементи на платформата се изградени во Java програмскиот јазик, но тие користат Linux специфични својства со цел да функционираат правилно.

Архитектурата на P-TOSCA PaaS платформата содржи три генерални архитектурни елементи:

- *Интерфејсен модул* кој служи како интерфејсот кон клиентот. Се состои од веб апликација за директна комуникација со корисникот и веб сервис кој може да биде употребен од корисникот или од друга P-TOSCA PaaS инстанца.
- *P-TOSCA централен модул* кој ја процесира и извршува спецификацијата. Содржи процесор на P-TOSCA моделот , извршувач на артефакти, комуникатор со облакот и процесор на архиви. Овој модул користи и надворешни библиотеки.
- *Податочен модул* кој ја содржи историјата на корисничките активности.



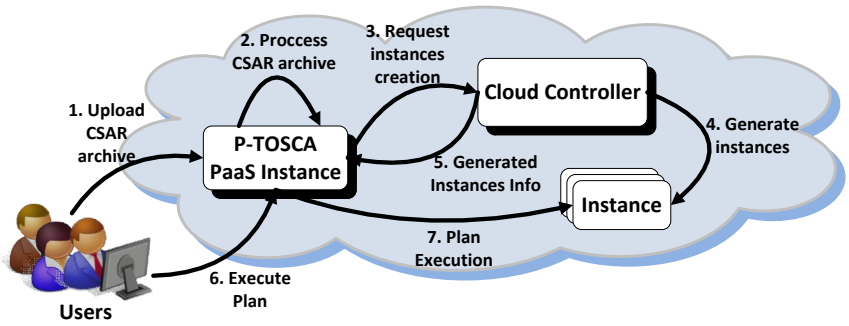
Слика 4.2 Архитектура на P-TOSCA PaaS имплементацијата

Концептуалниот дијаграм на TOSCA базираната платформа, вклучувајќи ги и интеракции, е прикажан на Слика. 4.3 и прикажува општо сценарио на обработка и примена на архива со користење на оваа платформа. Сценариото ги вклучува следните чекори:

1. Корисникот/клиентот пристапува и ја предава CSAR архивата од својата апликација.
2. Откако архивата целосно ќе биде доставена платформата започнува со обработка и најпрво го гради тополошкиот модел.
3. Потоа комуницира со контролерот на облакот и според тополошкиот модел ги креира дефинираните инстанци.
4. Откако платформата ќе прими информација за креираните инстанци, ги процесира плановите и му нуди на корисникот избор кој план да го изврши.
5. Клиентот избира план.
6. Платформата го извршува планот преку директна комуникација со инстанците вклучени во планот.

4.5.2 P-TOSCA централен модул

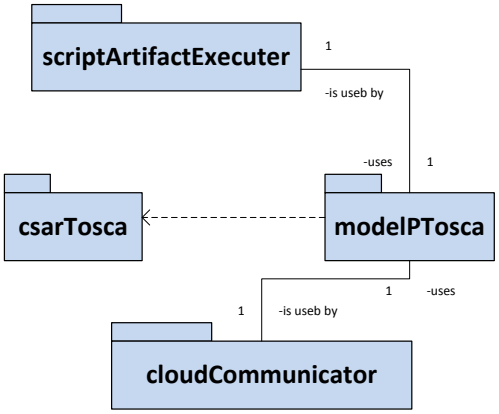
P-TOSCA централениот модул извршува неколку функции:



Слика 4.3 Концептуален дијаграм на TOSCA базирана Платформа како Сервис

- процесира CSAR архиви,
- ги генерира елементите на P-TOSCA моделот, и
- извршува планови дефинирано со моделот.

Овој модул е имплементиран во Java и генералната структура на овој елемент е прикажана на Слика. 4.4 Се состои од четири основни пакети: csarTosca, modelTosca , cloudCommunicator и scriptArtifactExecuter.



Слика 4.4 Структура на TOSCA API моделот

4.5.3 Процесор на CSAR архиви

Дел од структурата на CSAR архивата е дефинирана од страна на TOSCA [91] и претставува zip документ кој мора да содржи TOSCA-Metadata директориум (with TOSCA.meta документ) и Definitions директориум. Креаторот на архивата може да постави и друга содржина.

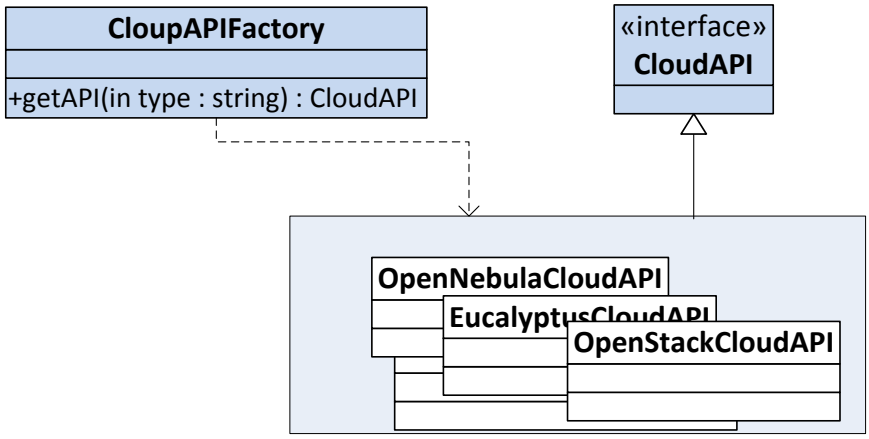
Пакетот csarTosca ја отпакува CSAR аривата, ја потврдува структурата на архивата и го обработува TOSCA.meta документот за да добие листа на дефиниции за апликацијата. Се состои од три Java класи, чии имиња се јасни: csarUnpacker, csarVerifier and csarProcessMeta.

4.5.4 Комуникација со облак

Пакетот cloudCommunication ги содржи класите кои се потребни за комуникација со контролерот на облакот. Овој пакет е изграден врз основа на шаблонот Фабрички метод, т.е. содржи т.н. фабричка класа (CloupAPIFactory) која креира објекти од тип CloudAPI (кој е интефејс). CloudAPI интерфејсот декларира методи за креирање на инстанци, алокација на IP адреси, безбедносни поставувања и останати параметри за инстанци. Се дефинираат класи за различни типови на системи на облак кои го имплементираат CloudAPI интерфејсот (и неговите методи). Овие класи подоцна се употребуваат од страна на modelTosca пакетот да креираат и конфигурираат инстанци од топологијата. Класите комуницираат директно со APIто на контролерот на облакот или употребуваат дополнителни библиотеки за оваа активност. Слика. 4.5 ја прикажува архитектурата на cloudCommunication пакетот.

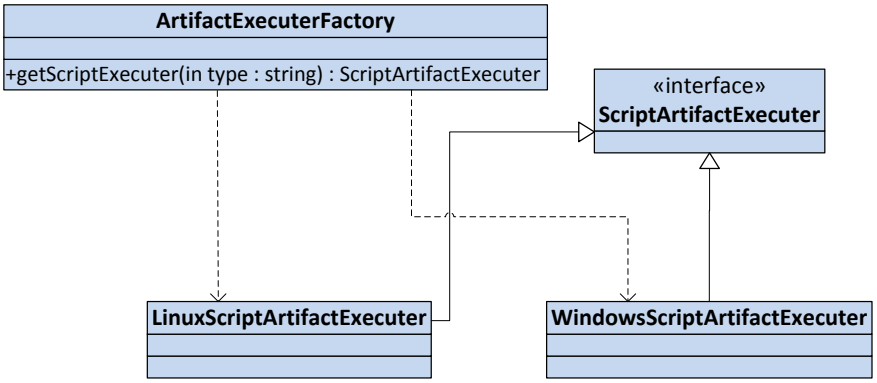
4.5.5 Извршување на артефакти со скрипти

Пакетот scriptArtifactExecuter содржи класи кои се употребуваат за извршување на артефакти со скрипти на инстанци. Овој пакет исто така е изграден врз основа на шблонот Фабрички метод, т.е. содржи т.н.фабричка класа (ArtifactExecuterFactory) која креира објекти од тип ScriptArtifactExecuter (кој е интефејс). ScriptArtifactExecuterинтерфејсот декларира методи за извршување на скрипти. Две класи го имплементираат овој интерфејс: LinuxScriptArtifactExecuter и WindowsScriptArtifactExecuter. Овие класи динамички градат скрипти за да ги имплементираат артефактите со скрипти на новите инстанци. Употребуваат ssh и powershell соодветно. Слика. 4.6 ја прикажува архитектурата на cloudCommunication пакетот.



Слика 4.5 Структура на Cloud Communicator пакетот

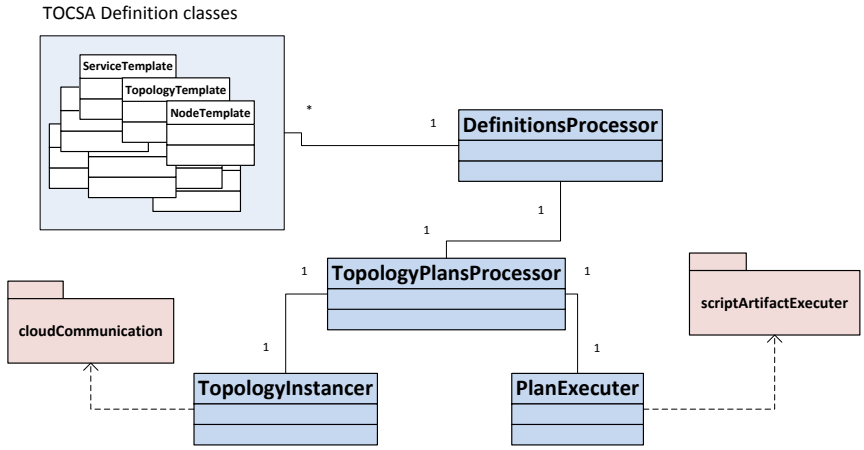
За извршување на скрипти на Windows Server се употребува преинсталиран PowerShell сервер.



Слика 4.6 Структура на Script Artifacts Execution пакетот

4.5.6 Обработка на TOSCA моделот

Пакетот за обработка на TOSCA моделот (modelTosca) е најкомплексниот пакет во архитектурата и има многу одговорности. Архитектурата на пакетот е прикажана на Слика. 4.7. Пакетот содржи:



Слика 4.7 Структура на пакетот за TOSCA моделот

- Дефиниција на класите од TOSCA XML шемата;
- DefinitionsProcessor класа која ги валидира дефинициите и ги обработува во соодветни објекти
- TopologyPlansProcessor класа која генерира објекти за топологија и планови според резултатот DefinitionsProcessor, а воедно и ги процесира зависностите/врските меѓу објектите;
- TopologyInstancer класа која креира инстанци од генерираната топологија со употреба на класите од cloudCommunication пакетот и
- PlanExecuter класа која извршува даден план со употреба на класите од scriptArtifactExecuter пакетот.

4.5.7 Претставување на TOSCA моделот

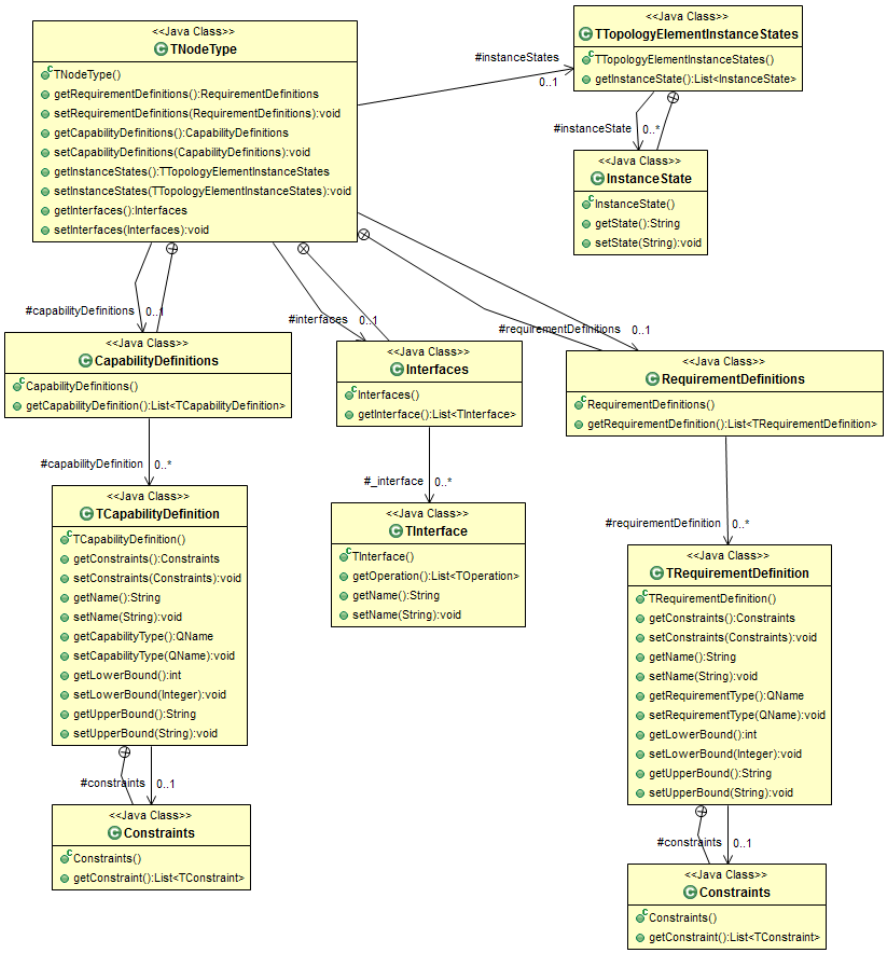
TOSCA XML schema-та содржи голем број на дефиниции и за сите се креирани соодветни класи. Во овој дел се прикажани само дел од (најбитните) класи.

Слика. 4.8 ги прикажува дел од класите поврзани со дефиницијата на типови на јазли (NodeType).

Слика. 4.9 ги прикажува дел од класите поврзани со дефиницијата на типови на релации (RelationshipType).

Слика. 4.10 ги прикажува дел од класите поврзани со дефиницијата на тополошки јазли (NodeTemplate).

Слика. 4.11 ги прикажува дел од класите поврзани со дефиницијата на релациите на тополошките јазли (RelationshipTemplate).



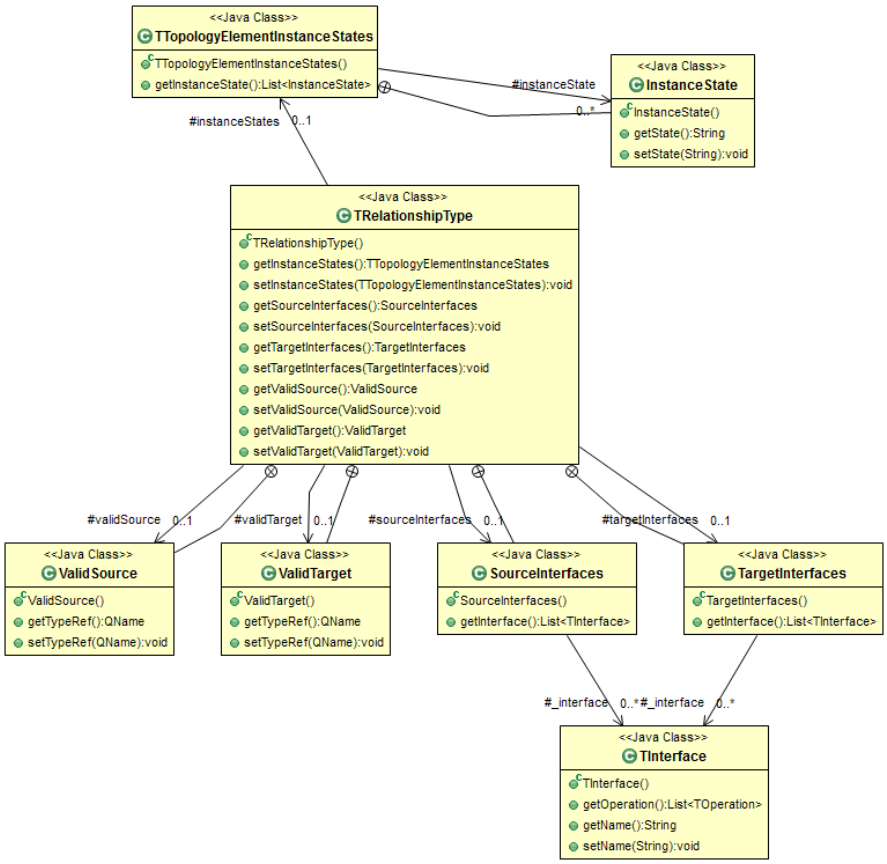
Слика 4.8 Дел од класи за дефиниција на тип на јазли

Слика. 4.12 ги прикажува дел од класите поврзани со дефиницијата на плановите (Plans).

Слика. 4.13 ги прикажува дел од класите поврзани со дефиницијата на сервисот со тополошки јазли и планови (ServiceTemplate).

4.5.8 Интерфејс и употреба на платформата

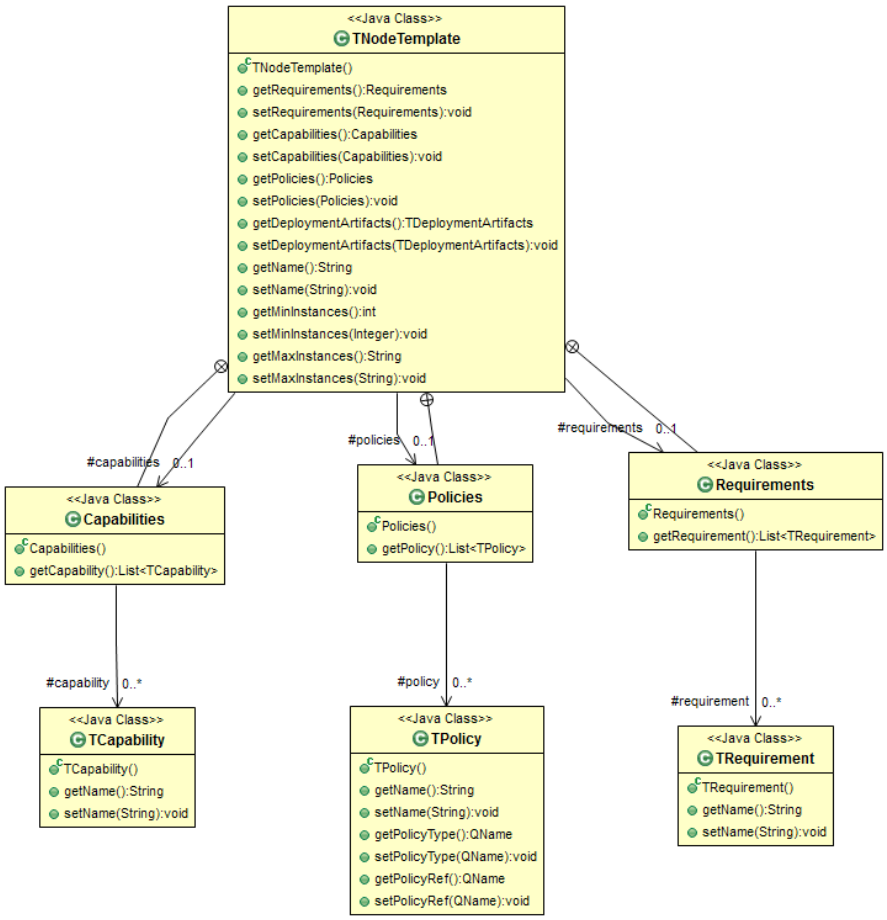
Платформата нуди веб интерфејс на корисниците, кој е JSP servlet базирана веб апликација. Интерфејсот е интерактивен и ги води корисниците низ процесор. Во оваа апликација jsp документите го претставуваат



Слика 4.9 Дел од класи за дефиниција на релации меѓу типови на јазли

графичкиот интерфејс, а servlet-те комуницираат со моделот model. Навигацијата е едноставна и ги вклучува следните чекори:

1. Процесот започнува со најава на корисникот на облакот.
2. Продолжува со достава на CSAR архивата.
3. Корисникот ја потврдува достава и избира да се отпакува и валидира архивата и да се обработи моделот.
4. Откако ќе се изгради моделот се бара потврда од корисникот за генерирање на инстанци според доставените дефиниции. Притоа се дава предупредување дека во текот на овој процес корисникот нема да има пристап до новите инстанци бидејќи ќе се употребат специфични безбедносни конфигурации и корисникот ќе може да превземе контрола на крај на процесот.
5. Со потврда на корисникот се креираат инстанците и корисникот добива информација за нивните карактеристики. Притоа се дава листа

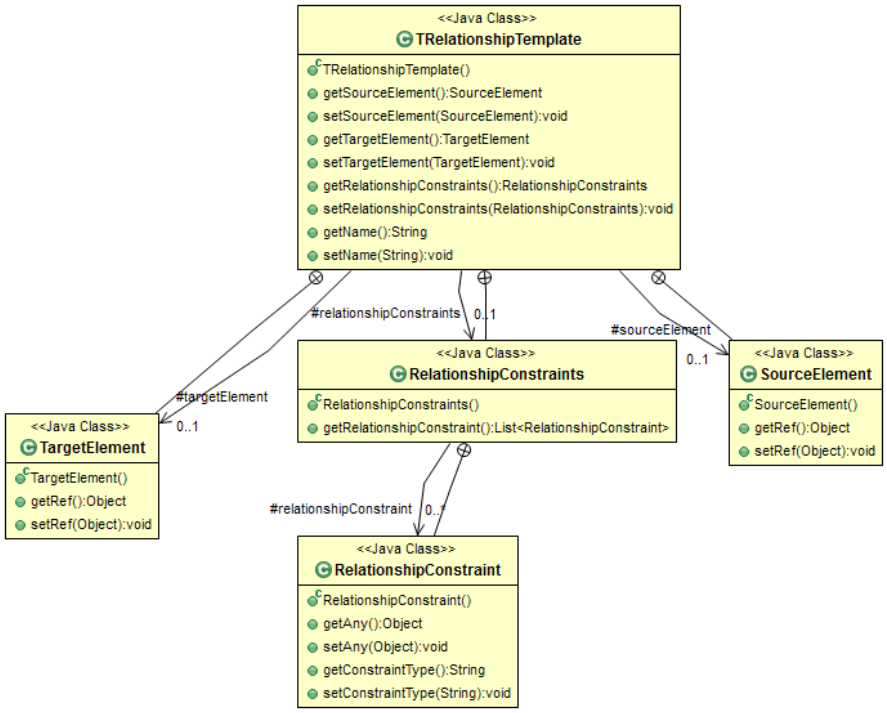


Слика 4.10 Дел од класи за дефиниција на тополошки јазли

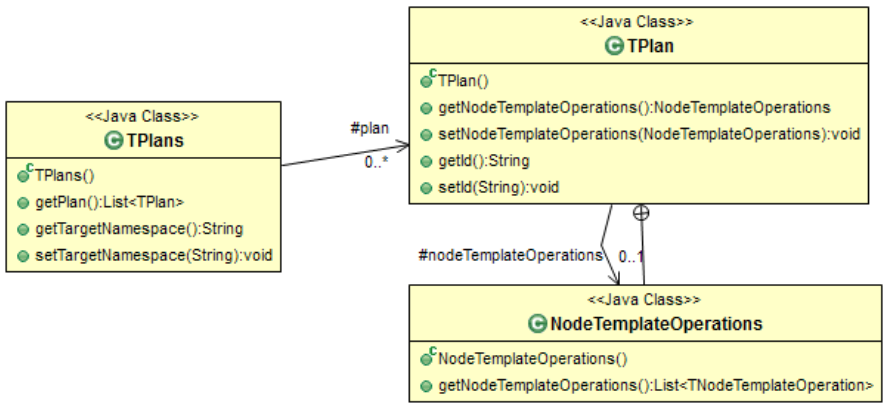
од дефинирани планови и корисникот треба да одбере план кој сака да се изврши.

6. Избраниот план се извршува и корисникот добива целосен извештај на сите извршени активности. Повторно се нуди избор на корисникот да избере план за извршување или за завршување на процесот.
7. Во моментот кога корисникот избира да го заврши процесот моделот ги применува дефинираните безбедносни побарувања од корисникот и доколку е избрано генерирање на нов безбедносен клуч, истиот се дава на корисникот.

Целиот процес на навигација и интеракција меѓу корисникот, веб апликацијата и моделот е претставен на секвенцен дијаграм на Слика. 4.14 Овој дијаграм ги покажува стандардните чекори на извршување (т.е. не

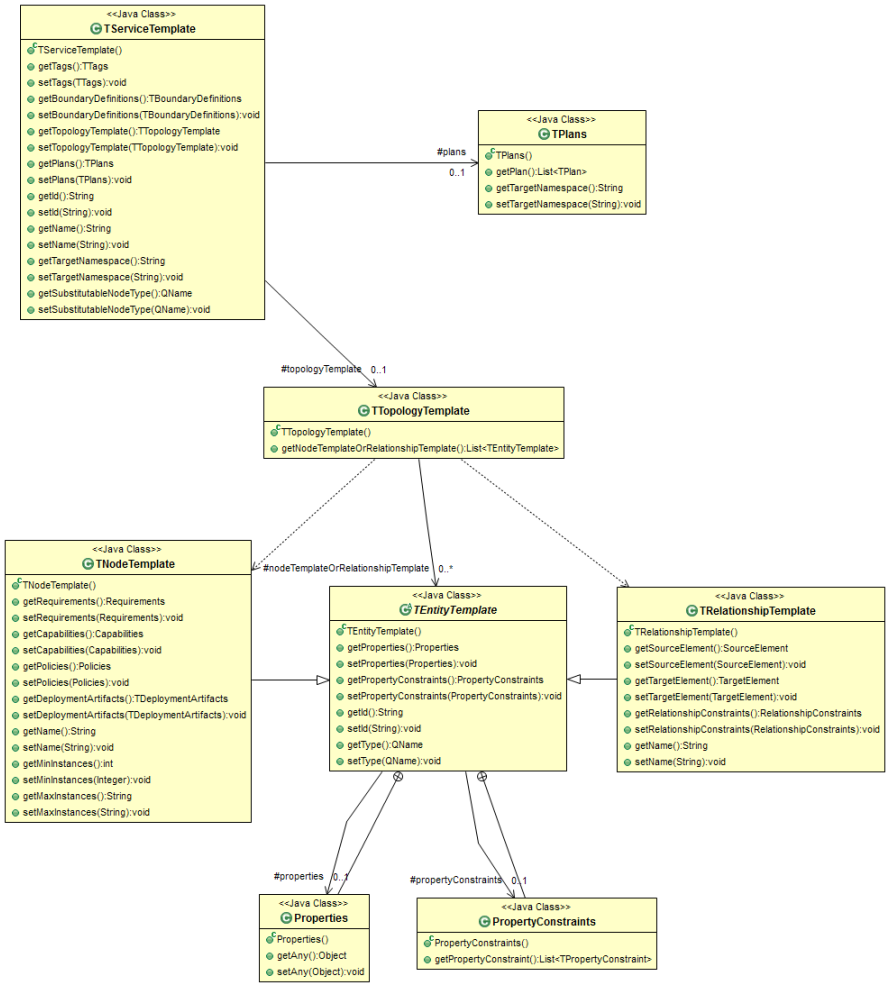


Слика 4.11 Дел од класи за дефиниција на релации меѓу тополошки јазли



Слика 4.12 Дел од класи за дефиниција на планови

ги вклучува алтернативни чекор во случај на грешка).Дијаграмот е воедно поедноставен, прикажува пакети наместо класи, и повратните повици не се прикажани за прегледност и едноставност.

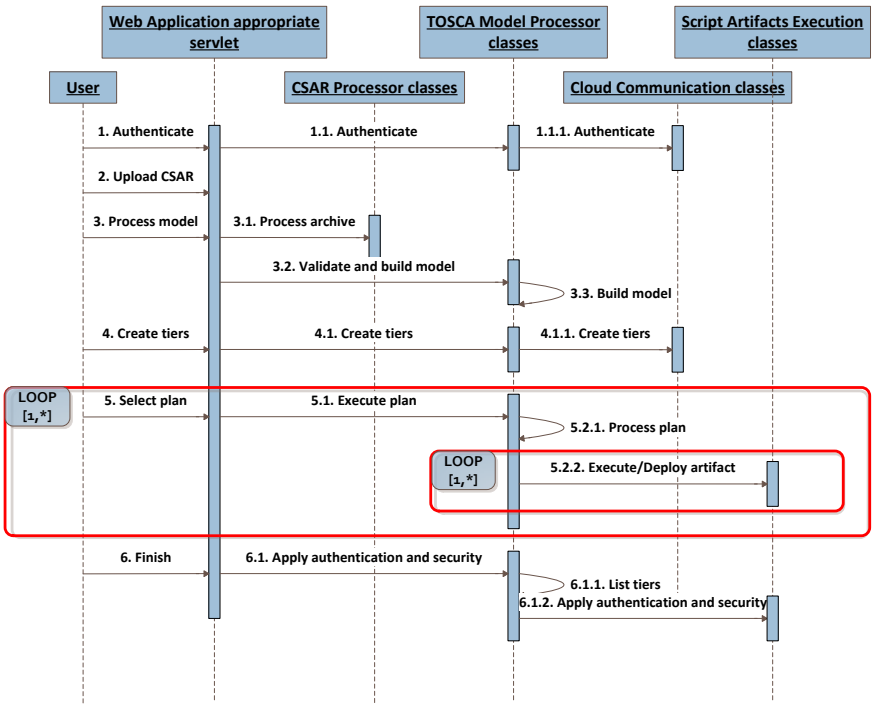


Слика 4.13 Дел од класи за дефиниција на сервис со тополошки јазли и планови

Оваа платформа нуди и REST веб сервиси за да овозможи P-TOSCA PaaS инстанците да комуницираат, прикажани во Табела. 4.1.

Табела 4.1 Веб сервиси за комуникација меѓу платформите

Сервис	**Опис**
auth	Сервис кој извршува автентикација на корисник
definition	Сервис кој враќа дефиниции за дадена апликација



Слика 4.14 Секвенцен дијаграм на примена на TOSCA платформата

4.6 Дискусија

Во овој дел одделно ќе се разгледуваат предности и недостатоците на TOSCA моделот и платформата која е креирана.

4.6.1 TOSCA концепт и предности и недостатоци за примена

Бидејќи TOSCA е нов стандард и сеуште е во процес на развој, се започна од идеја да се истражи на можноста на TOSCA за примена. При овој процес се идентификувани критични точки каде е потребен понатамошен развој на TOSCA. Во моментов не постојат комерцијални решенија што ја поддржуваат примената на TOSCA стандардот, единствено тековно се случуваат истражувачки напори [72], [7]. постои иницијална алатка за моделирање на топологија [57, 8]. Прифаќање на овој стандард не може да се гарантира, но може да се очекува како резултат на флексибилноста

што ја нуди само ако се понудат доволно докази за примена на концептот и средини за тестирање во облак од страна на провајдерите.

Употреба на општа наменски јазик како BPMN со цел да се дефинираат плановите за управување со животниот циклус, исто така, е уште една критична точка која се бара понатамошно додефинирање. Во овој момент дефиниција на план со BPMN е многу апстрактна и тешко може да се изврши, бидејќи побарува преведување и извршување во специфична околина на облак и асоцијација со топологијата. Во моментов група на истражувачи работи за да се прошири BPMN со TOSCA-специфични елементи цитираат BPMN4TOSCA.

Создавање на CSAR архива не е лесна задача во овој момент и се должи на фактите споменати во ова поглавје претходно. Улогата на TOSCA CSAR архитект бара значителни вештини за создавање артефакт и дополнително вештини за генерирање на скрипти. CSAR може да содржи и извршни и бинарни артефакти (како инсталациони датотеки итн ..), што може да резултира со големи архиви. Секоја грешка во архивата ќе бара измена и достава која може да трае долго. Од таа причина облак провајдерите треба да понуди тестирање средини за декларативна обработка на архивата, пред вистинскиот процес на примена да се случи.

Сепак, други проблеми вклучуваат моделирање апликации кои користат некои комерцијални софтверски пакети и сервери кои не се лесно да се конфигурираат користење на скрипти и создавање на безбедна средина.

Предноста на TOSCA моделот е повторна употреба во различни облак средини и фактот дека голем број на софтверска архитектура стилови можат да се моделираат со овој модел. Флексибилноста на овој модел нуди лесно создавање на топологија за помали апликации. Од друга страна креирање на модел за големи и комплексни апликации може да биде тешка задача, без алатка за моделирање со интуитивен и јасен интерфејс.

4.6.2 Предности и недостатоци на P-TOSCA базирана PaaS

P-TOSCA е проширена спецификација на TOSCA која е создадена во рамки на ова истржување за да се надминат некои од претходно именуваните проблеми. Покрај тоа, создавањето на тополошките елементи како инстанци овозможува понатамошна употреба како класичен облак IaaS пристап, паралелно со користењето TOSCA базираната PaaS. Платформата нуди лесен и интуитивен интерфејс кој го води корисникот низ процесот и го информира за извршените задачи и резултати. Употребата на шаблонот фабрика модел кој се користи за управување со комуникација со облак API и со артефакти овозможува проширливост за вклучување на нови облак системи или управување на артефакт типови.

Нашата примарна грижа е да се најде автоматско решение за извршување на артефакти на Windows сервери, без користење на преинсталиран PowerShell сервер на Windows Server инстанца.

Специфична карактеристика на платформата е фактот дека корисникот немаат контрола врз инстанците во текот на процесот на примена, бидејќи платформа користи специфични безбедносни политики и автентикација во текот на овој процес. По завршување на овој процес контролата се враќа назад на корисникот. Ова е се воведува заради спречување на ранливост во текот на процес на примена.

Друга специфична карактеристика е секвенцијалната декларација и извршување на планови која се разликува од TOSCA првичната идеја.

4.7 Главен придонес

Во оваа глава во детали се идентификуваат недостатоците на сегашната TOSCA спецификација и се воведуваат нови елементи и проширувања на оригиналните, со цел да се создаде целосна спецификација која може да се обработи и изврши. Креирана е работна платформа која може да ја процесира оваа спецификација, која засега е единствен пример на извршување на TOSCA моделот.

Глава 5
Применливост и користење на P-TOSCA моделот

Преглед Во оваа глава се прикажува употребата на TOSCA базираната платформа над различни типови на софтверски архитектури. Примената вклучува дефиниција на TOSCA моделот (топологија и артефакти).

5.1 Вовед

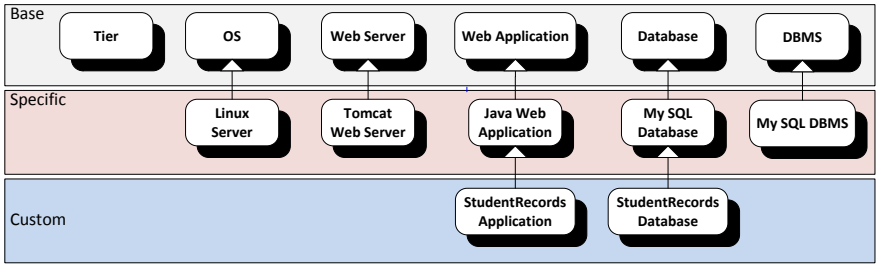
TOSCA базираната платформа може да се користи за различни стилови на софтверски архитектури. За таа цел успешно се генерирани модели на за дистрибуирани слоевити апликации и сервисно ориентирани апликации. Во овој дел се опишуваат дефинициите кои се вклучени за овие типови и дел од нивните придружни артефакти.

5.2 Слоевити (N-Tier) апликации

Моделот на слоевити (N-tier) апликации често се користи во реалниот живот и тоа многу често во вид на дистрибуирани апликации. Од таа причина ние одбравме 2-слојни и 3-слојни апликации за тестирање на користење на платформа.

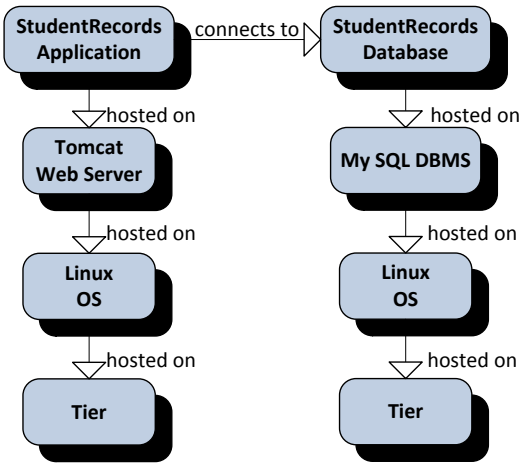
5.2.1 2-слојни апликации

Во рамките на ова истражување е направена Java MVC симплифицирана апликација за управување со студентски досиеја. Се состои од 2 слоја: слој на база на податоци и слој кој ги содржи презентација и логика на апликацијата. Првиот слој за база на податоци е MySQL систем за упра-



Слика 5.1 Типови на јазли и хиерархија за апликација за управување со студентски досиеја

вување со бази на податоци и во овој слој се поставени податоците кои ги користи апликацијата. Вториот слој е Tomcat веб сервер кој е домаќин на презентациското и логичкото ниво на апликација. Интерфејсот на апликацијата се состои од JSP датотеки и со акциите управуваат сервлети кои комуницираат со класите на моделот (кои ја претставуваат логиката на апликацијата). Генерирана типови на јазли (основни, специфични и произволни) за оваа апликација се прикажани на Слика. 5.1. Тополошката дефиниција за оваа апликација е прикажана на Слика. 5.2. Сите основни видови на јазли се добиени од типот на root јазол.



Слика 5.2 Топологија на апликација за управување со студентски досиеја

Код. 5.1 прикажува дел од дефинициите на типови на јазли за апликација за управување со студентски досиеја. Поради едноставност на приказ namespace-те не се прикажуваат.

```
<NodeType name="WebServer">
    <documentation>Web Server</documentation>
    <DerivedFrom typeRef="RootNodeType"/>
    <RequirementDefinitions>
      <RequirementDefinition lowerBound="1" name="container"
        requirementType="SoftwareContainerRequirement" upperBound="1"/>
    </RequirementDefinitions>
    <CapabilityDefinitions>
      <CapabilityDefinition
        capabilityType="WebApplicationContainerCapability"
        lowerBound="0" name="webapps" upperBound="unbounded"/>
    </CapabilityDefinitions>
</NodeType>
<NodeType name="WebApplication">
    <documentation>Web Application</documentation>
    <DerivedFrom typeRef="RootNodeType"/>
    <RequirementDefinitions>
      <RequirementDefinition lowerBound="1" name="container"
        requirementType="WebApplicationContainerRequirement" upperBound="1"/>
    </RequirementDefinitions>
</NodeType>
<NodeType name="TomcatWebServer">
    <documentation>Tomcat Web Server</documentation>
    <DerivedFrom typeRef="WebServer"/>
    <PropertiesDefinition element="TomcatWebServerProperties"/>
    <CapabilityDefinitions>
      <CapabilityDefinition
        capabilityType=TomcatWebApplicationContainerCapability"
        lowerBound="0" name="webapps" upperBound="unbounded"/>
    </CapabilityDefinitions>
    <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
        <Operation name="configure"/>
        <Operation name="start"/>
        <Operation name="stop"/>
        <Operation name="uninstall"/>
      </Interface>
    </Interfaces>
</NodeType>
<NodeType name="JavaWebApplication">
    <documentation>Tomcat Web Application</documentation>
    <DerivedFrom typeRef="WebApplication"/>
    <RequirementDefinitions>
      <RequirementDefinition lowerBound="1" name="container"
        requirementType="TomcatWebApplicationContainerRequirement"
              upperBound="1"/>
    </RequirementDefinitions>
</NodeType>
<NodeType name="StudentRecordsApplication">
    <documentation>StudentRecords Application</documentation>
    <DerivedFrom typeRef="JavaWebApplication"/>
    <PropertiesDefinition element="StudentRecordsApplicationProperties"/>
    <RequirementDefinitions>
```

```
    <RequirementDefinition lowerBound="1" name="database"
      requirementType="MySQLDatabaseEndpointRequirement"
          upperBound="1"/>
  </RequirementDefinitions>
  <Interfaces>
    <Interface name="lifecycle">
      <Operation name="install"/>
      <Operation name="configure"/>
      <Operation name="start"/>
      <Operation name="uninstall"/>
    </Interface>
  </Interfaces>
</NodeType>
```

Код 5.1 Дефиниција на типови на јазли за апликација за управување со студентски досиеја

Код. 5.2 ја прикажува дефиницијата на тополошкиот јазол за сервер за Tomcat.

```
<NodeTemplate id="VmTomcat" name="VM for Tomcat" type="LinuxServer">
      <Properties>
        <ServerProperties>
            <NumCpus>1</NumCpus>
            <Memory>1024</Memory>
            <Disk>10</Disk>
            <InitialNumInstances>
                1
            </InitialNumInstances>
            <ServerIPAddress>
                194.149.135.5
                <!-- This property
                    will be ignored -->
            <ServerIPAddress>
        </ServerProperties>
      </Properties>
      <Requirements>
        <Requirement id="VmTomcat_container" name="container"
          type="ServerContainerRequirement"/>
      </Requirements>
      <Capabilities>
        <Capability id="VmTomcat_os" name="os"
          type="OperatingSystemContainerCapability"/>
      </Capabilities>
</NodeTemplate>
```

Код 5.2 Дефиниција на Tomcat сервер тополошки јазол

Код. 5.3 ја прикажува дефиницијата на тип на скриптирачки артефакт, како и дефиниција на артефакти за инсталација и конфигурација на апликација.

```
<ArtifactType name="ScriptArtifact">
  <documentation>Script Artifact</documentation>
  <DerivedFrom typeRef="RootArtifactType"/>
```

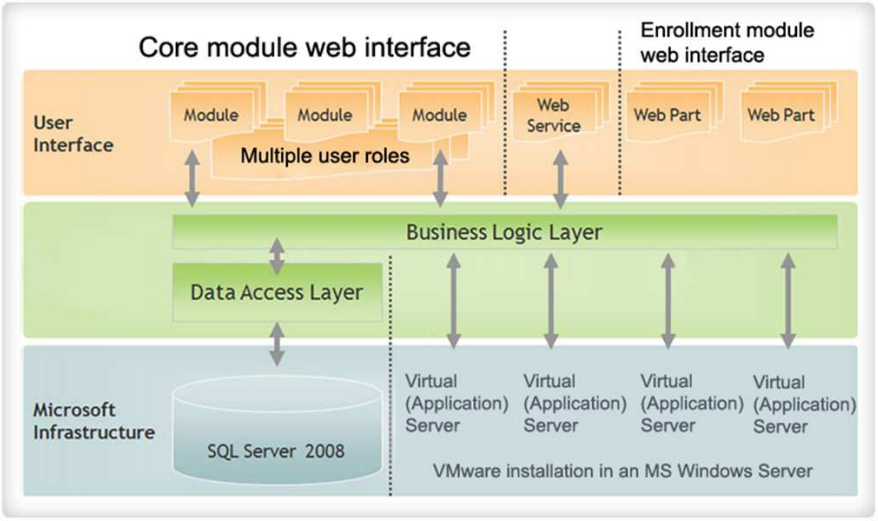
```
    <PropertiesDefinition element="ScriptArtifactProperties"/>
</ArtifactType>

<ArtifactTemplate id="srapp-installsh" type="ScriptArtifact">
    <Properties>
      <ScriptArtifactProperties>
        <ScriptLanguage>sh</ScriptLanguage>
        <PrimaryScript>scripts/StudentRecordsApplication/install.sh</PrimaryScript>
      </ScriptArtifactProperties>
    </Properties>
    <ArtifactReferences>
      <ArtifactReference reference="scripts/StudentRecordsApplication">
        <Include pattern="install.sh"/>
      </ArtifactReference>
    </ArtifactReferences>
</ArtifactTemplate>

<ArtifactTemplate id="srapp-configsh" type="ScriptArtifact">
    <Properties>
      <ScriptArtifactProperties>
        <ScriptLanguage>sh</ScriptLanguage>
        <PrimaryScript>scripts/StudentRecordsApplication/configure.sh</PrimaryScript>
                <InputParametars>
                    <InputParametar nodeTemplateId="VmMySql"
                        property="ServerIPAddress"/>
                </InputParametars>
      </ScriptArtifactProperties>
    </Properties>
    <ArtifactReferences>
      <ArtifactReference reference="scripts/MVCServletApplication">
        <Include pattern="configure.sh"/>
      </ArtifactReference>
    </ArtifactReferences>
</ArtifactTemplate>
```

Код 5.3 Дефиниција на тип на артефакт и артефакти за инсталација и конфигурација на апликацијата

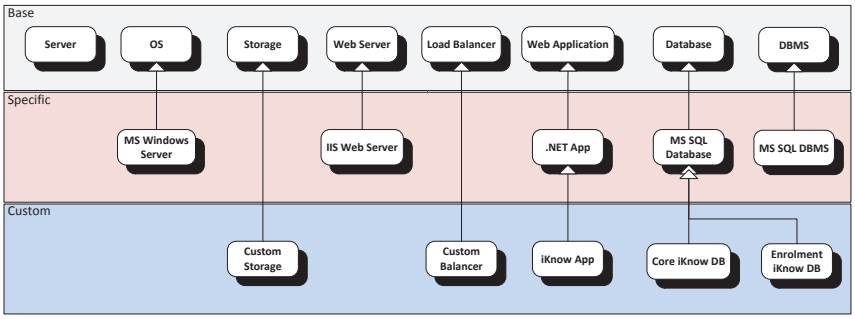
Покрај дефинициите, архивата вклучува и артефакт на апликацијата (war датотека), скриптирачки артефакти за креирање на базата на податоци и иницијални податоци, како и за скриптирање артефакти Tomcat и MySQL инсталација и конфигурација.

5.2.2 3-слојни апликации

Како пример за 3-слојна апликација е избран Универзитетскиот систем iKnow [10], кој сам по себе претставува специфично решение поради архитектурата (прикажана на Слика. 5.3) и моделот на примена кои ги користи. Генерирана типови на јазли (основни, специфични и произволни) за оваа апликација се прикажани на Слика. 5.4.



Слика 5.3 Архитектура на iKnow [10]



Слика 5.4 Типови на јазли и хиерархија за iKnow

Код. 5.4 прикажува дел од дефинициите на основни јазли за iKnow. Поради едноставност на приказ namespace-те не се прикажуваат.

```
<NodeType name="Server">
        <DerivedFrom typeRef="RootNodeType"/>
      <RequirementDefinitions>
        <RequirementDefinition lowerBound="0" name="container"
requirementType="ServerContainerRequirement"
upperBound="1"/>
      </RequirementDefinitions>
        ...
</NodeType>
<NodeType name="WebServer">
        <DerivedFrom typeRef="RootNodeType"/>
        <Operation name="install"/>
```

```
        <Operation name="configure"/>
        <Operation name="start"/>
        <Operation name="stop"/>
        <Operation name="uninstall"/>
        ...
</NodeType>
<NodeType name="LoadBalancer">
        <DerivedFrom typeRef="RootNodeType"/>
            <Operation name="install"/>
        <Operation name="configure"/>
        <Operation name="start"/>
        <Operation name="stop"/>
        <Operation name="uninstall"/>
        ...
</NodeType>
<NodeType name="WebApplication">
        <DerivedFrom typeRef="RootNodeType"/>
        <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
        <Operation name="configure"/>
        <Operation name="uninstall"/>
      </Interface>
    </Interfaces>
        ...
</NodeType>
<NodeType name="Database">
        <DerivedFrom typeRef="RootNodeType"/>
        <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
        <Operation name="uninstall"/>
        <Operation name="start"/>
        <Operation name="stop"/>
      </Interface>
    </Interfaces>
        ...
</NodeType>
<NodeType name="DBMS">
        <DerivedFrom typeRef="RootNodeType"/>
        <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
        <Operation name="configure"/>
        <Operation name="start"/>
        <Operation name="stop"/>
        <Operation name="uninstall"/>
      </Interface>
    </Interfaces>
        ...
</NodeType>
```

Код 5.4 Дефиниција на основни типови на јазли

Код. 5.5 ја прикажува дефиницијата на IIS Web Server јазелот, кој е специфичен тип на јазол.

```
<NodeType name="IISWebServer">
    <DerivedFrom typeRef="WebServer"/>
    <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
          <InputParameters>
                <InputParameter name="httpport" type="integer"/>
                <InputParameter name="username" type="string"/>
                <InputParameter name="password" type="string"/>
        ...
      </Interface>
    </Interfaces>
</NodeType>
```

Код 5.5 Дефиниција на IIS Web Server јазол

Код. 5.6 ја прикажува дефиницијата на Core database јазолот, кој е од тип на произволен јазол.

```
<NodeType name="CoreiKnowDB">
    <DerivedFrom typeRef="MSSQLDB"/>
    <Interfaces>
      <Interface name="lifecycle">
        <Operation name="install"/>
        <Operation name="start"/>
        <Operation name="stop"/>
        <Operation name="uninstall"/>
        <Operation name="backup"/>
      </Interface>
    </Interfaces>
</NodeType>
```

Код 5.6 Дефиниција на Core DB јазолот

Код. 5.7 а прикажува дефиницијата на тополошкиот јазол за IIS Web Server.

```
<NodeTemplate id="IISWebServer"
        name="IIS Web Server"
        type="IISWebServer">
    <Properties>
        <httpport>80</httpport>
        <username>sa</username>
        <password>sa</password>
    </Properties>
</NodeTemplate>
```

Код 5.7 Дефиниција на IIS Web Server тополошки јазол

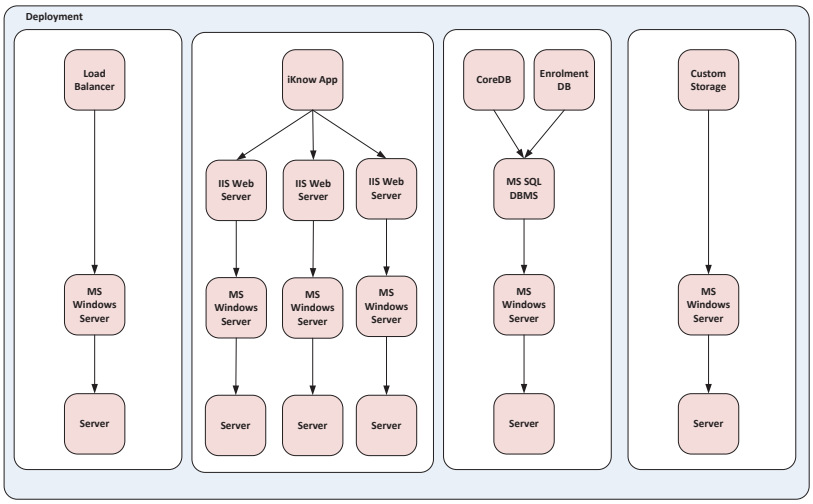
Код. 5.8 ја опишува логичката релација меѓу јазли, додека Код. 5.9 опишува конкретна релација (врска) iKnow Core Database и MS SQL

тополошките јазлите со употреба на својствата за побарувања и способности.

```
<RelationshipType name="DeployedOn">
    <DerivedFrom typeRef="RootRelationshipType"/>
    <ValidSource typeRef="WebApplication"/>
    <ValidTarget typeRef="WebServer"/>
</RelationshipType>

<RelationshipType name="ConnectsTo">
    <DerivedFrom typeRef="RootRelationshipType"/>
    <ValidSource typeRef="WebApplication"/>
    <ValidTarget typeRef="Database"/>
</RelationshipType>

<RelationshipType name="HostedOn">
    <DerivedFrom typeRef="RootRelationshipType"/>
    <ValidSource typeRef="Database"/>
    <ValidTarget typeRef="DBMS"/>
</RelationshipType>

<RelationshipType name="CoreDBHostedOnMSSQLDBMS">
    <DerivedFrom typeRef="HostedOn"/>
    <SourceInterfaces>
      <Interface name="HostedOn">
        <Operation name="hostOn"/>
      </Interface>
    </SourceInterfaces>
    <ValidSource typeRef="CoreiKnowDB"/>
    <ValidTarget typeRef="MSSQLDBMS"/>
</RelationshipType>

<RelationshipType name="iKnowAppDeployedOnIISWebServer">
    <DerivedFrom typeRef="DeployedOn"/>
    <SourceInterfaces>
      <Interface name="DeployedOn">
        <Operation name="deployOn"/>
      </Interface>
    </SourceInterfaces>
    <ValidSource typeRef="iKnowApp"/>
    <ValidTarget typeRef="IISWebServer"/>
</RelationshipType>
```

Код 5.8 Пример на опис на релации за iKnow

```
<RelationshipTemplate id="CoreDB_HostedOn_MSSQLDBMS"
     name="hosted on" type="CoreDBHostedOnMSSQLDBMS">
     <SourceElement ref="CoreDB_container"/>
     <TargetElement ref="MSSQLDBMS_databases"/>
</RelationshipTemplate>

<NodeTemplate id="CoreDatabase"
        name="iKnowCoreDB"
        type="CoreDB">
```



Слика 5.5 Сценарио за примена на iKnow

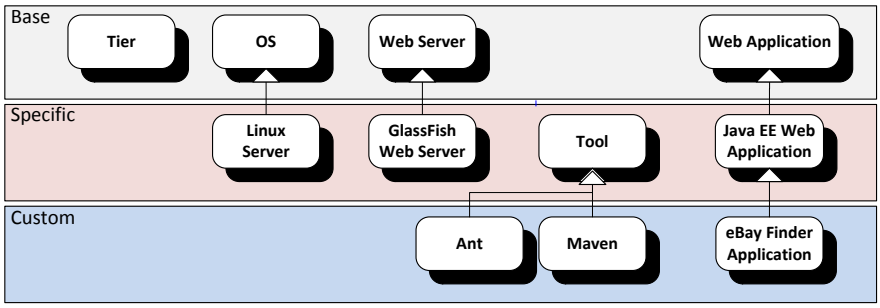
```
    <Properties>... </Properties>
    <Requirements>
      <Requirement id="CoreDB_container" name="container"
                   type="MSSQLDBContainerReq"/>
    </Requirements>
    <Capabilities>...</Capabilities>
</NodeTemplate>

<NodeTemplate id="MSSql" name="MS_SQL" type="MySQL">
    <Properties>... </Properties>
    <Requirements>... </Requirements>
    <Capabilities>
      <Capability id="MSSql_databases" name="databases"
                  type="MSSQLDBContainerCap"/>
    </Capabilities>
  </NodeTemplate>
```

Код 5.9 Пример за релација меѓу тополошки јазли за iKnow

Слика. 5.5 го прикажува сценарио за примена на топологија за iKnow од дадената спецификација.

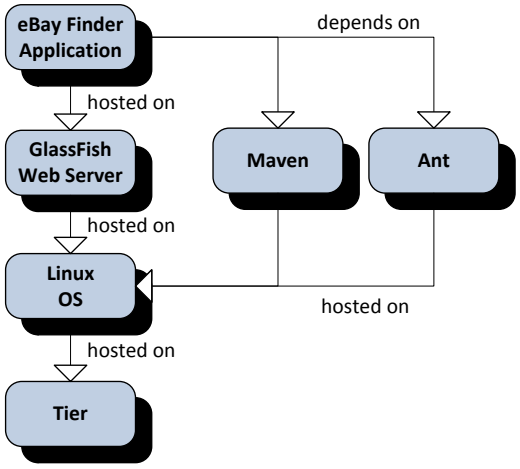
5.3 Сервисно-ориентирани апликации

За овој тип на апликации работиме со модифицирана верзија (и само за eBay) на SOA Shopper проектот презентиран во книгата "SOA Using



Слика 5.6 Типови на јазли и хиерархија за eBay Finder SOA апликацијата

Java Web Services"[38]. Апликацијата го употребува eBay SOAP APIто [23] за да добие податоци за понуди на eBay и нуди различни типови на консументи на услугите кои ги нуди (вклучувајќи и веб прелистувач). Генерирана типови на јазли (основни, специфични и произволни) за оваа апликација се прикажани на Слика. 5.6. Тополошката дефиниција за оваа апликација е прикажана на Слика. 5.7. Сите основни видови на јазли и специфичниот тип Tool се добиени од типот на root јазол.



Слика 5.7 Топологија на eBay Finder SOA апликацијата

Код. 5.10 ја прикажува дефиницијата на тополошкиот јазол за GlassFish апликацискиот сервер.

```
<NodeTemplate id="GlassFishWebServer"
        name="GlassFish Web Server"
        type="GlassFishWebServer">
    <Properties>
```

```
            <httpport>8080</httpport>
            <httpsport>8181</httpsport>
            <username>admin</username>
            <password>adminadmin</password>
    </Properties>
</NodeTemplate>
```

Код 5.10 Дефиниција на GlassFishWebServer тополошки јазол

Код. 5.11 ја прикажува дефиницијата на артефакт за конфигурација на GlassFish апликацискиот сервер.

```
<ArtifactTemplate id="glassfish-configsh" type="ScriptArtifact">
    <Properties>
      <ScriptArtifactProperties>
        <ScriptLanguage>sh</ScriptLanguage>
        <PrimaryScript>scripts/GlassFishWebServer/configure.sh</PrimaryScript>
            <InputParametars>
                <InputParameter nodeTemplateId="GlassFishWebServer"
                    property="httpport"/>
                <InputParameter nodeTemplateId="GlassFishWebServer"
                    property="httpsport"/>
                <InputParameter nodeTemplateId="GlassFishWebServer"
                    property="username"/>
                <InputParameter nodeTemplateId="GlassFishWebServer"
                    property="password"/>
            </InputParametars>
      </ScriptArtifactProperties>
    </Properties>
    <ArtifactReferences>
      <ArtifactReference reference="scripts/GlassFishWebServer">
        <Include pattern="configure.sh"/>
      </ArtifactReference>
    </ArtifactReferences>
</ArtifactTemplate>
```

Код 5.11 Дефиниција на артефакт за конфигурација на GlassFish апликацискиот сервер

Покрај дефинициите, архивата вклучува и артефакт на апликацијата (war датотека), Java EE 7 скрипта од Oracle (која значително ја зголемува големината на архивата), скриптирачки артефакти за конфигурација на GlassFish серверот и инсталациски и конфигурирачки скрипти.

5.4 Дискусија

5.4.1 Ограничувања на домен на примена

Платформата базирана на проширената спецификација на TOSCA може да се примени за доста архитектурни стилови од различни катего-

рии, вклучувајќи: клиент-сервер архитектура, компонентно-базирана архитектура, слоевита архитектура итн... За жал постојат ограничувања ко не можат да бидат надминати, па така следните типови на апликации не можат да се моделираат:

- Апликации развиени во специфична PaaS околина кои користат уникатни затворени библиотеки за кои не постои соодветна замена
- Апликации кои побаруваат специфични библиотеки или драјвери кои не можат да се вклучат како артефакти
- Апликации или дополнителни компоненти кои не може да се инсталираат или конфигурираат со употреба на скрипти
- Апликации кои употребуваат компоненти чиј договор за лиценца или поддршка не дозволува виртуелизација

Дополнително предложениот модел не поддржува миграција и портабилност на податочни складишта.

Мора да се нагласи дека апликациите може да ги користат само достапните ресурси за типови на оперативни системи, па доколку одредена апликација побарува оперативен систем кој не е достапен, истата не може да се мигрира.

5.5 Главен придонес

Во оваа глава е прикажана примена на дефинираната TOSCA базирана платформа за различни типови на архитектури на апликации и со тоа се потврдува нејзината употребливост.

Глава 6
Евалуација на P-TOSCA моделот

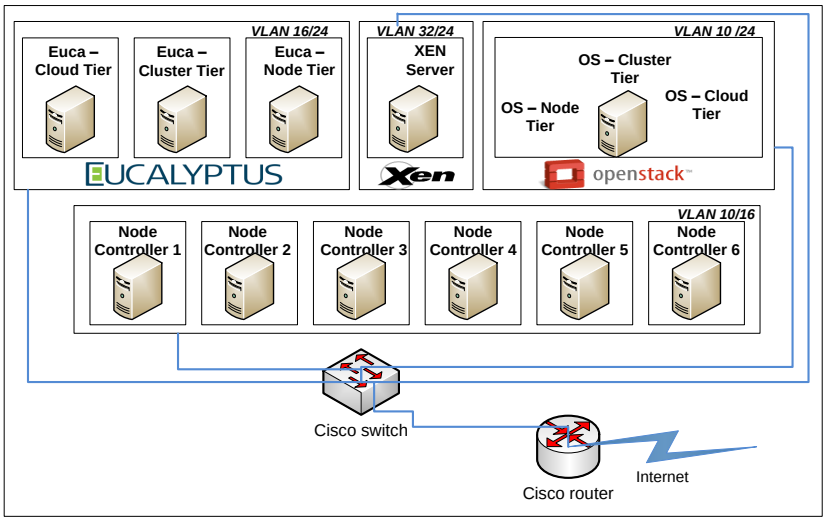
Преглед Со цел да се евалуира вредноста на P-TOSCA модел во однос на заштеда на време извршени се тестирања и споредби на времиња потребни за миграција и пренесување на апликација на облак. Притоа се разгледувани различни сценарија кои вклучуваат мануелна миграција и пренесување и миграција и пренесување со употреба на моделот. Сите тестирања се извршени во тестната околина поставена од LiiT проектот.

6.1 Вовед

Во оваа глава е даден преглед на резултати од тестирања извршени со цел а да се евалуира вредноста за користење на на P-TOSCA модел во однос на заштеда на време. Најпрво се дава преглед на користена методологија за тестирање, а потоа резултатите од тестирањето спроведена на две платформи за облак: OpenStack и Eucalyptus, поставени во лабораторијата на LiiT проектот.

6.2 Методологија

Методологијата за тестирање на P-TOSCA моделот е дефинирана со карактеристиките на тестната околина, тест случаите (сценаријата) кои се разгледуваат, како и тестните податоци со кои се врши тестирањето. Секој експеримент е извршен пет пати и се евалуира просечното време потребно за извршување на експериментот.



Слика 6.1 Архитектура на тестна околина

6.2.1 Преглед на околина за тестирање

Тестната околина е развиена како дел од LiiT проектот [80] и вклучува две платформи за облак: OpenStack и Eucalyptus. Опремата за двете платформи содржи 11 работни станици, 1 Cisco свич, и 1 Cisco рутер. Архтектурата е прикажана на Слика. 6.1.

Eucalyptus употребува три работни станици (по една за секоја Eucalyptus компонента) и истите се инсталирани со CentOS 6.5, додека пак сите компоненти на OpenStack се поставени на една работна станица инсталирана со Ubuntu Server 12.04.3 LTS. Мрежата е конфигурирана во 4 VLAN-ви. VLAN-те се меѓусебно поврзани со Cisco преклопник (switch), кој пак има линк до Cisco рутер и кој претставува порта (gateway). Структурата овозможува флексибилност на начин што последниот ред на јазли-контролори (кој претставува node controllers group) може да се употреби и од страна на OpenStack и од Eucalyptus во зависност од потребите за скалирање на перформансите.

6.2.2 Тест случаи (сценарија)

Генерирани се повеќе сценарија вклучувајќи и споредба на перформанси меѓу:

- *Тест случај 1*: Мануелна миграција,
- *Тест случај 2*: Миграција со користење на моделот меѓу исти околини на облак и
- *Тест случај 3*: Миграција со користење на моделот меѓу различни околини на облак.

6.2.2.1 Тест случај 1: Мануелна миграција

За да се евалуира времето потребно за стандардна мануелна миграција се разгледува времето потребно за корисникот да креира инстанци и да постави безбедносни правила за нив. Во зависност од големината на интеракцијата, дополнително се разгледуваат три различни случаи:

- *Тест случај 1а*: Користење на конзола или оддалечен пристап без користење на помошни скрипти
- *Тест случај 1б*: Користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација
- *Тест случај 1в*: Користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација и пренасочување на излезот од конзола во датотеки

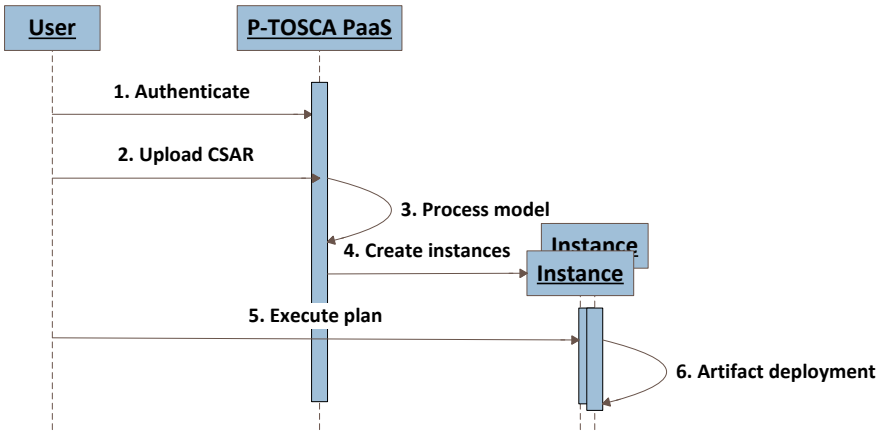
Во Тест случајот 1в се користи пренасочување на излезот од конзола за заштеда на време потребно за достава на излезот од конзолата до корисникот

6.2.2.2 Тест случај 2: Иницијална миграција во облак со користење на моделот

Ова сценарио претпоставува дека апликацијата за прв пат се мигрира на облак со користење на инстанца од платформата која го користи моделот. Притоа во вкупното време за миграција се состои од следните активности: достава на однапред изговена CSAR архива до платформата од страна на корисникот (T_upload), време потребно за креирање на инстанците од страна на платформата ($T\_VMcreate$) и време потребно за извршување на зададени планови (T_plans). Чекорите кои се извршуваат при овој тест случај се прикажани на Слика. 6.2.

6.2.2.3 Тест случај 3: Миграција со користење на моделот меѓу различни околини на облак

Ова сценарио претпоставува дека апликацијата е веќе поставена во една околина на облак со користење на инстанца од платформата која го користи моделот во дадениот облак. Во сценарио корисникот користи

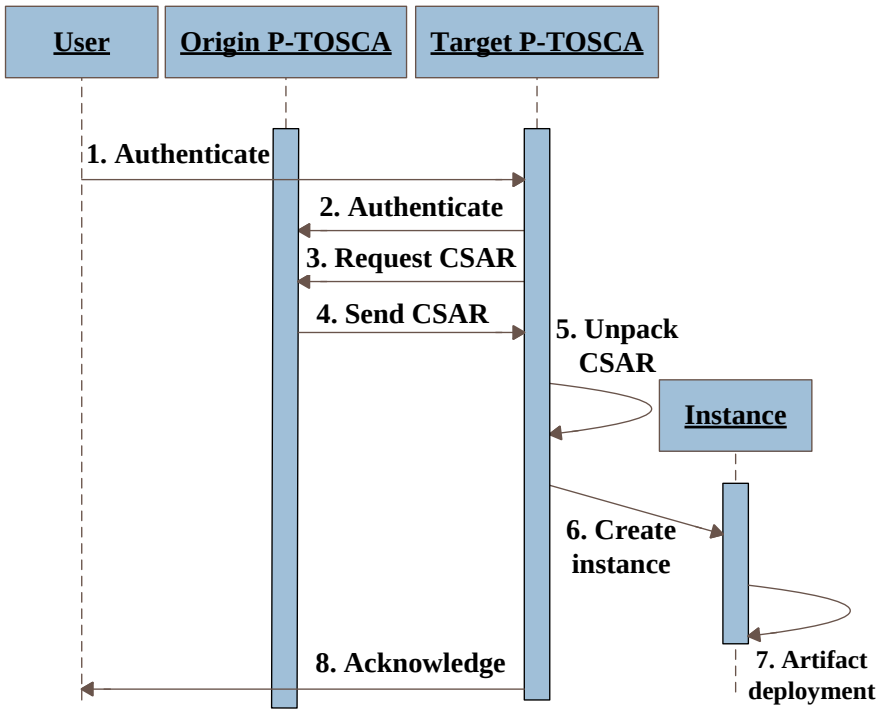


Слика 6.2 Чекори за извршување на тест случај 2

инстанца од платформата за друга околина и притоа наведува дека сака да ја мигрира апликацијата и ја дава локацијата на инстанцата каде е веќе поставена апликацијата. Инстанцата на новата околина комуницира со инстанцата од старата околина со користење на веб сервиси. Притоа во вкупното време за миграција се состои од следните активности: време потребни за превземање на топологијата на апликацијата, време потребно за креирање на нови инстанците од страна на платформата и време потребно за превземање и извршување на зададени планови. Чекорите кои се извршуваат при овој тест случај се прикажани на Слика. 6.3. За овој тест случај понатаму ќе се разгледуваат две можни сценарија: миграција на апликација од OpenStack на Eucalyptus и обратно.

6.2.3 Тест податоци

За спроведување на евалуацијата е избрана Java MVC апликацијата за управување со студентски досиеја. Оваа апликација е избрана бидејќи нејзината архитектура и примена е повеќеслојна и дистрибуирана, т.е. побарува две инстанци од кои едната се користи за апликациски сервер и поставување на апликацијата, а другата како податочен сервер каде е поставена базата со податоци.



Слика 6.3 Чекори за извршување на тест случај 2

6.3 Резултати од тест случаите за целна платформа OpenStack

Во овој дел се евалуираат трите тестни случаи каде целната платформа на мигрирање и пренесување е OpenStack.

6.3.1 Резултати од Тест случај1: Мануелна миграција на OpenStack

При тестирањето се извршени повеќе обиди за секое од мануелните сценарија и потребното време за секое од нив соодветно е прикажано во Табела. 6.1, Табела. 6.2 и Табела. 6.3.

Просечно време за секое од сценаријата е:

- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап без користење на помошни скрипти: 23 мин 31 сек

Табела 6.1 Резултати од Тест случај 1а: Мануелна миграција на OpenStack

Сценарио	Бр. обид	Вкупно време
Тест случај 1а на OpenStack	1	1572 сек
Тест случај 1а на OpenStack	2	1594 сек
Тест случај 1а на OpenStack	3	1352 сек
Тест случај 1а на OpenStack	4	1427 сек
Тест случај 1а на OpenStack	5	1413 сек

Табела 6.2 Резултати од Тест случај 1б: Мануелна миграција на OpenStack

Сценарио	Бр. обид	Вкупно време
Тест случај 1б на OpenStack	1	1381 сек
Тест случај 1б на OpenStack	2	1215 сек
Тест случај 1б на OpenStack	3	1166 сек
Тест случај 1б на OpenStack	4	1097 сек
Тест случај 1б на OpenStack	5	1154 сек

Табела 6.3 Резултати од Тест случај 1в: Мануелна миграција на OpenStack

Сценарио	Бр. обид	Вкупно време
Тест случај 1в на OpenStack	1	1092 сек
Тест случај 1в на OpenStack	2	1125 сек
Тест случај 1в на OpenStack	3	983 сек
Тест случај 1в на OpenStack	4	864 сек
Тест случај 1в на OpenStack	5	837 сек

- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација: 20 мин 2 сек
- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација како и пренасочување на излезот од конзола во датотеки за заштеда на време потребно за достава на излезот од конзолата до корисникот: 16 мин 20 сек

6.3.2 Резултати за Тест случај 2: Иницијална миграција со користење на моделот на OpenStack

При тестирањето се извршени повеќе обиди и потребното време за секој од нив е прикажано во Табела 6.4.

Просечно време за ова сценарио е: 4 мин 52 сек

Табела 6.4 Резултати од Тест случај 2: Иницијална миграција со користење на моделот на OpenStack

Сценарио	**Бр. обид**	**T_upload**	**T_VMcreate**	**T_plans**	**Вкупно време**
Тест случај 2 на OpenStack	1	1 сек	18.5 сек	282 сек	302 сек
Тест случај 2 на OpenStack	2	2 сек	18 сек	319 сек	339 сек
Тест случај 2 на OpenStack	3	1 сек	18.8 сек	257 сек	277 сек
Тест случај 2 на OpenStack	4	1 сек	18.9 сек	259 сек	279 сек
Тест случај 2 на OpenStack	5	1 сек	17.5 сек	245 сек	264 сек

6.3.3 Резултати за Тест случај 3: Миграција со користење на моделот од Eucalyptus на OpenStack

При тестирањето се извршени повеќе обиди и потребното време за секој од нив е прикажано во Табела 6.5.

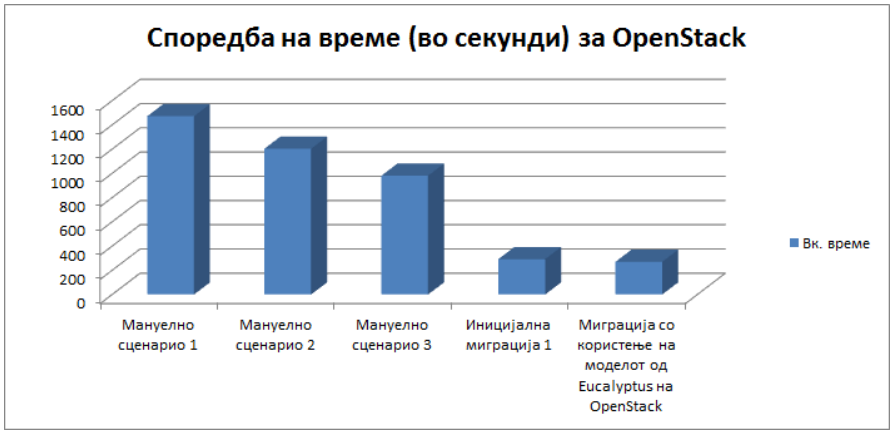
Табела 6.5 Резултати за Тест случај 3: Миграција со користење на моделот од Eucalyptus на OpenStack

Сценарио	**Вкупно време**
Тест случај 3: од Eucalyptus на OpenStack	274 сек
Тест случај 3: од Eucalyptus на OpenStack	315 сек
Тест случај 3: од Eucalyptus на OpenStack	257 сек
Тест случај 3: од Eucalyptus на OpenStack	228 сек
Тест случај 3: од Eucalyptus на OpenStack	272 сек

Просечно време за ова сценарио е: 4 мин 8 сек

6.3.4 Споредба на резултати за OpenStack

На Слика. 6.4 е прикажан столбестдијаргам кој дава преглед на вкупните времиња потребни за извршување на сценаријата на OpenStack облак. Може да се забележи дека употребата на платформата најмалку двократно го намалува потребното време за поставување на апликација во облак.



Слика 6.4 Споредба на времето (во секунди) потребно за извршување на сценаријата на OpenStack

6.4 Резултати од тест случаите за целна платформа Eucalyptus

Во овој дел се евалуираат трите тестни случаи каде целната платформа на мигрирање и пренесување е Eucalyptus.

6.4.1 Резултати од Тест случај 1: Мануелна миграција на Eucalyptus

При тестирањето се извршени повеќе обиди за секое од мануелните сценарија и потребното време за секое од нив соодветно е прикажано во Табела. 6.6, Табела. 6.7 и Табела. 6.8.

Табела 6.6 Резултати од Тест случај 1а: Мануелна миграција на Eucalyptus

Сценарио	Бр. обид	Вкупно време
Тест случај 1а на Eucalyptus	1	1386 сек
Тест случај 1а на Eucalyptus	2	1483 сек
Тест случај 1а на Eucalyptus	3	1513 сек
Тест случај 1а на Eucalyptus	4	1364 сек
Тест случај 1а на Eucalyptus	5	1290 сек

Просечно време за секое од сценаријата е:

Табела 6.7 Резултати од Тест случај 1б: Мануелна миграција на Eucalyptus

Сценарио	Бр. обид	Вкупно време
Тест случај 1б на Eucalyptus	1	1351 сек
Тест случај 1б на Eucalyptus	2	1223 сек
Тест случај 1б на Eucalyptus	3	1061 сек
Тест случај 1б на Eucalyptus	4	1013 сек
Тест случај 1б на Eucalyptus	5	1034 сек

Табела 6.8 Резултати од Тест случај 1в: Мануелна миграција на Eucalyptus

Сценарио	Бр. обид	Вкупно време
Тест случај 1в на Eucalyptus	1	895 сек
Тест случај 1в на Eucalyptus	2	915 сек
Тест случај 1в на Eucalyptus	3	872 сек
Тест случај 1в на Eucalyptus	4	683 сек
Тест случај 1в на Eucalyptus	5	803 сек

- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап без користење на помошни скрипти: 23 мин 27 сек
- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација: 18 мин 56 сек
- Мануелна инсталација на инстанците со користење на конзола или оддалечен пристап со користење на помошни скрипти за делумна автоматизација како и пренасочување на излезот од конзола во датотеки за заштеда на време потребно за достава на излезот од конзолата до корисникот: 13 мин 53 сек

6.4.2 Резултати од Тест случај 2: Иницијална миграција со користење на моделот на Eucalyptus

При тестирањето се извршени повеќе обиди и потребното време за секој од нив е прикажано во Табела 6.9.

Просечно време за ова сценарио е: 4 мин 8 сек

Табела 6.9 Резултати од Тест случај 2: Иницијална миграција со користење на моделот на Eucalyptus

Сценарио	**Бр. обид**	**T_upload**	**T_VMcreate**	**T_plans**	**Вкупно време**
Тест случај 2 на Eucalyptus	1	1 сек	7.4 сек	220 сек	228 сек
Тест случај 2 на Eucalyptus	2	1 сек	7.2 сек	214 сек	222 сек
Тест случај 2 на Eucalyptus	3	3 сек	6.7 сек	298 сек	308 сек
Тест случај 2 на Eucalyptus	4	1 сек	7 сек	223 сек	231 сек
Тест случај 2 на Eucalyptus	5	1 сек	6.7 сек	243 сек	251 сек

6.4.3 Резултати од Тест случај 3: Миграција со користење на моделот од OpenStack на Eucalyptus

При тестирањето се извршени повеќе обиди и потребното време за секој од нив е прикажано во Табела 6.10.

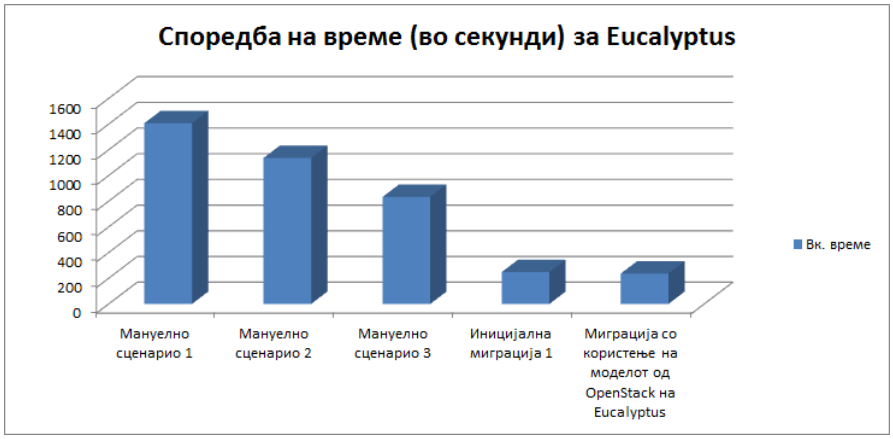
Табела 6.10 Резултати од миграција со користење на моделот од OpenStack на Eucalyptus

Сценарио	**Вкупно време**
Тест случај 3: од OpenStack на Eucalyptus	212 сек
Тест случај 3: од OpenStack на Eucalyptus	309 сек
Тест случај 3: од OpenStack на Eucalyptus	242 сек
Тест случај 3: од OpenStack на Eucalyptus	193 сек
Тест случај 3: од OpenStack на Eucalyptus	228 сек

Просечно време за ова сценарио е: 4 мин 8 сек

6.4.4 Споредба на резултати за Eucalyptus

На Слика. 6.5 е прикажан столбестдијаргам кој дава преглед на вкупните времиња потребни за извршување на сценаријата на Eucalyptus облак. Може да се забележи дека употребата на платформата најмалку двократно го намалува потребното време за поставување на апликација во облак.



Слика 6.5 Споредба на времето (во секунди) потребно за извршување на сценаријата на Eucalyptus

6.5 Изложеност на новите инстанците за време на креирање на топологија и спреведување на планови

Во процесот на креирање на инстанци и изврување на планови над нив платформата поставува, поточно дозволува TCP комуникација и тоа само на порта 22. Притоа за автентикација на инстанците се користат предефинирани парови од клучеви. По завршување на процесот се применуваат безбедносните правила дефинирани од корисникот, како и побараната автентикација.

6.6 Главен придонес

Со користење на имплементираната околина за P-TOSCA моделот забележани се значајни резултати за подобрување на времето за миграција и портација на една апликација во облак или од еден облак на друг. Така потребното време повеќекратно се намалува.

Глава 7
Заклучни согледувања

Преглед Во оваа глава се дадени заклучните согледувања и притоа се дава анализа на резултатите, нивната применливост, како и идна работа.

7.1 Вовед

Во оваа глава најпрво се дава анализа на секој од постигнатите резултати, потоа се дискутира нивната применливост и на крај се презентираат плановите за идна работа.

7.2 Анализа на резултатите

Со оваа докторска работа се добиени повеќе резултати во однос на евалуација на модели за стандарди за интероперабилност на Пресметување во облак, моделирање на интероперабилност на Софтвер како сервис со користење на адаптери, моделирање, дефинирање на портабилност и проширување на TOSCA моделот за апликации во облак, креирање на околина за примена на новиот P-TOSCA модел и евалуација од примена на моделот.

Евалуација на предложени модели за интероперабилност на Пресметување во облак

Како дел од истражувањето за оваа докторска дисертација, а опишани во Глава.2, се испитувани предлог моделите за овозможување на интероперабилност на Пресметување во облак [73][65, 68]. За таа цел е извршен

преглед и анализа на моделите: Унифициран интерфејс за облак / Брокер за облак, Платформа за оркестрација на облак / Оркестрациски слој, Нацрт за Интероблак, Интерфејс за управување со инфраструктура на облак и Отворен интерфејс за пресметување во облак. Покрај овие модели како предлози за стандандарди дополнително се анализирани и други предлог стандарди: Интерфејс за управување со податоци во облак, IEEE P2301 (Профили за облак) и IEEE P2302 (Интероблак) [63].

Пристапот кој е употребен за оваа евалуација е базиран на претходни искуства за евалуација на модели на интероперабилност и стандарди [69, 103, 104, 52, 55, 109]. Рамката за евалуација се базира на т.н. индикатори, со кои се доделуваат вредности за дадени карактеристики на предложжените модели/стандарди. Во оваа рамка се разгледуваат следните каракатеристики: сервисен модел за кој се однесува моделот/стандардот, статусот на документацијата, статусот на реализацијата и поддршката на предлог моделот/стандардот (поточно дали познати понудувачи на услуги во облак се вклучуваат во работата или ја поддржуваат изработката на дадениот модел/стандард). Секој од индикаторите е со вредност од 1 до 5. Секој предлог модел/стандард се евалуира по сите индикатори и финалната оцена се формира како сума од сите индикатори.

По евалуацијата на секој стандард посебно заклучено е дека стандардот Отворен интерфејс за пресметување во облак има најдобри оцени, а воедно веќе има свои имплементации за дел од платформите за облак. Според оценките овој стандард најверојатно ќе биде прифатен од повеќето платформи и понудувачи. И стандардот Интерфејс за управување со податоци во облак има добри оцени, но му недостасуваат повеќе имплементации за различни платформи на облак. На стандардот Интерфејс за управување со инфраструктура на облак му недостасува имплементација. На предлог моделите и стандарди Брокер за облак и Оркестрациски слој им недостасува поддршка. Стандардите IEEE P2301 (Профили за облак) и IEEE P2302 (Интероблак) сеуште се во развој и немаат ни предлог документација.

Моделирање на интероперабилност на Софтвер како сервис со користење на адаптери

Кога е потребно да се овозможи соработка и интероперабилност меѓу повеќе сервиси често се случува интеракциите меѓу сервисите да не се компатибилни, поточно да имаат различни интерфејси (signature incompatibilities) или пак различна логика за комуникација (protocol incompatibilities). Овие некомпатибилности не дозволуваат семантичка интероперабилност меѓу сервисите. Еден често користен пристап за справување со вакви ситуации е употребата на адаптери кои ќе вршат конверзија на барања во соодветен побаруван формат. За таа цел е извршена анализа

на типовите на програмирачки интерфејси на апликации кои најчесто се нудат од производителите на софтверите и разгледана е употребата на стандардниот Адаптер шаблон, како и изменета верзија на шаблонот Скалабилен адаптер. Адаптер шаблонот се користи за креирање на едноставни посредници меѓу услуги од ист тип, додека изменетата верзија на Скалабилен адаптер шаблонот се користи кога треба да се овозможи посредување и комуникација на сервиси кои што имаат делумно преклопување на функционалностите и типовите на податоци.

Со оглед на тоа дека моделирање на интероперабилност на Софтвер како сервис е возможно даден домен, креирани се повеќе предлози и решенија со користење на софтверски адаптери за да се овозможи соработка и размена на податоци.

Првото решение е за користење на едноставни софтверски адаптери за креирање интероперабилност на календари овозможени од повеќе различни популарни понудувачи на услуга: Google Calendar, Microsoft Live и Facebook. Притоа се анализирани различни типови на стандарди кои се користени од овие понудувачи како и нивните програмирачки интерфејси на апликации. Генерирано е решение кое нуди интеграција и размена на податоци од дадените типови на календари и со тоа е овозможена интероперабилност во овој домен. Самото решение е реализирано како сервис кој комуницира преку SOAP протокол со корисникот, додека адаптерите кои ги користи овие решение комуницираат преку RESTful сервисите на услугите со размена на JSON пораки. Со менувањето на интерфејсите и на начинот на размена на податоци, и софтверскиот адаптер е соодветно менуван за да се задржи функционалноста [74].

Второто решение е за користење на софтверски адаптери за креирање интероперабилност на услуги за складирање овозможени од повеќе различни популарни понудувачи на услуга: Dropbox, Google Drive и SkyDrive (OneDrive). Генерирано е решение кое претставува веб апликација со која корисникот комуницира. Апликацијата содржи едноставни адаптери кои комуницираат со услугите за складирање преку нивните REST сервиси и при оваа комуникација се врши размена на JSON пораки. На тој начин се овозможува на корисникот едноставен интерфејс за дадените услуги за користење и преку кој може да врши измена, бришење и миграција на датотеки од една услуга на друга [62].

Третата анализа е предлог за овозможување интероперабилност меѓу различни типови на софтвер кои се користат од страна на универзитети. Извршена е анализа на постоечки нормативи и стандарди и разгледувана е нивната примена во различни сценарија. За разгледување се зема поставеноста на сервисите на Факултетот за информатички науки. Како заклучок се дадени предлог стандарди за употреба кои делумно ќе го надминат недостатокот на интеоперабилност. Така за системите за управување со универзитети е предложена употреба на SCORM и IMS Enterprise стандардите, за системите за управување со учење е предложена употреба на SCORM, QTI и IMS Enterprise стандардите и за систе-

мите за електронско тестирање е предложена употреба на SCORM и QTI стандардите. Со примена на наведените стандарди може да се овозможи размена на информации и соработка на наведените типови на системи [71, 53].

Моделирање и дефинирање на портабилност на апликации во облак

За моделирање на портабилност на апликации во облак се истражуваат можности за модели и различни пристапи кои вклучуваат употреба на стандарди, генерирање на помошни решенија со апстрактни јазици и користење на семантички технологии. За детална анализа е избран предлог моделот Тополошки и оркестрациски сервиси за апликации (Topology and Orchestration Services for Applications - TOSCA). Извршена е детална анализа на можностите за тополошко моделирање со котистење на дадените типови: NodeType, RelationshipType, ArtifactType, NodeTemplate, RelationshipTemplate и ArtifactTemplate [58] како и дадена е предлог рамка за платформа за користење на овој модел и опишани се генералните функционалности кои што би требало да се поддржат, како што е процесирање на CSAR архива, процесирање на TOSCA модел, експортирање на моделот и артефактите и мониторирање на работата на применетите модели [72].

За истражување на примената и можностите на моделот за TOSCA извршено е истражување за креирање на тополошки модел на iKnow апликацијата - систем за управување со Универзитет [60]. Системот е избран поради неговата комплексност и специфичен начин на примена. Креирана е опсежна тополошка спецификација на системот и дефиниран е начин на примена. Дополнително се креирани спецификации и за апликации кои користат двослоен модел со дистрибуирана примена, како и за апликации кои користат сервисно-ориентирана архитектура [59, 64, 67].

Проширување на TOSCA моделот за портабилност и креирање на околина за примена

При креирање на околина за употреба на TOSCA моделот воочени се некои битни недостатоци како што е недоволната дефиниција на својства на јазли во топологијата, недостаток од спефицикација на безбедносни политики и права на пристап. Воочена е недоволната дефинираност на планови со користење на BPMN и BPEL јазиците, а со тоа и недостаток на доволно информации за соодветно конфигурирање на топологијата, посебно бидејќи не се обезбедува механизам на предавање вредности меѓу

јазлите како на пр. IP адреса на јазол која се дефинира динамички во време на градење на топологијата од страна на моделот.

Од таа причина е креиран нов моделот P-TOSCA кој претставува проширување и менување на TOSCA со нови дефиниции:

- се специфицира употреба на *надворешен namespace* за ServerProperties
- се специфицира употреба на *initial number of instances* дете-елемент на ServerProperties елементот
- се проширува ServerProperties елементот на Node Template со *Server-IPAddress* елементот
- се специфицира употреба на *надворешен namespace* за ScriptArtifact-Properties
- се проширува Properties елементот на Artifact Template со *InputParameters* елемент
- се проширува Properties елементот ба Node Template со *ServerSecurity-Properties* елемент
- воведување на *XML дефиниран план*

Развиена е целосна архитектура за употреба на овој модел на повеќе различни системи на облак истата вклучува повеќе нивоа како интерфејсен модул, P-TOSCA централен модул и Податочен модул. Креирана е имплементација на архитектурата за OpenStack и Eucalyptus. Потоа експериментално е докажана примената на P-TOSCA моделот во реална средина со користење на проширени спецификации за различни типови на апликации и со тоа е докажана примената на проширениот модел во реална средина [70].

Евалуација на практичното решение за примена на моделот за портабилност

Спроведена е евалуација на развиената околина во однос на време и перформанси за да се утврди исплатливост на моделот. За таа цел генерирани се повеќе сценарија вклучувајќи и извршена е споредба на перформанси меѓу стандардната мануелна миграција, миграција со користење на моделот од сопствена околина во околина на облак и миграција со користење на моделот меѓу различни околини на облак. За секој од тестовите се изведени повеќе обиди и се мерени времињата потребни за реализација на секое сценарио. Дополнително истиот процес е применет за две околини: OpenStack и Eucalyptus. За секое сценарио и околина се пресметува просечно потребно време за извршување и се прави споредба на секоја околина посебно. Не се врши заедничка споредба на резултатите од различни околини заедно поради нивните различни поставувања и различна хардверска опрема. Генералниот заклучок е дека користењето

на автоматизирана миграција и портирање е неколку кратко побрзо од мануелното [66].

Дополнително е извршена евалуација на безбедноста на примената на овој модел, како и на креираната околина тргнувајќи од резултатите добиени во [105].

7.3 Применливост на резултатите

Во овој дел се идентификува и објаснува применливоста на резултатите.

Автоматизирана миграција меѓу облаци

Со користење на P-TOSCA моделот и неговата имплементација за различни облаци [70] се овозможува автоматизирана миграција на апликациите поставени на еден облак во друг со користење на сервисите на платформата. За секоја апликација која корисникот сака да ја мигрира во облак креира CSAR архива со дадениот модел и ја извршува дадената топологија и процеси со користење на имплементацијата на платформата за дадениот облак. Доколку понатаму сака да ја мигрира својата апликација на друга околина, имплементацијата за моделот на новиот облак директно комуницира преку сервиси со стариот облак, ги превзема дефинициите и артефактите и ги применува во новата околина.

Независност во избор на платформа за облак

Главниот придонес на оваа дисертација се базира врз независноста на клиентот од изборот на облак за поставување на својата апликација поради можност за лесна миграција и промена на понудувачот на услуги во облак. Ова се постигнува со примена на P-TOSCA моделот, односно опишување на апликациите со овој модел и со користење на имплементацијата на платформа за овој модел во повеќе околини, па корисниот не се грижи за изборот на облак бидејќи со користење на моделот автоматски може да ја изгради потребната топологија во новата платформа. На тој начин се спречува заклучување на корисникот во еден облак.

Притоа земени се во предвид предизвиците и ризиците од безбедноста на предложената архитектура при нејзината примена во различни околини [106] како и поедноставување на можноста за воспоставување на систем за управување со безбедност на информации [102].

Намалување на времето за миграција и промена на облак

Со користење на P-TOSCA моделот[70] значително се намалува времето потребно за поставување на апликација во облак, како и времето потребно за промена на понудувач на услуги во облак. Откако еднаш ќе биде изградена спецификацијата на дадената апликација со користење на моделот, истата многу брзо се применува и се гради топологијата во даден облак. Исто така миграцијата при замена на еден облак со друг е многу побрза со користење на изградената спецификација и нејзина размена меѓу два облаци. Добиените експериментални резултати покажуваат повеќекратна заштеда на време, односно повеќето експерименти покажаа дека со користење на спецификацијата од моделот времето се намалува дури и повеќе од 4 пати. Оваа разлика во време значително се должи за времето потребно на корисникот мануелно да креира инстанци, да постави безбедносни правила и рачно да ги примени креираните артефакти [66].

Стандардизирано опишување на топологија на апликација и потребните операции и својства за облак

Со користење на P-TOSCA моделот [70] овозможуваме стандардизиран опис на дадена апликација во однос на нејзината топологија како и дефинирање на потребните операции и својства за примена на апликацијата во облак. Опис се генерира со користење на XML јазик, кој е доста често користен и разбирлив за корисниците, и дадени namespace-и. Со тоа самата спецификација е дадена во компјутерски-разбирлив јазик и по потреба лесно може да се преведе во друг побаруван формат. Дополнително спецификација јасно ги дефинира потребните својства на топологија и операциите кои треба да се извршуваат над нив и со тоа овозможува користење на сите погодности во облак како еластичност и скалабилност.

Градење на адаптери за интероперабилност на Софтвер како сервис

Со дефинираната методологија за генерирање на адаптери се овозможува генерирање на решенија за интероперабилност на Софтвер како сервис услуги од даден домен [62]. Така со користење на адаптер шаблонот нуди градење едноставни посредници меѓу услуги со слични функцио-

налности и податоци, додека изменетата верзија на Скалабилен адаптер шаблонот може лесно да се примени кога треба да се овозможи посредување и комуникација на сервиси кои што имаат делумно преклопување на функционалностите и типовите на податоци.

7.4 Идна работа

Примена на P-TOSCA моделот во повеќе реални апликации

Со примена на P-TOSCA моделот е предвидено дефинирање на спецификација за скалабилното и еластично e-Assessment решение [101]. Целта е полесно изведување на масивни, скалабилни и еластични курсеви како Архитектура и организација на компјутери, посебно поради задоволството на студентите за користење на електронски алатки [3]. Моделот ќе биде применет и за други апликации кои се реализирани како производ на истражувања [1, 61, 54]

Креирање имплементации и нивно тестирање за други платформи на облак

Предвидени се креирање на повеќе имплементации на архитектурата за да се овозможи функционирање и на други платформи покрај OpenStack и Eucalyptus. Така во наредна фаза се планирани имплементации и нивно тестирање за Amazon (со користење на веќе готовата имплементација за Eucalyptus, со оглед на тоа дека нудат ист интерфејс) и OpenNebula.

Воведување на сервиси за мониторирање на перформансите

Во архитектурата која е изградена како резултат на оваа докторска работа се планира да се додадат сервиси за мониторирање на перформансите на апликациите кои се поставени во облак со нејзина помош, и на тој начин да се овозможи на корисникот полесно да идентификува кога треба да се изврши склалирање или менување на ресурсите кои ги користи.

Можност за додавање на нови дефиниции

Планирано е проширување на архитектурата која е изградена за да овозможи додавање на нови дефиниции за топологијата на веќе применета апликација или менување на дел од постоечките дефиниции. Со тоа би се олеснило понатамошното одржување на апликациите, кои во својот животен век подлежат на менување и растење.

Користење на BPMN и BPEL јазиците за дефиниција на планови

Планирано е понатамошно проширување на моделот со опција за користење на планови дефинирани во BPMN и BPEL јазиците. Ова проширување би се спровело доколку се понуди попрецизна дефиниција за користење на BPMN и BPEL јазиците за креирање на планови од страна на TOSCA спецификацијата.

Референци

Литература

1. Ackovska, N., Kostoska, M.: Sign language tutor–rebuilding and optimizing. In: Proceedings of the 37th International Convention MIPRO 2014,. pp. 822–827 (2014), in press, IEEE conference proceedings
2. Armenski, G.: Designing Interoperable E-Assessment System. In: INTED2013 Proceedings. pp. 5137–5144. 7th International Technology, Education and Development Conference, IATED (2013)
3. Armenski, G., Kostoska, M., Ristov, S., Gusev, M.: Student satisfaction of e-learning tools for computer architecture and organization course. In: Global Engineering Education Conference (EDUCON), 2014 IEEE. pp. 630–637. IEEE (2014)
4. Badger, L., Bohn, R., Chandramouli, R., Grance, T., Karygiannis, T., Patt-Corner, R., Voas, J.: Cloud computing use cases (May 2011), http://www.nist.gov/itl/cloud/use-cases.cfm
5. Base, E.I.S.: Cloud interoperability (2011), http://www.fines-cluster.eu/fines/mw/index.php/Cloud_Interoperability
6. Bernstein, D., Ludvigson, E., Sankar, K., Diamond, S., Morrow, M.: Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In: Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on. pp. 328–336 (May 2009)
7. Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak, A., Wagner, S.: OpenTOSCA – a runtime for TOSCA-based cloud applications. In: 11^{th} International Conference on Service-Oriented Computing. LNCS, Springer (2013)
8. Breitenbücher, U., Binz, T., Kopp, O., Leymann, F., Schumm, D.: Vino4tosca: A visual notation for application topologies based on tosca. In: OTM 2012, Part I. Lecture Notes in Computer Science (LNCS), vol.

7565, pp. 416–424. Springer-Verlag (2012), http://dx.doi.org/10.1007/978-3-642-33606-5_25
9. Breiter, G., Behrendt, M., Gupta, M., Moser, S., Schulze, R., Sippli, I., Spatzier, T.: Software defined environments based on tosca in ibm cloud implementations. IBM Journal of Research and Development 58(2), 1–10 (March 2014)
10. Chorbev, I., Gusev, M., Gjorgjevikj, D., Madevska-Bogdanova, A.: Architecture of an electronic student services system and its implementation. In: Markovski, S., Gusev, M. (eds.) ICT Innovations 2012, Web proceedings. pp. 381 – 400 (2012)
11. Cordys: Now is the time to take the cloud seriouslyy (2011), http://www.cordys.com/cloud_orchestration
12. Cretella, G., Di Martino, B.: Towards automatic analysis of cloud vendors APIs for supporting cloud application portability. In: Sixth International Conference on Complex, Intelligent and Software Intensive Systems (CISIS). pp. 61 – 67 (2012)
13. Crow, D.: Apple's ical and ical/vcal format (2012), http://davidcrow.ca/article/641/apples-ical-and-icalvcal-format
14. Dawson, F.: Emerging calendaring and scheduling standards. Computer 30(12), 126–128 (Dec 1997)
15. DFTM: Open virtualization format specification version 1.1.0 (Jan 2010), http://www.dmtf.org/sites/default/files/standards/documents/DSP0243_1.1.0.pdf
16. Distributed Management Task Force: Dmtf to develop standards for managing a cloud computing environment (Apr 2009), http://www.dmtf.org/sites/default/files/DMTF_Cloud_Incubator_PR_FIN.pdf
17. Distributed Management Task Force: Interoperable clouds (Nov 2009), http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0101_1.0.0.pdf
18. Distributed Management Task Force: Architecture for managing clouds (Oct 2010), http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0102_1.0.0.pdf
19. Dropbox: (2014), https://www.dropbox.com/
20. Dropbox: (2014), https://www.dropbox.com/developers/core/docs
21. Durairajan, S., Sundararajan, P.: Portable service management deployment over cloud platforms to support production workloads. In: Cloud Computing in Emerging Markets (CCEM), 2013 IEEE International Conference on. pp. 1–7 (Oct 2013)
22. Dusseault, L., Whitehead, J.: Open calendar sharing and scheduling with caldav. Internet Computing, IEEE 9(2), 81–89 (Mar-Apr 2005)
23. eBay: (2014), http://developer.ebay.com/developercenter/soap/
24. Elgedawy, I.: On-demand conversation customization for services in large smart environments. IBM Journal of Research and Development 55(1.2), 5:1–5:14 (Jan 2011)

25. Fielding, R.: Architectural Styles and the Design of Network-based Software ArchitecturesArchitectural Styles and the Design of Network-based Software Architectures. Ph.D. thesis, University of California (2000), http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm
26. Force, D.M.T.: Use cases and interactions for managing clouds (Jun 2010), http://www.dmtf.org/sites/default/files/standards/documents/DSP-IS0103_1.0.0.pdf
27. Force, I.E.T.: Http extensions for distributed authoring – webdav (Feb 1999), http://tools.ietf.org/html/rfc2518
28. Force, I.E.T.: Calendaring extensions to webdav (caldav) (Mar 2007), http://tools.ietf.org/html/rfc4791
29. Force, I.E.T.: Internet calendaring and scheduling core object specification (icalendar) (Sept 2009), http://tools.ietf.org/html/rfc5545
30. Forum, C.C.I.: Unified cloud (2009), https://code.google.com/p/unifiedcloud/
31. Galan, F., Sampaio, A., Rodero-Merino, L., Loy, I., Gil, V., Vaquero, L.M.: Service specification in cloud environments based on extensions to open standards. In: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE. ACM New York (2009)
32. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional (1995)
33. Google: Import events from icalendar or csv files (2012), http://support.google.com/calendar/bin/answer.py?hl=en&answer=37118
34. Google: (2014), https://www.dropbox.com/
35. Google: (2014), https://developers.google.com/storage/docs/overview
36. Goyal, P.: Enterprise usability of cloud computing environments: Issues and challenges. In: 19th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises. pp. 54–59 (2010)
37. Gusev, M., Armenski, G.: E-Assessment Systems and Online Learning with Adaptive Testing. In: Ivanovic, M., Jain, L. (eds.) e-Learning Paradigms and Applications - Agent-based Approach, pp. 229–249. Springer Verlag (2013)
38. Hansen, M.D.: SOA Using Java Web Services. Pearson Education, Inc, Upper Saddle River, NJ (2007)
39. Harrer, A., Pinkwart, N., McLaren, B., Scheuer, O.: The scalable adapter design pattern: Enabling interoperability between educational software tools. Learning Technologies, IEEE Transactions on 1(2), 131–143 (April 2008)
40. Hill, Z., Humphrey, M.: CSAL: A cloud storage abstraction layer to enable portable cloud applications. In: IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom). pp. 504 – 511 (2010)

41. Hu, F., Qiu, M., Li, J., Grant, T., Tylor, D., McCaleb, S., Butler, L., Hamner, R.: A review on cloud computing: Design challenges in architecture and security. CIT 19(1), 25–55 (2011)
42. IBM: Ibm lotus notes 8.5 icalendar: Interoperability, implementation, and application (2012), http://www.ibm.com/developerworks/lotus/library/notes85-icalendar/
43. IEEE: Ieee standard glossary of computer networking terminology. IEEE Std 610.7-1995 pp. 1– (1995)
44. IEEE P2301 Working Group: IEEE P2301 (Cloud Profiles) (2013), http://grouper.ieee.org/groups/2301/
45. IEEE P2302 Working Group: IEEE P2302 (Intercloud) (2013), http://grouper.ieee.org/groups/2302/
46. iKnow: (2012), http://iknow.ii.edu.mk/
47. IMS Global Learning Consortium: Ims enterprise specification (2002), http://www.imsglobal.org/enterprise/
48. IMS Global Learning Consortium: Ims question and test interoperability specification (2012), http://www.imsglobal.org/question/
49. Internet Engineering Task Force (IETF): The websocket protocol (Dec 2011), http://tools.ietf.org/html/rfc6455
50. JISC: Effective Practice with e-Assessment (2007), http://www.jisc.ac.uk/media/documents/themes/elearning/effpraceassess.pdf
51. Jovanov, M., Gusev, M.: Performance evaluation of a new approach for automatic question production. In: Davcev, D., Gómez, J. (eds.) ICT Innovations 2009, pp. 275–283. Springer Berlin Heidelberg (2010)
52. Kiroski, K., Gusev, M., Kostoska, M.: A new methodology to benchmark sophistication of e-invoicing and e-ordering. In: ICT Innovations 2010, pp. 358–368. Springer (2011)
53. Kiroski, K., Gusev, M., Kostoska, M., Ristov, S.: Cloud e-university services. In: ICT Innovations 2011, Web Proceedings, Skopje, Macedonia. pp. 501–506 (2012)
54. Kiroski, K., Gusev, M., Kostoska, M., Ristov, S.: Modifications and improvements on cen/bii profiles. In: Kocarev, L. (ed.) ICT Innovations 2011, Advances in Intelligent and Soft Computing, vol. 150, pp. 395–404. Springer Berlin / Heidelberg (2012)
55. Kiroski, K., Kostoska, M., Gusev, M., Ristov, S.: Growth rate analysis of e-government development. In: Proceedings of the Fifth Balkan Conference in Informatics. pp. 106–111. BCI '12, ACM, Novi Sad, Serbia (2012), http://doi.acm.org/10.1145/2371316.2371337
56. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Bpmn4tosca: A domain-specific language to model management plans for composite applications. In: Mendling, J., Weidlich, M. (eds.) Business Process Model and Notation. Lecture Notes in Business Information Processing, vol. 125, pp. 38 – 52. Springer Berlin Heidelberg (2012)

57. Kopp, O., Binz, T., Breitenbücher, U., Leymann, F.: Winery – modeling tool for TOSCA-based cloud applications. In: 11th International Conference on Service-Oriented Computing. LNCS, Springer (2013)
58. Kostoska, M., Gusev, M., Ristov, S., Kiroski, K.: Cloud service portability opportunities. In: Proceedings of the 8th Annual South-East European Doctoral Student Conference. pp. 313–320 (2013), best paper award
59. Kostoska, M., Ackovska, N., Gjorgjievski, D., Skopje, M.: Ciit 2010 web application. In: Proceedings of the Seventh International Conference for Informatics and Information Technology CIIT 2010. pp. 137–141 (2010)
60. Kostoska, M., Chorbev, I., Gusev, M.: Creating portable tosca archive for iknow university management system. In: Proceedings of the Federated Conference on Computer Science and Information Systems 2014. IEEE (2014), to be published
61. Kostoska, M., Gusev, M.: Treasury information systems and architectures–possibilities and directions for further development in macedonia. In: Local Proceedings of ICT Innovations 2009 (2009)
62. Kostoska, M., Gusev, M.: Adapters as cloud interoperability. Tech. Rep. LiiT:47/13, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
63. Kostoska, M., Gusev, M.: Evaluation of possible cloud computing standards. In: Proceedings of the Tenth International Conference for Informatics and Information Technology CIIT 2013. pp. 261–263 (2013)
64. Kostoska, M., Gusev, M.: N-tier application cloud portability. Tech. Rep. LiiT23/14, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
65. Kostoska, M., Gusev, M.: Overview of cloud interoperability and portability trends. Tech. Rep. LiiT:45/13, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
66. Kostoska, M., Gusev, M.: Perrformance od application cloud poratbility. Tech. Rep. LiiT:25/14, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
67. Kostoska, M., Gusev, M.: Service oriented application cloud portability. Tech. Rep. LiiT:24/14, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
68. Kostoska, M., Gusev, M.: State of the art in cloud interoperability. Tech. Rep. LiiT:46/13, University Ss Cyril and Methodius, Skopje, Macedonia, Faculty of Information Sciences and Computer Engineering (2013)
69. Kostoska, M., Gusev, M., Kiroski, K.: Evaluation methodology for national enterprise architecture frameworks. In: Web proceedings of ICT Innovations 2010. pp. 129–138 (2010)

70. Kostoska, M., Gusev, M., Ristov, S.: Tosca based portability model for paas hosted applications. Future Generation Computer Systems (2014), submited for review
71. Kostoska, M., Gusev, M., Ristov, S.: Interoperability challenges among university electronic services systems. In: Proceedings of the 37th International Convention MIPRO 2014,. pp. 828–833 (2014), in press, IEEE conference proceedings
72. Kostoska, M., Gusev, M., Ristov, S.: A new cloud services portability platform. Procedia Engineering 69(0), 1268 – 1275 (2014), http://www.sciencedirect.com/science/article/pii/S1877705814003646, 24th {DAAAM} International Symposium on Intelligent Manufacturing and Automation, 2013
73. Kostoska, M., Gusev, M., Ristov, S., Kiroski, K.: Cloud computing interoperability approaches–possibilities and challenges. In: Local Proceedings of the Fifth Balkan Conference in Informatics. pp. 30–34. BCI '12, CEUR-WS.org, Novi Sad, Serbia (2012), http://ceur-ws.org/Vol-920/
74. Kostoska, M., Velkoski, G., Bozinoski, K., Ristov, S., Gusev, M.: Service agents for calendar exchange. In: Local Proceedings of the Fifth Balkan Conference in Informatics. pp. 142–146. BCI '12, CEUR-WS.org, Novi Sad, Serbia (2012), http://ceur-ws.org/Vol-920/
75. Kubicek, H., Cimander, R.: Three dimensions of organizational interoperability. European Journal of ePractice 6, 1–12 (2009)
76. Kubicek, H., Cimander, R., Scholl, H.J.: Organizational Interoperability in E-Government. Springer-Verlag, Berlin Heidelberg 2011 (2011)
77. Kumar, B., Cheng, J., Mcgibbney, L.: Cloud computing and its implications for construction it. In: Tizani, W. (ed.) Computing in Civil and Building Engineering, Proceedings of the International Conference. p. 315. Nottingham University Press (2010)
78. Lewis, G.: Role of standards in cloud-computing interoperability. In: 46th Hawaii International Conference on System Sciences (HICSS). pp. 1652 – 1661 (2013)
79. Li, F., Vogler, M., Claessens, M., Dustdar, S.: Towards automated iot application deployment by a cloud-based approach. In: Service-Oriented Computing and Applications (SOCA), 2013 IEEE 6th International Conference on. pp. 61–68 (Dec 2013)
80. LiiT: Laboratory for internet and innovative technologies (Jan 2014), http://liit.finki.ukim.mk/
81. Loutas, N., Kamateri, E., Tarabanis, K.: A semantic interoperability framework for cloud platform as a service. In: IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom). pp. 280 – 287 (2011)
82. Microsoft: View and subscribe to internet calendars (2012), http://office.microsoft.com/en-us/outlook-help/view-and-subscribe-to-internet-calendars-HA010167325.aspx

83. Microsoft: (2014), https://onedrive.live.com/
84. Microsoft: (2014), http://msdn.microsoft.com/en-us/library/dn631819.aspx
85. Moodle: (2013), http://moodle.org
86. Moscato, F., Aversa, R., Di Martino, B., Fortis, T.: An analysis of mOSAIC ontology for cloud resources annotation. In: Federated Conference on Computer Science and Information Systems (FedCSIS). pp. 973 – 980 (2011)
87. National Institute of Standards and Technology: The NIST Definition of Cloud Computing (Jyl 2012), http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
88. National Institute of Standards and Technology: Nist cloud computing standards roadmap (Jul 2013), http://www.nist.gov/itl/cloud/upload/NIST_SP-500-291_Version-2_2013_June18_FINAL.pdf
89. National Institute of Standards and Technology: Nist framework and roadmap for smart grid interoperability standards, release 1.0 (Jul 2013), http://www.nist.gov/public_affairs/releases/upload/smartgrid_interoperability_final.pdf
90. OASIS: Topology and orchestration specification for cloud applications (tosca) primer version 1.0 (Jan 2013), http://docs.oasis-open.org/tosca/tosca-primer/v1.0/tosca-primer-v1.0.html
91. OASIS: Topology and orchestration specification for cloud applications version 1.0. (Mar 2013), http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html
92. Oberle, K., Fisher, M.: Etsi cloud - initial standardization requirements for cloud services. In: GECON'10 Proceedings of the 7th international conference on Economics of grids, clouds, systems, and services. pp. 105–115. Springer-Verlag Berlin, Heidelberg (2010)
93. Open Grid Forum: Open cloud computing interface - core (Apr 2011), http://ogf.org/documents/GFD.183.pdf
94. Parameswaran, A., Chaddha, A.: Cloud interoperability and standardization. Tech. Rep. VOL 7 NO 7, SETLabs Briefings 2009 (2009)
95. Petcu, D.: Portability and interoperability between clouds: challenges and case study. In: ServiceWave'11 Proceedings of the 4th European conference on Towards a service-based internet. pp. 105–115. Springer-Verlag Berlin, Heidelberg (2011)
96. Petcu, D.: How to build a reliable mOSAIC of multiple cloud services. In: Proceedings of the 1st European Workshop on Dependable Cloud Computing (2012)
97. Petcu, D., Macariu, G., Panica, S., Crăciun, C.: Portable cloud applications—from theory to practice. Future Generation Computer Systems 29(6), 1417 – 1430 (2013), http://www.sciencedirect.com/science/article/pii/S0167739X12000210

98. Prodan, R., Ostermann, S.: A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In: Grid Computing, 2009 10th IEEE/ACM International Conference on. pp. 17–25 (Oct 2009)
99. Ranabahu, A., Maximilien, E., Sheth, A., Thirunarayan, K.: Application portability in cloud computing: An abstraction driven perspective. IEEE Transactions on Services Computing PP(99), 1 (2013)
100. Rimal, B., Choi, E., Lumb, I.: A taxonomy and survey of cloud computing systems. In: INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on. pp. 44–51 (Aug 2009)
101. Ristov, S., Gusev, M., Armenski, G., Velkoski, G.: Scalable and elastic e-assessment cloud solution. In: Global Engineering Education Conference (EDUCON), 2014 IEEE. pp. 762–769 (April 2014)
102. Ristov, S., Gusev, M., Kostoska, M.: Information security management system for cloud computing. In: ICT Innovations 2011, Web Proceedings, Skopje, Macedonia (2011)
103. Ristov, S., Gusev, M., Kostoska, M.: Cloud computing security in business information systems. International Journal of Network Security & Its Applications (IJNSA) 4(2), 75–93 (2012)
104. Ristov, S., Gusev, M., Kostoska, M.: A new methodology for security evaluation in cloud computing. In: MIPRO, 2012 Proc. of the 35th Int. Convention, IEEE Conference Publications. pp. 1808–1813 (2012)
105. Ristov, S., Gusev, M., Kostoska, M.: Security assessment of openstack open source cloud solution. In: Proceedings of the 7th South East European Doctoral Student Conference (DSC2012) (2012)
106. Ristov, S., Gusev, M., Kostoska, M., Kiroski, K.: Business continuity challenges in cloud computing. In: ICT Innovations 2011, Web Proceedings, Skopje, Macedonia (2011)
107. Ristov, S., Gusev, M., Kostoska, M., Kjiroski, K.: Virtualized environments in cloud can have superlinear speedup. In: ACM Proceedings of 5th Balkan Conference of Informatics (BCI2012) (2012)
108. SCORM: (2004), http://www.adlnet.org/scorm/
109. Sonntagbauer, P., Gusev, M., Rotim, S.T., Stefanovic, N., Kiroski, K., Kostoska, M.: e-government and e-business in western balkans 2010. In: ICT Innovations 2010, pp. 152–165. Springer Berlin Heidelberg (2011)
110. Storage Networking Industry Association: Cloud data management interface (CDMI) (Jun 2012), http://snia.org/sites/default/files/CDMI%20v1.0.2.pdf
111. The E-Learning Framework: (2004), http://www.elframework.org/
112. UKOLN: Looking at interoperability (Apr 2005), http://www.ukoln.ac.uk/interop-focus/about/flyer-interoperability.pdf
113. Union, E.: European interoperability framework for pan-european e-government services (2004), http://ec.europa.eu/idabc/servlets/Docd552.pdf?id=19529
114. W3C: Architecture of the world wide web (Dec 2003), http://www.w3.org/TR/2003/WD-webarch-20031209/

115. W3C: Web services architecture (Feb 2004), http://www.w3.org/TR/ws-arch/#whatis
116. W3C: The websocket api (Jun 2014), http://dev.w3.org/html5/websockets/
117. Zhang, Z., Wu, C., Cheung, D.W.: A survey on cloud interoperability: Taxonomies, standards, and practice. SIGMETRICS Perform. Eval. Rev. 40(4), 13–22 (Apr 2013), http://doi.acm.org/10.1145/2479942.2479945
118. Zhang, Z., Wu, C., Cheung, D.W.: A survey on cloud interoperability: taxonomies, standards, and practice. ACM SIGMETRICS Performance Evaluation Review 40(4), 13–22 (2013)

Индекс