UDC: 004.4:004.23 Info M: str. 30-38

ОПШТИ МОДЕЛ ПЛАТФОРМСКЕ НЕЗАВИСНОСТИ
GENERAL CONCEPT OF THE PLATFORM INDEPENDENCY MODEL

Синиша Влајић, Илија Антовић, Душан Савић, Марија Видаковић
Универзитет у Београду, Факултет организационих наука

РЕЗИМЕ: Предмет овог рада је концепт платформски независних софтверских архитектура, као и значај овог концепта у развоју пословних софтверских система, али и за софтверско инжењерство као науку. Овај рад представља начине и типове интерпретације овог концепта, као и алате и механизме за његову реализацију у реалном окружењу. У раду је представљен општи модел платформске независности (General Concept of the Independency (GCoI) model). Овај модел је изведен на основу најважнијих постојећих механизама платформске независности. У раду су идентификована четири кључна механизма за постизање платформске независности, а сва четири су описана коришћењем GCoI модела. GCoI модел представља основни концепт на који се ослања платформска независност. Java и .NET су данас најпопул арније платформе за развој пословних софтверских система. Обе платформе имају механизме који омогућавају развој софтверских система, а који су при томе независни у односу на оперативне системе и хардверске платформе на којима се извршавају. Упоредили смо ове платформе како бисмо открили на који начин и у којој мери свака од њих реализује концепт платформске независности – које су сличности и разлике у постизању овог циља. Пажња је посвећена и Моделом вођеној архитектури (Model Driven Architecture (MDA)), која је заснована на визији платформски независносне софтверске архитектуре, која се остварује коришћењем платформски независних модела.

КЉУЧНЕ РЕЧИ: Платформска независност, Општи модел платформске независности, механизми за остваривање платформске независности, Софтверско инжењерство, Платформски независне софтверске архитектуре.

ABSTRACT: The main topic of this paper is a platform independent software architecture concept and the importance of this concept for enterprise software systems development and the software engineering as a science. This paper presents the ways and types of interpretation of this concept, as well as the tools and mechanisms of its realization in real world environment. The paper introduces General Concept of the Independency (GCoI) model. The model is derived from most important platform independence mechanisms. The paper identifies four key mechanisms for achieving platform independence, and all of them are described using GCoI model. The GCoI model represents the fundamental concept that lies behind the platform independence. Java and .NET are the most powerful platforms for the enterprise software systems development today. They both have mechanisms well suited for realization of software applications, which are platform independent in respect to operating systems and hardware platforms. These two technologies are compared in order to find out how and to which extent each of them realizes this concept – what are similarities and differences in achieving this task. Attention is also given to MDA, because it is based on a vision of platform independent software architecture, which is achieved with platform independent models.

KEY WORDS: Platform independency, General Concept of the Independency (GCoI) model, mechanisms for achieving platform independence, software engineering, Platform independent software architecture.

1. УВОД

Појава нових софтверских архитектура (Liu et al. 2011; Bass, Clements and Kazman, 2003; Gorton, 2011) и софтверских платформи, нарочито у контексту приступа рачунарства у облаку (*Cloud Computing)*, намеће потребу и изазов за идентификацију и разумевање концепата и механизама који омогућавају интеграцију и прилагођавање у хетерогеним окруженима.

Основни циљ овог рада је да се идентификују механизми за остваривање независности између софтверских архитектура и софтверских платформи. Приликом развоја нове платформе или софтверске архитектуре, инжењери морају бити свесни свих потребних елемената који морају постојати како би се остварила платформска независност. У том смислу прво ћемо покушати да дефинишемо платформски независне архитектуре, као и да идентификујемо основне карактеристике које софтверска архитектура мора да поседује како би је сматрали платформски независном. Покушаћемо и да класификујемо типове платформске независности, као и да идентификујемо механизме који су коришћени за остваривање сваког типа платформске независности. На основу анализе идентификованих механизама биће дефинисан општи модел за остваривање платформске независности. За примере преко којих ће бити објашњен општи модел платформске независности биће коришћене платформски независне архитектуре као што су SOA (Vitvar et al. 2007; Erl, 2005), COA (Gorton, 2011) и MDA (Meghan; Object Management Group, 2003), као и најважније софтверске платформе Java и .NET.

У другом делу рада дефинисали смо и објаснили платформу и платформски независне софтверске архитектуре и указали на основне проблеме у развоју оваквих система. У трећем делу рада приказана је класификација различитих типова платформске независности. Ова класификација изведена је на основу уочених веза између софтверских архитектура и софтверских платформи. У следећем делу биће описана четири механизма која се користе за остваривање платформске независности. Сваки од ових механизама је објашњен коришћењем Општег модела платформске независности (General Concept of the Independency (GCoI) model). GcoI модел представља основни концепт на којем

се базира платформска независност, а у петом делу аутори истичу значај овог модела и представљају будуће правце истраживања.

2. ПЛАТФОРМСКА НЕЗАВИСНОСТ

У овом делу рада дефинисаћемо и објаснити концепте платформе и платформске независности, као и пратформски независне софтверске архитектуре. Такође ће бити речи о кључним концептима који се тичу платформске независности и који воде до њене реализације. Биће указано и на главне узроке проблема у имплементацији платформске независности.

2.1. *Дефиниција платформе*

Платформска независност и платформа као концепт се појављују у различитим областима софтверског инжењерства. Разматрање концепта платформске независности започиње разматрањем концепта платформи и њихове основне класификације.

Object Management Group OMG дефинише платформу као:

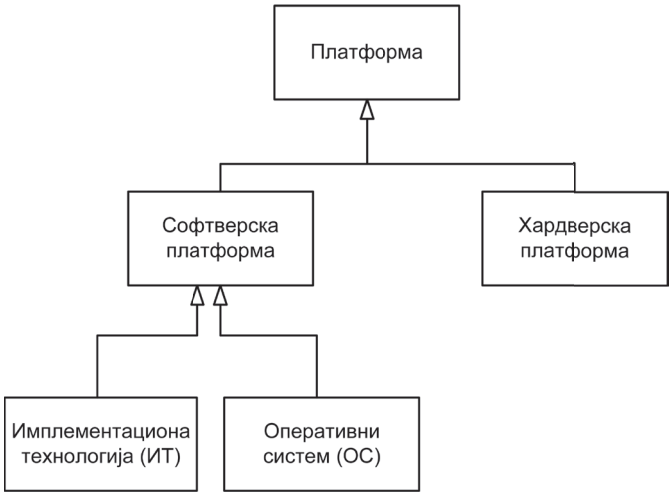
„Скуп подсистема/технологија које пружају кохерентан скуп функционалности кроз интерфејсе и специфициране обрасце коришћења које сваки подсистем који зависи од платформе може користити без обзира на детаље начина имплементације платформе која дату функционалност пружа.“ (Object Management Group, 2003)

Платформа се најчешће дефинише као комбинација хардвера и софтвера који се користе како би се извршила софтверска апликација. Платформа се једноставно може описати као оперативни систем или архитектура рачунара, или као њихова комбинација.

Постоје два основна типа платформе:

- Хардверска платформа може бити посматрана као архитектура рачунара или процесорска архитектура. На пример, x86 и x86-64 процесори чине једну од најчешћих процесорских архитектура рачунара опште намене.
- Софтверске платформе могу бити или оперативни системи (Microsoft Windows, Linux, Mac) или имплементационе технологије, или најчешће њихова комбинација. Java и .NET су софтверске платформе које представљају скуп технологија и програмских језика које заједно са специјализованим развојним окружењима омогућавају развој сложених софтверских система. Java и .NET представљају платформе које омогућавају да се апликације (развијене коришћењем тих технологија), извршавају на различитим оперативним системима и хардверским платформама.

Платформа представља подсистем или скуп подсистема који пружа одређену функционалност. Платформа представља основу за извршење других софтверских система и апликација. На Слици 1. представљена је основна класификација платформи.



Слика 1: *Основна класификација платформи*

2.2. *Платформски независне софтверске архитектуре: дефиниција и карактеристике*

У ужем смислу, софтверска архитектура је “опис подсистема и компонената које чине софтверски систем као и веза које међу њима постоје.” (Buschmann et al. 1996) У овом раду ће бити разматране Моделом вођена архитектура (Model Driven Architecture – MDA) (Meghan), Компонентно оријентисана архитектура (Component Oriented Architecture – COA) (Gorton, 2011) и Сервисно оријентисана архитектура (Service-Oriented Architecture – SOA) (Erl, 2005).

Платформски независна софтверска архитектура не укључује било какве специфичности имплементационих технологија, нити детаље везане за циљну хардверску платформу или оперативни систем на којем ће се апликација извршавати.

Платформски независна софтверска архитектура треба да задовољава следеће захтеве:

- Платформски независна софтверска архитектура не садржи детаље везане за хардверску платформу на којој се софтвер извршава.
- Платформски независна софтверска архитектура не садржи детаље везане за оперативни систем на којем се софтвер извршава.
- Платформски независна софтверска архитектура не садржи детаље везане за имплементациону технологију.
- Платформски независна софтверска архитектура је специфицирана универзално прихваћеним језицима за спецификацију софтверских система.
- Платформски независна софтверска архитектура може имати велики број имплементација.
- Платформски независна софтверска архитектура је заснована на принципу апстракције платформе.

Како су искључени детаљи везани за имплементацију и циљану платформу, учесници у процесу развоја се могу концентрисати на кључне и суштинске аспекте софтверске архитектуре и софтверског система. Поред тога, много је

лакше објаснити архитектуру људима који немају техничког знања, а који су заинтересовани или имају утицаја на развој или финансирање софтверског система.

Универзално прихваћене нотације и методе чине да платформски независне софтверске архитектуре унапређују и олакшавају разумевање софтверског система као и комуникацију у процесу развоја.

2.3. *Концепт платформске независности: циљеви и значај*

Концепт платформске независности повезан је са концептима преносивости (portability), поновног коришћења (reusability), универзалности и финансијске одрживости. Преносивост, вишеструко коришћење и финансијска одрживост се могу сматрати примарним циљевима приликом реализације платформски независне софтверске архитектуре.

Универзална и широко прихваћена правила не доживљавају брзе и честе промене, колико се оне дешавају у случају хардверских платформи, оперативних система и технологија. Универзално прихваћени принципи обично важе и ван граница времена и простора у којем се појављују. Платформски независне софтверске архитектуре теже да не застаревају брзо, из разлога што чувају знање и искуство које је акумулирано током много година и генерација, знање које може бити поново искоришћено у бројним ситуацијама. Такве архитектуре представљају снажан основ будућег развоја софтверских система.

На основу претходно изнесеног можемо закључити: Најважнији циљ реализације платформске независности је софтверска апликација која може бити извршена на више оперативних система и хардверских платформи без потребе за изменама или прилагођавањем. Таква решења пружају највећи ниво преносивости.

У односу на софтверске платформе, најважнији циљеви реализације платформске независности су:

- Да омогуће да се иста софтверска архитектура може искористити у различитим имплементацијама коришћењем различитих технологија.
- Заштита резултата процеса пројектовања софтвера од опасности брзог застаревања које може бити проузроковано променама у технологији, оперативном систему или хардверској платформи, тј. платформи било које врсте.

3. КЛАСИФИКАЦИЈА ПЛАТФОРМСКЕ НЕЗАВИСНОСТИ

Када се говори о платформској независности, неопходно је навести тип платформске независности о којој се говори, тј. на шта се тачно независност односи. Зато ће у раду бити коришћени термини ко/шта је независно у односу на кога/шта.

Независност може бити постигнута у односу на:

- Софтверску платформу, или
- Хардверску платформу,

а међу софтверским платформама разликујемо:

- Оперативне системе, и
- Имплементационе технологије

Платформу можемо посматрати као софтверску или хардверску. Правимо разлику између имплементационих технологија као платформи (као што су Java и .Net) и оперативних система као платформи. Имплементационе технологије се извршавају на оперативним системима, док се оперативни системи извршавају на неким хардверским платформама. Софтверки систем је заснован на софтверској архитектури, имплементиран је коришћењем имплементационе технологије и извршава се на неком оперативном систему.

На основу претходно поменутих веза између софтверске архитектуре и софтверске платформе, у наставку ће бити изведена основна класификација платформске независности:

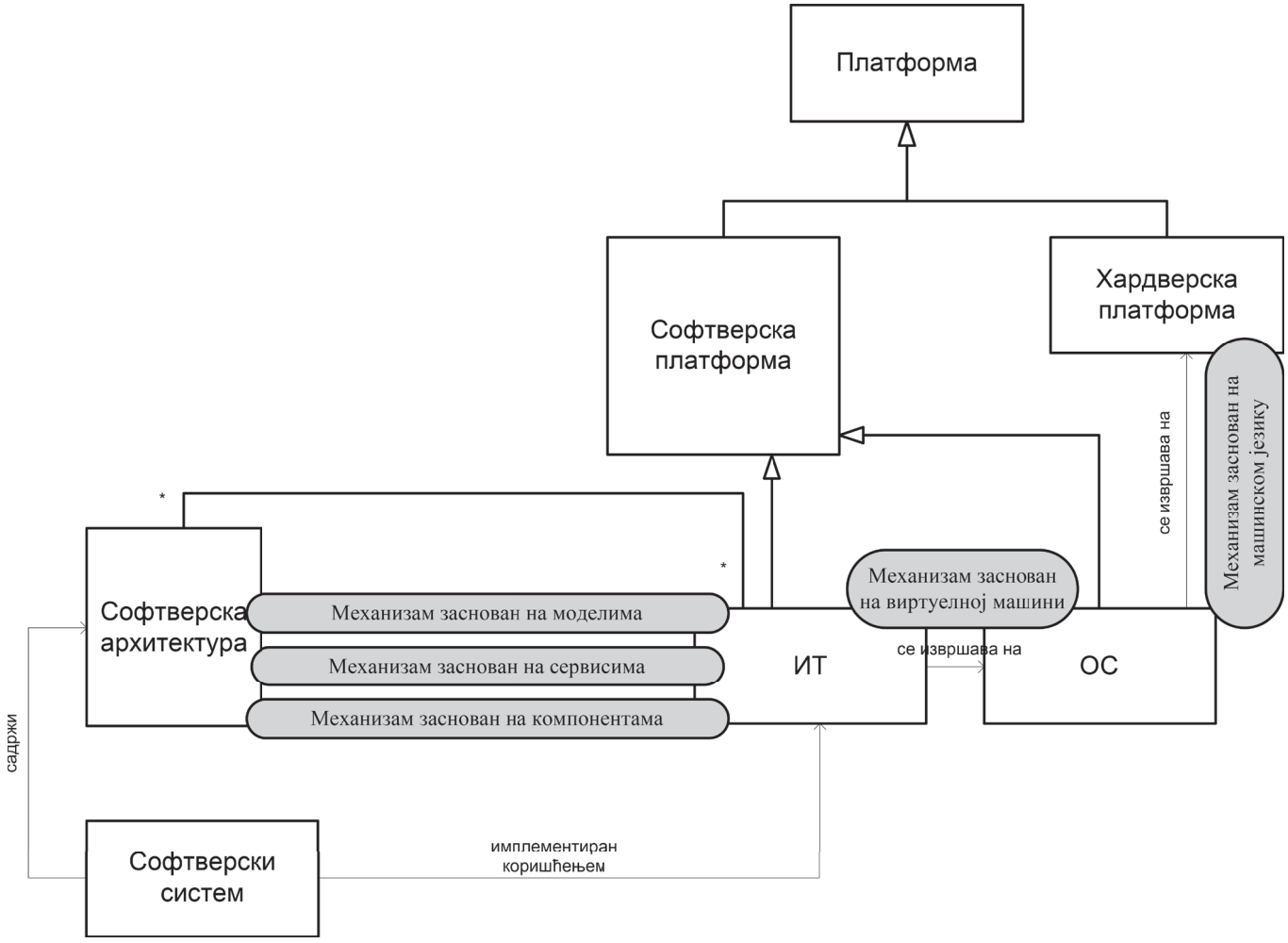
- PI1 – Независност софтверске архитектуре у односу на имплементациону технологију. У овој категорији разматрали смо појединачно:
  - PI11 – Независност MDA у односу на имплементациону технологију
  - PI12 – Независност COA у односу на имплементациону технологију
  - PI13 – Независност SOA у односу на имплементациону технологију
- PI2 – Независност имплементационе технологије у односу на оперативни систем
- PI3 – Независност оперативног система у односу на хардверску платформу

4. МЕХАНИЗМИ ЗА РЕАЛИЗАЦИЈУ ПЛАТФОРМСКЕ НЕЗАВИСНОСТИ

У овом делу рада описаћемо четири кључна механизма која се користе за остваривање платформске независности:

1. Механизам заснован на сервисима (Services-based mechanism)
2. Механизам заснован на компонентама (Component-based mechanism)
3. Механизам заснован на моделима (Model-based mechanism)
4. Механизам заснован на виртуелној машини (Virtual machine-based mechanism)

Ови механизми су приказани на Слици 2.



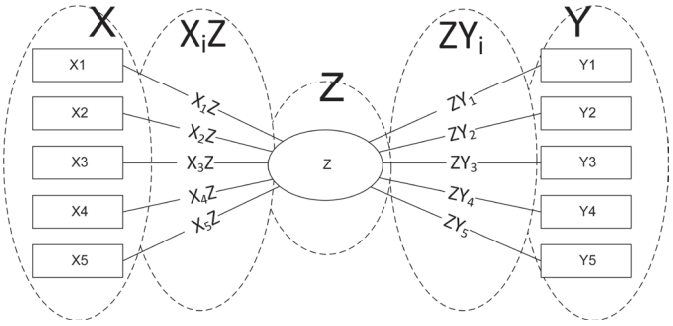
Слика 2: *Везе између софтверског система, софтверске архитектуре и платформе*

Механизам заснован на моделима, механизам заснован на сервисима, као и механизам заснован на компонентама се користе за остваривање независности софтверске архитектуре у односу на софтверску платформу.

Механизам заснован на виртуелној машини се корити за остваривање независности имплементационе технологије у односу на оперативни систем на којем се софтверски систем извршава. Независност између софтверске архитектуре и оперативног система се се постиже индиректно (транзитивна релација) коришћењем овог механизма.

Поред поменута четири механизма која су детаљно обрађена у раду, идентификован је и механизам заснован на машинском језику који омогућава оперативном систему независност у односу на хардверску платформу. У раду се неће детаљније разматрати овај механизам, а независност софтверске архитектуре од хардверске платформе се постиже индиректно (транзитивна релација) коришћењем овог механизма.

Сваки од поменутих механизама ће бити објашњен коришћењем Општег модела платформске независности (GCoI) који је приказан на Слици 3.



Слика 3. *Општи модел платформске независности (GCoI)*

GCoI модел представља основни концепт на којем је заснована платформска независност. Модел чини пет елемената (X, XZ, Z, ZY,Y). За сваки тип платформске независности (PI1, PI2 и PI3) идентификовани су X и Y као елементи међу којима треба успоставити независност. X и Y можемо посматрати као пар који чине софтверска архитектура (SOA, COA, MDA) и имплементациона технологија (Java, .Net) за независност типа PI1, односно пар

који чине имплементациона технологија (Java, .Net) и оперативни систем (Windows2000,WindowsXP, Linux) за независност типа PI2, као и пар који чине оперативни систем (Windows2000, Linux) и хардверска платформа (x86 PC, AS/400) за независност типа PI3. Како би била постигнута независност између X и Y потребно је да се уведе независна компонента Z. Сваки елемент из скупова X и Y треба да има другачију трансформацију према елементу Z (XiZ or ZYj, i=1..n, j=1..m). Ако XiZ и ZYj постоје, тада можемо рећи да је скуп X независан од скупа Y.

Зато насупрот зависности свих елемената из скупа X (X1.. Xn) од сваког елемента из скупа Y (Y1 …Ym), елементи су једино зависни од елемента Z. GCoI модел се може посматрати са два аспекта: аспекта структуре, када узимамо у обзир све елементе укључене у постизање независности, или као процес трансформације између елемената скупова X и Y.

GCoI модел може бити примењен на различите механизме за постизање платформске независности. Ако размотримо дефиницију патерна коју је дао Christopher Alexander (Christopher et al. 1997), можемо закључити да GcoI модел задовољава ову дефиницију, и да се може сматрати патерном. У софтверском инжењерству се патерни широко користе за решавање проблема зависности (coupling) између класа и објеката. За развој одрживих софтверских система важно је смањити зависност на најмању могућу меру. У наредном делу рада описаћемо механизме за остваривање платформске независности коришћењем GCoI модела.

Механизам заснован на сервисима се користи за остваривање независности софтверских компонената које су неком иплементационом технологијом имплементиране обично као сервиси (нпр. веб сервиси), у односу на компоненте које су имплементиране неком другом имплементационом технологијом.

Сервисно оријентисано рачунарство (Service-Orientated Computing (SOC)) је постао главни тренд у софтверском инжењерству који препоручује пројектовање система засновано на сервисима (Erl, 2005; Jonathan et al. 2008).

Веб сервиси пружају стандардан начин за постизање интероперабилности између различитих софтверских система који се извршавају на различитим платформама. Веб сервис је софтверски систем пројектован да подржи мрежну интеракцију са другим системима. Садржи интерфејс који је описан машински читљивим форматом (обично WSDL). Други системи интерагују са веб сервисом, на начин описан интерфејсом, коришћењем SOAP порука, које су серијализоване у XML формату, обично пренесене коришћењем HTTP-а, уз коришћење осталих веб стандарда.

Платформска независност веб сервиса се може уочити у два различита тренутка посматрања:

1. време развоја и
2. време извршења.

Важно је извршити анализу обе ситуације јер у свакој од њих можемо уочити различите технологије, трансформације, протоколе, језике, али исти принцип. Овај принцип је у сагласности са GCoI моделом.

На основу Општег модела платформске независности (GCoI) идентификовали смо кључне елементе механизма заснованог на сервисима. Елементи су приказани у Табели 1.

Када посматрамо веб сервис у процесу развоја, можемо уочити да се обе стране – пружалац сервиса и корисник сервиса, могу имплементирати коришћењем различитих технологија.

Табела 1. *Механизам заснован на сервисима примењен на процес развоја веб сервиса*

X – пружалац сервиса	**XZ**	**Z**	**ZY**	**Y-** корисник сервиса
Java	wsgen	WSDL	wsimport	Java
.Net	wsdl		wsdl	.Net
C++	wsdl2h		soapcpp2	C++
Delphi	delphi IDE		Delphi IDE wsdl importer	Delphi

Обе стране треба да имају приступ WSDL документу (Табела 1, колона Z). Приступом WSDL документу обе стране могу генерисати програмски код компонената које ће приликом извршења бити коришћене за успостављање комуникације (Табела 1, колона XZ – алати за генерисање компонената за пружаоца сервиса, колона ZY – алати за генерисање компонената за корисника сервиса). На овај начин XML засновани WSDL документ остаје једина компонента која везује различите технологије омогућавајући платформски независни развој пружалаца сервиса и корисника сервиса.

Када посматрамо веб сервис у време извршења, платформска независност се остварује коришћењем SOAP (Табела 2).

Табела 2. *Механизам заснован на сервисима и примењен у време извршења веб сервиса*

X – пружалац сервиса	**XZ**	**Z**	**ZY**	**Y-** корисник сервиса
Java	SOAP	SOAP Message	SOAP	Java
.Net				.Net
C++				C++
Delphi				Delphi

Када корисник сервиса позове неку функционалност веб сервиса, захтев се трансформише у у SOAP поруку (Табела 2, колона Z) и доставља пружаоцу сервиса најчешће коришћењем HTTP протокола. SOAP порука се тада трансформише у платформски специфични позив компоненте пружаоца сервиса која ће извршити захтевану функционалност и, по потреби, формирати одговор и послати га коринику сервиса на исти начин.

Механизам заснован на компонентама се користи за остваривање независности софтверских компонената, об-

ично имплементирани неком имплементационом технологијом као сервиси (нпр. CORBA), у односу на компоненте имплементиране неком другом имплементационом технологијом.

Развој софтвера који је одржив, ефикасан и високо флексибилан, развој софтвера заснован на компонентама може се користити за развој сложених софтверских система (Jonathan et al. 2008). *The Common Object Request Broker Architecture* (CORBA) је стандард дефинисан од стране *Object Management Group* (OMG) и омогућава софтверским компонентама које су написане различитим програмским језицима и које се извршавају на различитим рачунарима да раде заједно (тј. подржава више различитих платформи), Омогућава платформски независну, језички независну софтверску архитектуру за прављење дистрибуираних објектно оријентисаних апликација.

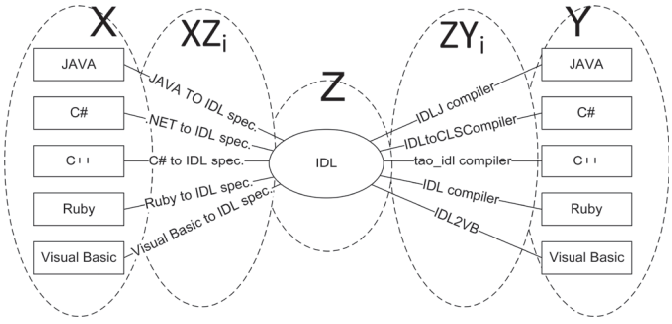
Постоје двије перспективе које можемо узети у обзир приликом описивања платформске независности на примеру технологије CORBA, као примеру механизма заснованог на компонентама:

1. време развоја и
2. време извршења.

Када посматрамо механизам заснован на компонентама у процесу развоја, Interface Definition Language (IDL) представља кључни концепт за остваривање платформске независности софтверског система који је развијен коришћењем једне имплементационе технологије, у односу на неку другу имплементациону технологију. На основу Општег модела платформске независности (GCoI), идентификовани су кључни елементи:

- Различите имплементационе технологије које припадају елементу X у GCoI моделу
- Различите спецификације које обезбеђују трансформацију из имплементационе технологије у IDL интерфејс (елемент XZ у GCoI моделу)
- IDL интерфејс (елемент Z у GCoI моделу)
- Различити компајлери који генеришу код у различитим имплементационим технологијама из IDL интерфејса (елемент ZY у GCoI моделу)
- Различите имплементационе технологије које припадају елементу Y у GCoI моделу

Сви ови елементи су приказани на Слици 4 и у Табели 3.



Слика 4: GCoI примењен на CORBA у процесу развоја

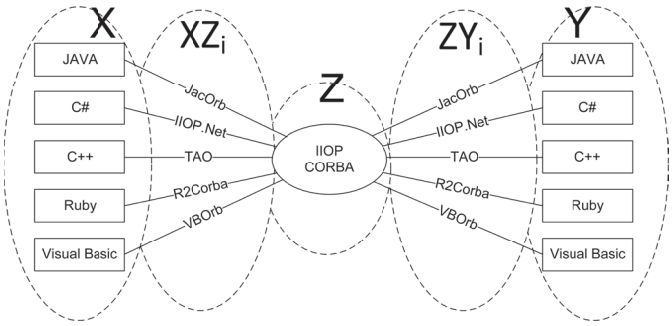
Табела 3. *GCoI примењен на CORBA у процесу развоја*

X	XZ	Z	ZY	Y
Java	JAVA to IDL specification	IDL interface	IDLJ compiler	Java
C#	.NET to IDL specification	IDL interface	IDLtoCLS compiler	C#
C++	C++ to IDL specification	IDL interface	Tao_idl compiler	C++
Ruby	Ruby to IDL specification	IDL interface	IDL compiler	Ruby
Visual Basic	Ruby to IDL specification	IDL interface	IDL2VB	Visual Basic

Ако узмемо у обзир време развоја, клијентске и серверске класе, које су генерисане коришћењем тзв. ORB компајлера за програмске језике, комуницирају међусобно помоћу класа произведених од стране ORB-а. ORB компоненте комуницирају коришћењем IOOP протокола који омогућава платформску независност на обе стране (клијентској и серверској). Зато је IOOP протокол кључни концепт који омогућава различитим софтверским компонентама да међусобно комуницирају. На основу Општег модела платформске независности (GCoI), идентификовани су кључни елементи:

- Различите имплементационе технологије (Java, C#...) које припадају елементу X у GCoI моделу
- Различите имплементације CORBA стандарда (елемент XZ у GCoI моделу)
- IOOP CORBA (елемент Z у GCoI моделу)
- Различите имплементације CORBA стандарда (елемент ZY у GCoI моделу)
- Различите имплементационе технологије (Java, C#...) које припадају елементу Y у GCoI моделу

Сви ови елементи су приказани на Слици 5. и у Табели 4.



Слика 5. *GCoI примењен на CORBA у време извршења*

Табела 4. *GCoI примењен на CORBA у време извршења*

X	XZ	Z	ZY	Y
Java	JacOrb	IIOP CORBA	JacOrb	Java
C#	IIOP.Net		IIOP.Net	C#
C++	TAO		TAO	C++
Ruby	R2Corba		R2Corba	Ruby
Visual Basic	VBOrb		VBOrb	Visual Basic

Механизам заснован на моделима се користи за остваривање платформске независности софтверске архитектуре у односу на софтверску платформу. Овај механизам је имплементиран у приступу Моделом вођене архитектуре (Model-Driven Architecture - MDA).

Концепт Моделом вођене архитектуре је објављен од стране OMG. Заснован је на креирању модела и трансформација између њих. OMG описује различите типове модела и њихових веза, али без спецификације начина на који су модели креирани и који се тачно модели и нотације користе за њихову репрезентацију, као ни на који начин се трансформишу једни у друге. Три нивоа модела на вишем нивоу апстракције се праве као графички, док је модел на најнижем нивоу модел имплементационог кода.

Computation Independent Model (CIM) – CIM је модел који не описује детаље пројектовања информационог система али прецизира активности које се извршавају информационим системом. Представља пословне проесе организације за коју се развија информациони систем (Kardos, 2010).

Platform Independent Model (PIM) – PIM је модел који описује информациони систем, али сакрива детаље коришћења конкретне технологије. PIM описује понашање и структуру система. Не описује оперативни систем, програмски језик и хардвер. PIM модели се користе за моделовање функционалности и структуре информационог система независно од технолошких детаља платформе на којој ће бити имплементиран.

Platform Specific Model (PSM) – PSM повезује спецификацију PIM са детаљима који специфицирају тип платформе коју ће користити информациони систем. PSM је одговоран за спецификацију техничких детаља који су потребни за имплементацију PIM, нпр. који оперативни систем ће бити коришћен, програмски језик и сл.

Implementation model (IM) – IM представља платформски специфичан програмски код. Овај модел се обично генерише из PSM, али може бити генерисан и из PIM. Представља код који може бити директно компајлиран и уведен без људске интервенције.

На основу GCoI модела идентификовали смо кључне елементе MDA. Ови елементи су представљени у Табели 5.

Табела 5 *GCoI примењен на MDA*

X - CIM	XZ - CIM-PIM	Z - PIM	ZY - PIM-PSM	Y - PSM
UML activity diagram Business Process Model Data Flow Diagram	\ Query/View/ Transformation DSL	UML use case UML activity diagram UML activity diagram UML use case diagram UML sequence diagram UML class diagram	UML Profile Query/View/ Transformation DSL GPL	JavaEE .Net ORM

PIM у MDA представља елемент који се користи за остварење независности софтверске архитектуре у односу на софтверску платформу. Различите трансформације из CIM у PIM могу бити написане доменски специчичним језицима као и трансформације из PIM у CIM.

Механизам заснован на виртуелној машини се користи за постизање независности софтверског система, који је имплементиран одређеном имплементационом технологијом у односу на оперативни систем на којем се софтверски систем извршава. На Слици 13 приказан је овај механизам.

Овај механизам се донекле разликује од осталих механизама за остваривање платформске независности, јер није директно везана за софтверски систем или софтверску архитектуру, већ индиректно, кроз имплементациону технологију која је коришћена за имплементацију софтверског система. Раније је објашњено да су имплементационе технологије и оперативни системи различити типови софтверских платформи. Механизам заснован на виртуелним машинама омогућава независност имплементационих технологија у односу на оперативни систем. Ово је кључан механизам јер, у комбинацији са осталим механизмима, омогућава независност различитих софтверских архитектура у односу на различите оперативне системе, а тиме и у односу на различите хардверске платформе.

Виртуалне машине су коришћене у бројним областима, од оперативних система до програмских језика и процесорских архитектура. Упркос чињеници да су веома сложени, софтверски системи постоје и настављају да еволуирају јер су пројектовани као хијерархија веома добро дефинисаних интерфејса који јасно раздвајају нивое апстракција. Апстракције сакривају имплементационе детаље ниског нивоа, и на тај начин се поједностављује процес пројектовања. Java и .Net су технологије и софтверске платформе које обе имају механизме за реализацију платформске независности софтверских система – апликација у односу на хардверску платформу и оперативни систем. Java и .Net имају веома сличан приступ имплементацији платформске независности.

Java виртуелна машина је виртуелни процесор који омогућава да се исти софтвер извршава на различитим платформама, јер се сам софтвер не извршава директно на оперативном систему, већ се извршава коришћењем Java виртуелне машине. Java апликације могу да се извршавају на свим платформама за које постоји Java виртуелна машина (JVM).

Такође, циљ компаније Microsoft у развоју .Net платформе је да омогући независност .Net апликација од хардверских платформи и оперативних система. У процесу компајлирања .Net изворног кода, компајлери уместо машинског кода производе инструкције у заједничком међу-језику - Common Intermediate Language (CIL). Ове инструкције се преводе у машински језик на машини на којој се апликација извршава.

На основу Општег модела платформске независности (GCoI), идентификовани су кључни елементи:

- Различите имплементационе технологије које припадају елементу X у GCoI моделу
- Различити компајлери који преводе изворни код у Java Byte code или CIL (елемент XZ у GCoI моделу)
- Java Byte code или CIL (елемент Z у GCoI моделу)
- Различити интерпретери који интерпретирају преведени изворни код за конкретан оперативни систем (елемент ZY у GCoI моделу)
- Различити оперативни системи који припадају елементу Y у GCoI моделу

Сви ови елементи су приказани на у Табели 6.

Табела 6. *Механизам заснован на виртуелној машини примењен на примеру Java и C#*

X	XZ	Z	ZY	Y
Java	javac	Bytecode	JVM	Windows, Solaris, Linux
C#	csc	CLI	CLR	Windows 2000,Windows XP, Windows 7

Сличан механизам се користи и код програмских језика као што су Jython, Scala, Boo и IronPython. У табели 7 представљени су кључни елементи GCoI модела примењени на ове програмске језике

Табела 7. *Механизам заснован на виртуелној машини примењен на примеру осталих програмских језика*

X	XZ	Z	ZY	Y
Jython	jythonc	Bytecode	JVM	Windows, Linux
Scala	scalac	Bytecode		Windows, Linux
IronPython	pyc	CLI	CLR	Windows, Linux
Boo	booc	CLI		Windows XP, Windows 7.

Табела 8 приказује везе које постоје између типова платформске независности и механизама за њихову реализацију. Колоне PI11 до PI3 представљају типове платформске независности, док редови R1 до R4 означавају типове механизама који су коришћени за њихову реализацију.

Табела 8. *Везе између типова платформске независности и механизама реализације*

▼ МЕХАНИЗМИ РЕАЛИЗАЦИЈЕ ▼	ТИПОВИ ПЛАТФОРМСКЕ НЕЗАВИСНОСТИ				
	PI11	***PI12***	***PI13***	***PI2***	***PI3***
R1 (механизам заснован на сервисима***)***			•		
R2 (механизам заснован на компонентама***)***		•			
R3 (механизам заснован на моделима***)***			•		
R4 (механизам заснован на виртуелним машинама***)***	•				
R5 (механизам заснован на машинским језицима***)***					•

5. РЕЗИМЕ И ЗАКЉУЧАК

Постоје бројне књиге и радови који се баве Java архитектуром која подржава платформску независност (McGovern et al. 2003; Gong et al. 2003) као и Веб сервисима као платформски независним моделом (Szyperski, 1999; Alonso, 2004) и CORBA платформски независним моделом (Merle et al. 1997; Merle et al. 1996; Wang et al. 2000). Иако сви ови радови описују платформски независне моделе, нисмо успели пронаћи радове који на интегралан начин анализирају све поменуте моделе.

У литератури не постоји истраживање које анализира различите механизме за остваривање платформске независности на овај начин, па можемо рећи да је ово прва студија ове врсте са циљем да се не само анализирају и пореде начини реализације платформске независности, већ и да се дефинишу општа правила и законитости који је омогућавају.

Кључни допринос овог рада је увођење GCoI модела и идентификација и објашњење различитих механизама остваривања платформске независности. GCoI модел представља основни концепт на којем је заснована платформска независност.

Овај модел је значајан из неколико разлога. Приликом развоја нове платформе или софтверске архитектуре, инжењери морају бити свесни свих потребних елемената који морају постојати како би се остварила платформска независност. У ери рачунарства у облаку (cloud computing) и паметних уређаја, потреба за оваквим решењима и механизмима за „адаптирање“ расте. GCoI модел представља општи принцип за реализацију ових механизама.

Наша практична искуства као универзитетских професора потврђују вредност ових модела у процесу учења. Када су студени упознати са GCoI моделом и концептом платформске независности, много лакше и брже разумеју везе између софтверских архитектура и платформи.

REFERENCES

[1] Alonso, G., Casati, F., Kuno, H. and Machiraju, V. (2004), *Web Services: Concepts, Architecture and Applications*. Springer

[2] Bass, L., Clements, P., Kazman, R. (2003), *Software Architecture in Practice* (Second Edition). Addison-Wesley

[3] Buschmann, F. (1996), Pattern-Oriented Software Architecture: A System of Patterns. *John Wiley & Sons*

[4] Christopher, A., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S. (1977) A Pattern Language. *Oxford University Press*, New York

[5] Erl, T. (2005), Service-Oriented Architecture – Concepts, Technology, and Design. *Pearson Education, Inc.*

[6] Gong, L., Ellison, G. and Dageforde, M. (2003) *Inside Java™ 2 Platform Security: Architecture, API Design, and Implementation*. Addison Wesley

[7] Gorton, I. (2011), *Essential Software Architecture* (2nd Edition), Springer

[8] Gorton, I. (2011), *Essential Software Architecture* (2nd Edition), Springer

[9] Jonathan, L., Shang-Pin, M., Ying-Yan, L., Shin-Jie, L., Yao-Chiang, W. (2008), Dynamic Service Composition: a Discovery-Based Approach. *International Journal of Software Engineering and Knowledge Engineering*, 199-222

[10] Kardos, M. (2010), Analytical method of CIM to PIM transformation in Model Driven Architecture (MDA). *Journal of Information and Organizational Sciences*; Vol 34, No 1.

[11] Liu, Y., Liang, X., Xu, L., Staples M., and Zhu L. (2011), Composing enterprise mashup components and services using architecture integration patterns. *Journal of Systems and Software*, Elsevier, Vol. 84, No. 9, pp. 1436-1446

[12] McGovern, J., Tyagi, S., Stevens, M., Mathew, S. (2003), *Java Web Services Architecture.* Addison Wesley

[13] Meghan, K., Model Driven Architecture Straight From The Masters, *The MDA Journal*

[14] Merle, P., Gransart, C. and Geib. J.M. (1996) CorbaWeb: A Generic Object Navigator. *Proceedings of the Fifth International World-Wide Web Conference*

[15] Merle, P., Gransart, C., Geib, J.M. (1997) Generic tools: a new way to use Corba. *In European Conference on Object-Oriented Programming*, *Workshop on CORBA: Implementation, Use, and Evaluation*

[16] Object Management Group (2003), Ontology Definition Metamodel Request For Proposal OMG Document: ad/2003-03-40

[17] Szyperski, C., *Component Software: Beyond Object-Oriented Programming*. Addison- Wesley, New York

[18] Vitvar, T., Mocan, A., Kerrigan, M., Zaremba, M., Maciej, M., Moran, M., Cimpian, E. (2007). Semantically-enabled service oriented architecture: concepts, technology and application. *Service Oriented Computing and Applications*, *1*(2), 129-154. Springer

[19] Wang, N., Schmidt, D. C. and Levine, D. (2000) Optimizing the CORBA Component Model for High-performance and Real-time Applications. *ACM/IFIP*

dr Siniša Vlajić, Fakultet organizacionih nauka Univerzitet u Beogradu.
Kontakt: vlajic@fon.bg.ac.rs
Oblasti interesovanja: Softverski proces, održavanje softvera, formalizacija softverskih paterna

dr Ilija Antović, Fakultet organizacionih nauka Univerzitet u Beogradu.
Kontakt: ilijaa@fon.bg.ac.rs
Oblasti interesovanja: Automatizacija razvoja korisničkog interfejsa, softverski zahtevi, softverski paterni, generatori koda

mr Dušan Savić, Fakultet organizacionih nauka Univerzitet u Beogradu.
Kontakt: dules@fon.bg.ac.rs
Oblasti interesovanja: Modelovanje i meta-modelovanje, softverski zahtevi, domenski specifični jezici, Java web programiranje

Marija Vidaković, IGT - Ogranak Beograd
Kontakt: marija.vidakovic@igt.com
Oblasti interesovanja: Softverske Arhitekture i Integracije, Softverski zahtevi, Dizajn korisničkog Interfejsa