Dr Milunka Damnjanović, red.prof,
OBJEKTNO ORIJENTISANE TEHNIKE PROJEKTOVANJA SISTEMA

15 Implementacija sistema

Faza implementacije:

Obavlja implementaciju modela projekta.

Razmatra ne-functionalne zahteve i razmeštanje izvršnih modula na čvorovima.

U toku ove faze razvijaju se dva modela:

1) model implementatije koji opisuje kako će elementi projekta biti implementirani kao softverski sistem,

2) model razmeštanja koji opisuje kako će implementirani softver biti razmešten na fizičkom hardveru.

Model implementacije:

Model implementacije prikazuje kako se različiti aspekti projekta prslikavaju u ciljni programski jezik.

On opisuje u kakvom su odnosu komponente, interfejsi, paketi i fajlovi.

Elementi koji se modeluju su:

1) paketi,

2) komponente,

3) njihovi međusobni odnosi

i oni su prikazani pomoću dijagrama implementacije komponenata.

Paketi – Packages:

Paket predstavlja fizičku viziju sistema.

Paketi su organizovani u hijerarhiji slojeva.

Paketi se koriste za:

1) asociranje odgovarajućih komponenata i interfejsa,

2) rešavanje problema sa imenovanjem,

3) obezbeđivanje privatnosti za klase, atribute i operacije koje ne treba da se vide izvan paketa.

Komponente:

Komponenta:
1) je fizički deo sistema,
2) je zamenljiva,
3) obezbeđuje realizaciju skupa interfejsa i prilagođava im se.

Komponente se grupišu u pakete.

Komponente se mogu organizovati specificiranjem zavisnosti, generalizacije,asocijacije, agregacije i realizacije odnosa među njima.

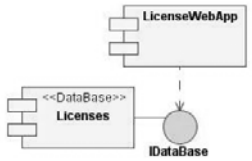
Modelovanje komponente vrlo je važno za kontrolu upravljanja verzija i konfiguracije u toku razvoja sistema.

5

Komponente i interfejsi:

• Komponenta koja realizuje neki interfejs vezuje se na njega pomoću njegove realizacije.

• Komponenta koja pristupa servisu druge komponente pomoću interfejsa vezuje se na interfejs koristeći odnose zavisnosti.



6

Tehnike modelovanja:

Za modelovanje statičkog aspekta implementacije sistema koriste se dijagram komponenata.

Da bi se modelovao ovaj aspekt, moguće je koristiti dijagrame komponenata na četiri načina:

1) modelovanje izvornog koda,

2) modelovanje izvršnih izvedbi,

3) modelovanje fizičkih baza podataka,

4) modelovanje adaptabilnih sistema.

7

Modelling Source Code

Component diagrams are used to model the configuration management of source files, which represent work-product components.

Modelling procedure:

1) identify the set of source code files of interest, either by forward or reverse engineering, and model them as components stereotyped as files

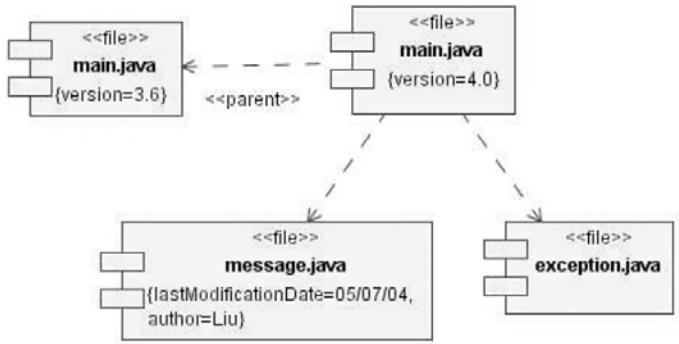
2) for larger systems, use packages to show groups of source code files

3) consider exposing a tagged value to show interesting information such as author, version number, etc.

4) model the compilation dependencies among these files using dependencies

8

Primer: Fajlovi izvornog koda



Modelovanje izvršivih rešenja:

Diagrami komponenata koriste se za vizualizaciju, specifikaciju, konstrukciju i dokumentaciju konfiguracije izvršivih rešenja, uključujući razmeštanje komponenata koje formiraju svako rešenje i odnose između tih komponenata.

Svaki dijagram komponenata fokusira se na jedan skup komponenata u vremenu, kao npr. svih komponenata koje se nalaze na jednom čvoru.

Procedura modelovanja:

1) identifikovati skup komponenata koje treba modelovati,

2) razmotriti stereotip svake komonente u skupu,

3) za svaku komponentu, razmotriti njen odnos sa njenim susedima.

10

Modelovanje fizičkih baza podataka:

Diagrami komponenata koriste se za vizualizaciju, specifikaciju, konstrukciju i dokumentaciju preslikavanja klasa u tabele u bazi podataka.

Procedura modelovanja:

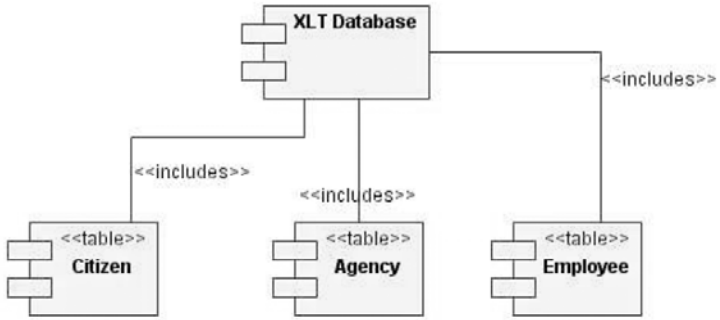
1) identifikovati klase u modelu koje predstavljaju logičnu šemu baze podataka,

2) Odabrati strategiju za preslikavanje ovih klasa tabelama, možda razmatrajući fizičku didstribuciju,

3) Kreirati dijagram komponente koji sadrži komponente stereotipovane kao tabele kojima se modeluje preslikavanje.

11

Primer – fizičke baze podataka:



12

Modelovanje adaptabilnih sistema #1:

Pomenuti dijagrami komponenata modelovali su statički aspekt.

Statički aspekt znači da njegove komponente provedu ceo život na jednom čvoru.

U nekim distribuiranim sistemima neophodno je modelovati dinamički aspekt.

Dinamički aspekt modeluje kcomponente koje se premeštaju sa jednog čvora na drugi.

Da bismo modelovali dinamički aspekat, potrebna nam je kombinacija:

1) dijagrama komponenata,

2) dijagrama objekata,

3) dijagrama interakcije.

13

Modelovanje adaptabilnih sistema #2:

Procedura modelovanja :

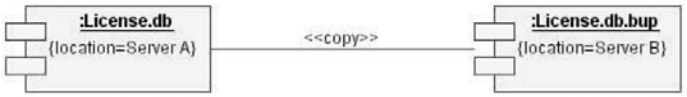
1) razmotriti fizičku distribuciju komponenata koje moraju da migriraju sa čvora na čvor,

2) specificirati lokaciju uzorka komponente markirajući ga veličinom markera koja se dalje može prikazati u dijagramu komponente,

3) modelovati akcije koje uzrokuju migraciju komponente kreirajući odgovarajući dijgram interakcije koji sadrži uzorke komponente.

14

Modelovanje adaptabilnih sistema #3:



15

Tabele, fajlovi i dokumenti :

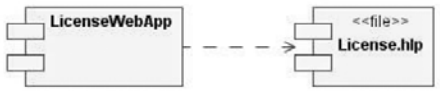
Implementacija može da obuhvati fajlove podataka, pomoćne dokumente, beleške, log fajlove, i fajlove za instalaciju i uklanjanje.

Procedura modelovanja :

1) identifikovati proizvoljne komponente koje su deo fizičke implementacije,

2) modelovati ove stvari kao komponente (uvesti nove stereotipe ako je potrebno),

3) modelovati odnose između proizvoljnih komponenata i ostalih izvršnih.



16

API:

An Application Programming Interface (API) prirodno je interfejs koji se ostvaruje pomoću jedne ili više komponenata.

Jedan koncept, dve perspektive:

1) kao onaj ko razvija: zainteresovan samo za interfejs,

2) kao management konfiguracije sistema: zainteresovan u kojoj komponenti se ostvaruje interfejs.

Operacije asocirane sa semantički bogatim API biće prilično ekstenzivne, i uglavnom nije potrebno vizualizovati ih. Umesto toga, modeluju se interfejsi koji grupišu ove operacije.

17

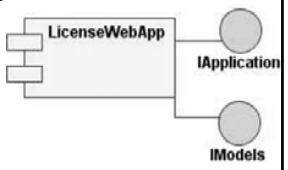
Modelovanje API:

Procedura:

1) identifikovati programske izglede i to kao interfejse, prikupljajući odgovarajuće atribute i operacije,

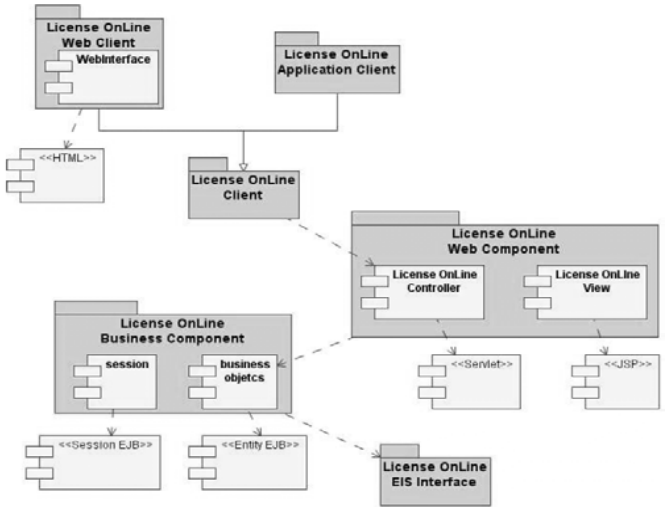
2) izložiti samo one osobine interfejsa koje su važne za vizualizaciju u datom kontekstu,

3) modelovati realizaciju svake API samo ako je neophodno da bi se prikazala konfiguracija specifične implementacije.



18

Primer:



Ukratko:

Diagrami implementacije komponenata opisuju u kakvom su odnosu komponente, interfejsi, paketi i fajlovi.

Diagrami implementacije komponenata mogu se koristiti za modelovanje izvornog koda izvršnih rešenja, fizičkih baza podataka i adaptabilnih sistema.

20

Dijagrami razmeštaja:

Diagram razmeštaja prikazuje konfiguraciju procesnih čvorova pri izvršenju i komponente koje žive na njemu.

Oni se koriste za modelovanje distribucije, isporuke i instalacije delova koji čine fizički sistem.

To uključuje modelovanje topologije hardvera na kome se sistem izvršava.

Dijagrami razmeštaja su suštinski dijagrami klasa koji se fokusiraju na sistemske čvorove i obuhvataju:

1) čvorove,

2) relacije zavisnosti i asocijacija,

3) komponente, i

4) pakete.

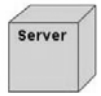
21

Čvorovi:

Čvorovi se koriste za modelovanje topologije hardvera na kome se izvršava sistem.

Komponente koje su ili razvijene ili se ponovo koriste, moraju biti razmeštene na nekom skupu hardvera da bi se izvršavale.

Čvorovi predstavljaju hardver na kome se ove komponente nalaze i izvršavaju.



UML notacija za čvorove dozvoljava vizualizaciju čvora nezavisno od specifičnog hardvera.

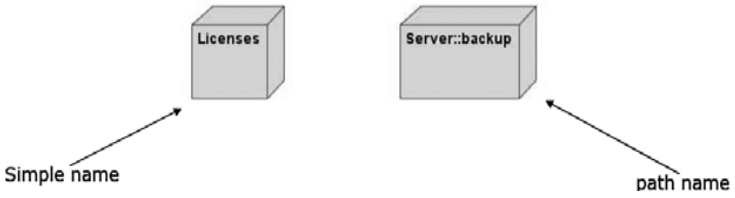
Stereotipi dopuštaju predstavljanje specifičnih vrsta procesora i uređaja.

22

Imena čvorova:

Svaki čvor mora imati ime po kome se razlikuje od ostalih čvorova.

Ime je tekstualni string koji se može pisati kao prosto ime ili kaoime puta.



Čvorovi i komponente

Komponente	Čvorovi
• učestvuju u izvršavanju sistema,	• izvršavaju komponente,
• predstavljaju fizičko pakovanje inače logičkih elemenata.	• predstavljaju fizičko razmeštanje komponenata.

Odnos razmeštanja između čvora i komponente može se prikazati pomoću odnosa razmeštaja.

Skup objekata ili komponenata koje su dodeljene čvoru kao grupa zovu se distribuciona jedinica.

24

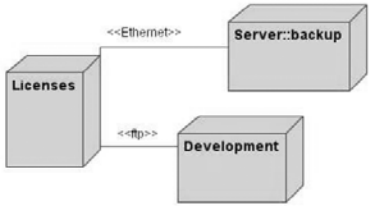
Organizovanje čvorova:

Čvorovi se mogu organizovati:

1) na isti način kao klase i komponente

2) specificiranjem razmeštaja, generalizacije, asocijacije, aggegacije, i realizacije odnosa između njih.

Najčešća vrsta odnosa korišćena između čvorova je asocijacija koja predstavlja fizičku vezu među njima.



Procesori i uređaji:

Processor je čvor koji ima sposobnost procesiranja (obrade). On može da izvršava komponentu.

Uređaj je čvor koji nema sposobnost obrade (najmanje na nivou apstrakcije prikazivanja).

26

Modelovanje čvorova:

Procedura:

1) identifikovati računarske elemente sa gledišta razmeštaja sistema i modelovati svaki kao čvor,

2) dodati odgovarajuće stereotipe čvorovima,

3) razmotriti atribute i operacije koje se mogu primeniti na svaki čvor.

27

Distribucija komponenata:

Da bi se modelovala topologija sistema, neophodno je specificirati fizičku distribuciju njegovih komponenata po procesorima i uređajima sistema.

Procedura:

1) dodeliti svaku komponentu u datom čvoru,

2) razmotriti dvostruke lokacije za komponentu ako je potrebno,

3) prikazati dodelu na jedan od sledećih načina:

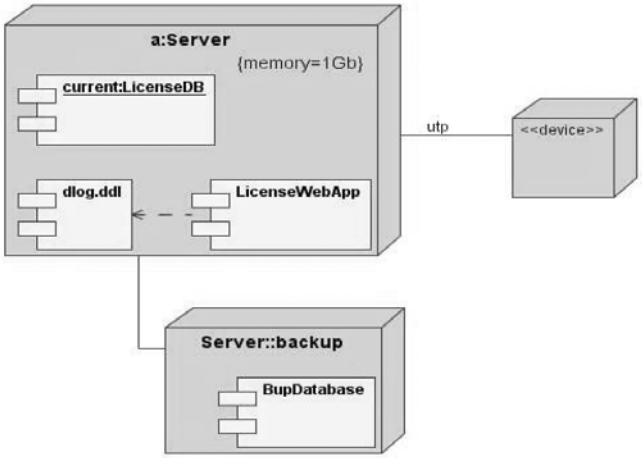
a) dodelu ne učiniti vidljivom,

b) koristiti odnos zavisnosti između čvora i komponente koja mu jedodeljena,

c) Izlistati komponente dodeljene čvoru u nekom posebnom odeljku.

28

Dijagram razmeštaja:



Opšta upotreba:

Dijagrami razmeštanja mogu se koristiti za:

1) ugrađene sisteme: Skup hardvera sa mnogo softvera, koji komunicira sa realnim svetom.

Sadrže softver ko kontroliše uređaje i koji se kontroliše spoljašnjom pobudom.

2) klijent/server sisteme: arhitekture koje prave jasnu razliku između interfejsa korisnika sistema (klijent) i sistemskih podataka (server)

3) distribuirane sisteme: okruženje sa višestrukim serverima.

30

Ugrađeni sistemi:

Procedura modelovanja:

1) identifikovati uređaje i čvorove koji su jedinstveni za sistem,

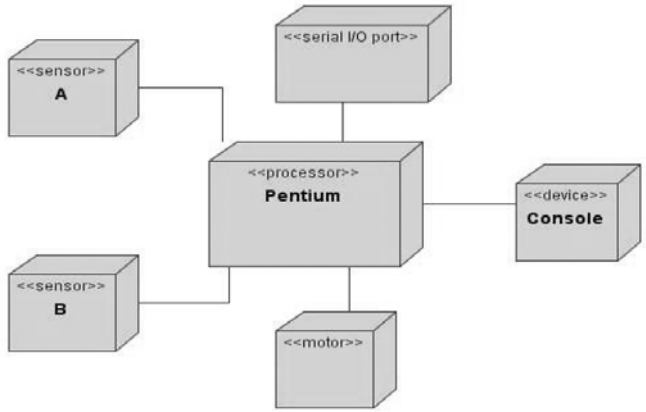
2) omogućiti vizuelnu indikaciju za neobične uređaje sa specifičnim stereotipima i odgovarajućim ikonama, razlikujući procesore i uređaje,

3) modelovati odnos između ovih procesora i uređaja koristeći dijagram razmeštanja i specificirati odnos između komponenata kod gledišta implementacije i čvorova sa gledišta razmeštaja,

4) ako je neophodno, proširiti na proizvoljni inteligentni uređaj modelovanjem njihove strukture detaljnijim dijagramom razmeštanja.

31

Primer - ugrađeni sistem:



32

Klijent/server sistemi:

Procedura modelovanja:

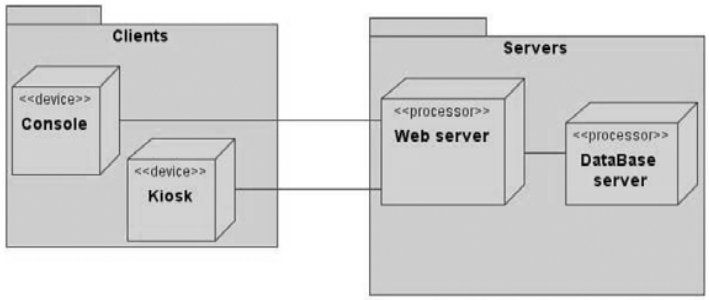
1) identifikovati čvorove koji predstavljaju u sistemu procesore klijenta i servera,

2) istaći uređaje koji su relevantni za sistem,

3) obezbediti vizualnu indikaciju za ove uređaje sa stereotipima,

4) modelovati topologiju ovih čvorova u dijagramu razmeštanja, i specificirati odnose između komponenata sa gledišta implementacije i čvorova sa gledišta razmeštaja.

Primer - klijent/server sistem:



Distribuirani sistemi:

Procedura modelovanja:

1) identifikovati uređaje i procesore sistema,

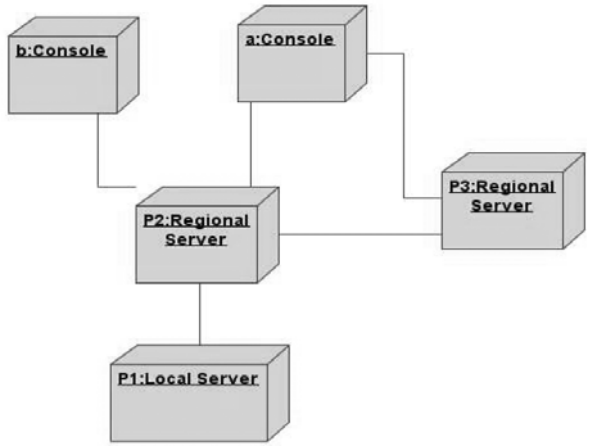
2) modelovati komunikacione uređaje sa dovoljno detalja ako se želi pristup svojstvima sistemske mreže ili uticaj promena na mrežu

3) posvetiti pažnju logičkom grupisanju čvorova koji se mogu specificirati korišćenjem paketa.

4) modelovati ove uređaje i procesore koristeći dijagrame razmeštaja,

5) ako je potrebno fokusirati se na dinamiku sistema, uvesti dijagrame korisničkih slučajeva za specifikaciju ponašanja od interesa, i proširiti ove dijagrame dijagramima interakcije.

Primer - distribuirani system:



Ukratko:

Dijagrami razmeštaja modeluju the topologiju hardvera na kome se sistem izvršava i softver instaliran na svakoj hardverskoj komponenti.

Dijagrami razmeštanja obuhvataju čvorove, pakete, komponente i njihove odnose.

Dijagrami razmeštanja mogu se koristiti za modelovanje različitih vrsta sistema kao što su ugrađeni sistemi, client-server ili distribuirani sistemi.

37

Razni UML CASE alati:

Alati se razlikuju u odnosu na::

1) sposobnost modelovanja na UML-u,

2) podršku u toku celog životnog ciklusa projekta,

3) direktni i povratni inženjering,

4) modelovanje podataka,

5) osobine,

6) cenu,

7) podržljivost,

8) lakoću upotrebe,

9) ...

38

Vrednovanje CASE alata:

Different Criteria for evaluating CASE tools:

1) repository support

2) round-trip engineering

3) HTML documentation

4) UML support

5) data modelling integration

6) versioning

7) model navigation

8) printing support

9) diagrams views

10) exporting diagrams

11) platform

39

Razni UML alati:

- **Enterprise Architect**
  organization: Sparx Systems
  web-site: http://www.sparxsystems.com.au/
- **MagicDraw**
  organization: No Magic Inc.
  web-site: http://www.nomagic.com/
- **Poseidon**
  organization: Gentleware
  web-site: http://www.gentleware.com/
- **Rational Rose**
  organization: IBM
  web-site: http://www306.ibm.com/software/rational/

40

Enterprise Architect

Criteria	Features
Platform	Windows
UML Compliance	Support for all 13 UML 2.0 diagrams
Usability	Replication capable – Comprehensive and flexible documentation
Development Environment	Multi-user enabled – Allows to replicate and share projects
Outputs & Code Generation	C++, Java, C#, VB, VB.Net, Delphi, PHP – HTML and RTF document generation - Forward and reverse database engineering
Data Repository	Models are stored in a data repository - Checks data integrity in the data repository – Provides a project browser
Other features	Allows scripting to extend functionality – Project estimation tools – User definable patterns

Magic Draw

Criteria	Features
Platform	Any where Java 1.4 is supported
UML Compliance	Support for UML 1.4 notation and semantics
Usability	Replication capable – Customizable views of UML elements – Customizable elements properties
Development Environment	Multi-user enabled – Lock parts of the model to edit – Commit changes – Model versioning and rollback
Outputs & Code Generation	Code generation and reverse engineering to C++, Java, C# - RTF and PDF document generation
Data Repository	Provides a project browser
Other features	Friendly and customizable GUI – Hyperlinks can be added to any model element

Poseidon:

Criteria	Features
Platform	Platform independent – Implemented in Java
UML Compliance	Supports all 9 diagrams of UML 1.4
Usability	Replication capable – Internationalization an localization for several languages
Development Environment	Collaborative environment based on client-server architecture – Locking of model parts – Secure transmission of files
Outputs & Code Generation	VB.Net, C#, C++, CORBA IDL, Delphi, Perl, PHP, SQL DDL – Round trip engineering for Java – Diagram export as gif, ps, eps and svg.
Data Repository	Uses MDR (Meta Data Repository) developed by Sun and based on the JMI (Java Metadata Interface) standard
Other features	Allows to import Rational Rose files

Rational Rose:

Criteria	Features
Platform	Windows
UML Compliance	Not fully supported UML 1.4
Usability	The add-in feature allows to customize the environment – User configurable support for UML, OMT and Booch 93.
Development Environment	Parallel multi-user development through repository and private support
Outputs & Code Generation	C++, Visual C++, VB6, Java – Documentation generation – Round trip engineering
Data Repository	Maintains consistency between the diagram and the specification, you may change any of them and automatically updates the information.
Other features	Can be integrated with other Rational products such as RequisitePro, Test Manager

Ukratko:

Postoji nekoliko CASE alata koji podržavaju OO modelovanje pomoću UML-a.

Pri vrednovanju alata, razmatraju se razni kriterijumi..

Ovde su upoređena četiri alata: Enterprise Architect, Magic Draw, Poseidon i Rational Rose.

45

46