Modelarea sistemelor informatice de e-Commerce

Subject: CUMPARA INTELIGENT DIN PRIMA RETEA DE MAGAZINE ONLINE SPECIALIZAT E.
From: "Newsletter i-shops.ro" <ishops.ro@gmail.com>
Date: Thu, 02 Apr 2009 01:27:19 -0100
To: "ciprian.dobre@cs.pub.ro" <ciprian.dobre@cs.pub.ro>

De ce modelăm?

- Aplicaţiile software pot fi complexe; cum le putem gestiona eficient?
  Posibilă soluţie: folosim o bună reprezentare a sistemului

- Model: reprezintă anumite proprietăţi ale unui obiect într-un anumit context
  - Reducerea complexităţii prin ascunderea detaliilor ce nu sunt necesare (abstractizare)
  - Anumite proprietăţi ale sistemului pot deveni mai vizibile (claritate)
  - Facilitează aplicarea unei metodologii
  - În general; modelele oferă o mai bună utilizabilitate

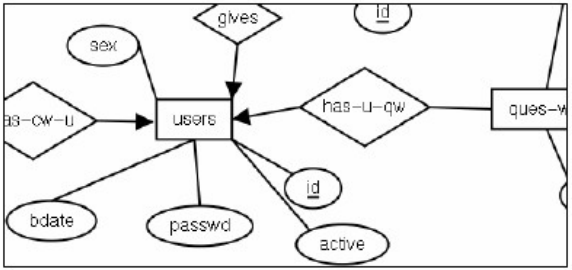
- De obicei se folosesc mai multe modele în paralel; modele diferite pentru diverse scopuri

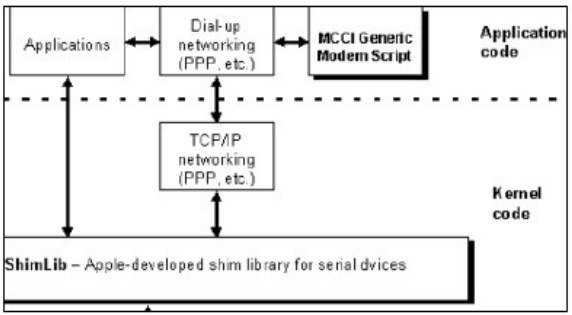
Modele

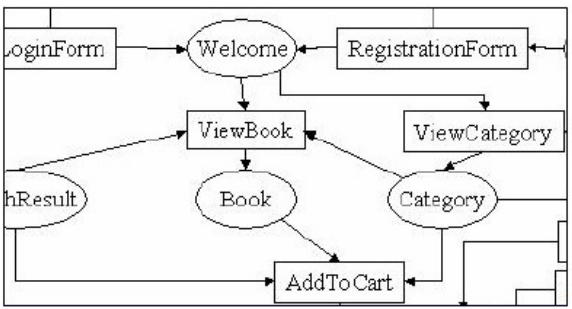
- **Modele de date**
  Tipuri de date şi relaţiile între acestea
  E.g. Diagrame ER, diagrame de clase

- **Modele arhitecturale**
  Componente ale unui sistem şi relaţiile între acestea

- **Modele pentru interfeţe utilizator**
  Structura UI (navigare, interacţiune, …)
  E.g. formcharts, diagrame ecran

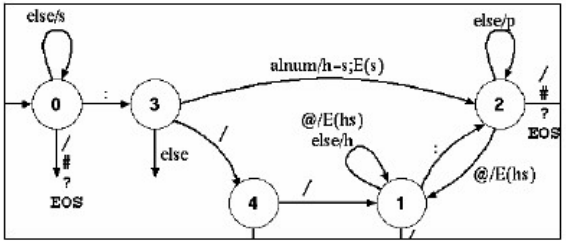




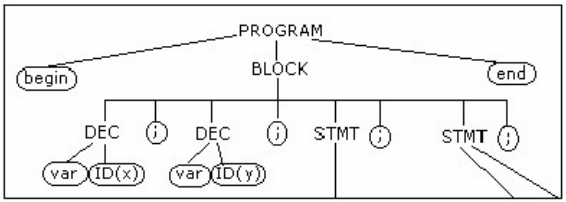


Mai multe modele

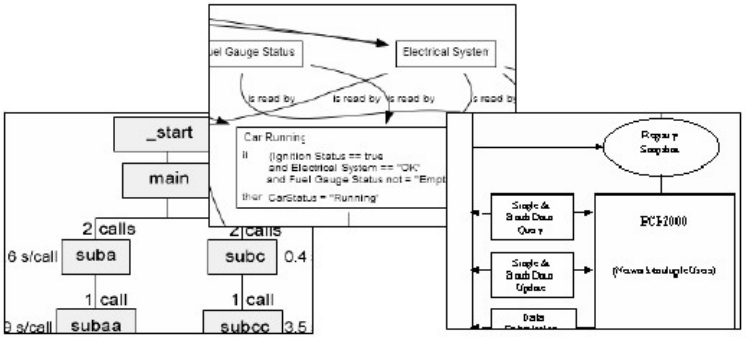
- Modele stare tranziţie
  Stări ale sistemului şi tranziţii între acestea
  E.g. Maşini cu stare pentru modelarea jocurilor



- Modele ale codului sursă
  Structura codului programului
  E.g. Arbori de sintaxă abstracţi (AST)



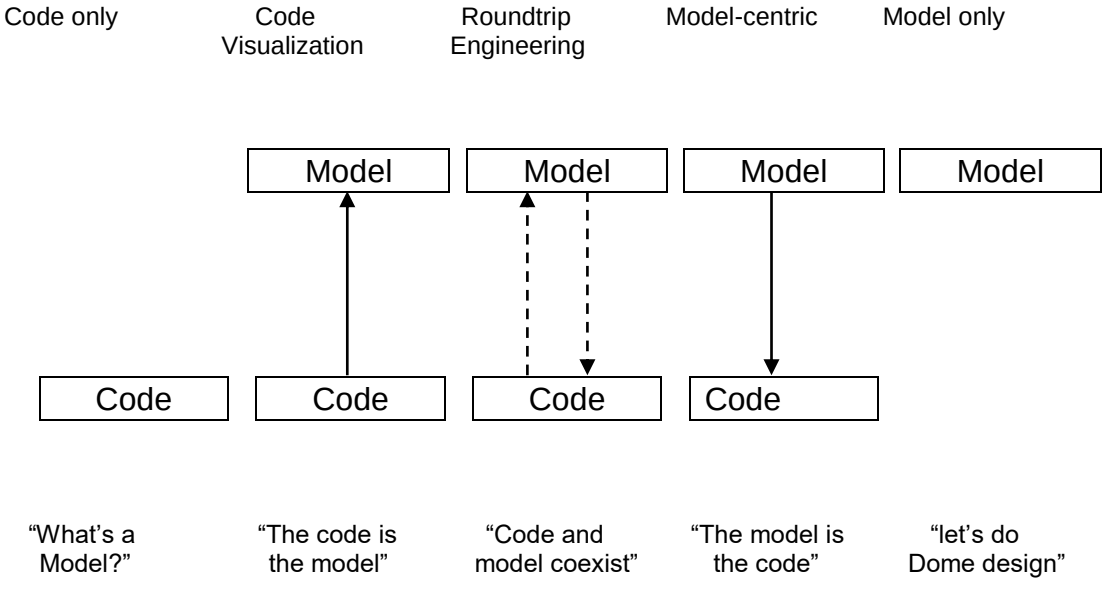
  Grafuri de apeluri, grafuri de dependinţe, diagrame de fluxuri de date
  & multe altele…



Modelare orientată obiect

- Punct de pornire: descrierea informală a cerinţelor
- Analiza OO:
  - Modelează invarianţi specifici domeniului unui sistem
  - Părţi stabile ce descriu concepte stabile
  - Ex: într-o universitate sunt întotdeauna studenţi, cursuri, profesori…
- Proiectarea OO:
  - Rafinează modelul de analiză cu părţi dependente de implementare
  - Ex: persistenţă, distribuţie, folosirea anumitor tehnologii sau componente

Moduri de sincronizare modele-cod sursă aplicate



Evoluţia sistemelor (1)

- Abordare code-only
  - sistemele sunt scrise direct într-un limbaj de programare
- Abordarea code visualization
  - după analiza problemei, codul e reprezentat grafic şi modificat
- Dezvoltarea round-trip engineering – RTE
  - Separarea modelelor de codul sursă
- Abordarea centrată pe modele (model centric approach)
  - modelele sistemului informatic sunt suficient de detaliate pentru a permite implementarea completă a acestuia
- Abordarea model-only
  - modelele sunt folosite strict la înţelegerea şi reprezentarea domeniului de studiat, a proceselor de afaceri, a analizei arhitecturii sistemului, etc.

Evoluţia sistemelor (2)

- Abordarea MDA
  - cea mai recentă abordare
  - asemănată mai mult cu o abordare în care codul este parţial sau complet generat pe baza mai multor modele, obţinute prin aplicarea de diferite limbaje standard de modelare
  - acesta este modul de dezvoltare de sisteme informatice care poate răspunde cu acurateţe cerinţelor beneficiarilor, oferind mai multă flexibilitate impusă de evoluţia acestora în timp
  - abordarea MDA permite dezvoltarea unui sistem IT pe baza standardelor existente deja, furnizând un cadru de interoperabilitate şi interconectare a sistemelor software diferite.

Practica modelării sistemelor informatice

- Modelele reprezintă o abstractizare aproximativă a elementelor reale ce urmează a fi realizate.
- Mai puţin la început pentru că aplicaţiile software erau mai simple, uşor de realizat, dar mai ales pentru că puteau fi modificate fără costuri suplimentare semnificative
  - exista tiparul imaginează-construieşte-modifică
- Astăzi sistemele informatice sunt din ce în ce mai complexe
  - se impune integrarea cu aplicaţii software deja existente
  - sunt utilizate permanent
  - se impune adaptarea lor la noile condiţii reale.

Ce modele sunt importante?

- Business Model
  - Vizualizarea proceselor de business
- System Architecture Model
  - Vizualizarea cerinţelor de sistem, structurii şi comportamentului
- Use Case Model
  - Vizualizarea cerinţelor funcţionale
- Analysis Model
  - “Ce” trebuie să realizeze sistemul pentru a realiza cerinţele funcţionale impuse
- User Experience Model
  - Vizualizarea interacţiunii utilizatorului cu sistemul
- Design Model
  - “Cum” realizează sistemul cerinţele funcţionale
- Data Model
  - Vizualizarea stocării persistente
- Implementation Model
  - Vizualizatea codului

Abordarea IBM

- Beneficiarii produselor software aşteaptă livrarea la timp şi de calitate a unui produs care să răspundă cerinţelor lor.
- Specialiştii firmei IBM au formulat patru caracteristici pe care trebuie să le îndeplinească în procesul de dezvoltare software, şi anume:

- dezvoltare iterativă;
- concentrare pe arhitectura sistemului informatic;
- asigurarea continuă a calităţii impuse de beneficiari şi gestionarea schimbărilor şi rezultatelor obţinute, cât şi a complexităţii sistemului IT;
- înţelegerea procesului de proiectare şi a riscurilor asociate.

Abordarea IBM

- Prin modelarea sistemelor informatice, dezvoltatorii software pot:

- crea modelele sistemului înaintea angajării de resurse adiţionale;
- proiecta aplicaţii software pornind de la cerinţe, dând siguranţă şi încredere în calitatea sistemului obţinut;
- aplica dezvoltarea iterativă în care modelele sau alte artefacte obţinute au un nivel ridicat de abstractizare, permiţând modificări rapide şi oricât de frecvente sunt necesare, funcţie de modificările mediului de lucru.

Reticenţa la modelare

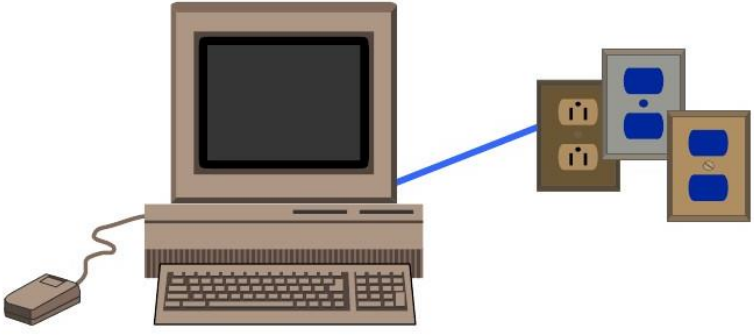
- Nu toţi dezvoltatorii software înţeleg necesitatea modelării software
  - Adesea sistemele informatice sunt simple, uşor de înţeles şi abia apoi devin din ce în ce mai complexe în mod natural.
  - În unele cazuri ei nu apelează la modelare pentru că pur şi simplu nu percep necesitatea acesteia decât mult mai târziu.
- Unii specialişti argumentează rezistenţa la modelarea software ca fiind o trăsatură de cultură, sau, pentru că aceasta presupune instrumente suplimentare, pregatire suplimentară, anumite costuri adiţionale, o durată mai mare de timp necesar de alocat şi eforturi suplimentare.

Avantajele modelării

- înţelegerea mai bună a problemei de rezolvat
- realizarea unui sistem informatic de calitate
- proiectarea şi construirea arhitecturii sistemului
- crearea unei vizualizări a codului sursă sau a altor forme de implementare a acestuia.

- Totuşi modelarea nu reprezintă ”totul sau nimic”, ea reprezintă doar o parte din procesul de dezvoltare software.

Viziunea OMG



The Global Information Appliance

Consensul modelării

- Eterogenitatea este pretutindeni
  - Nu există consens asupra platformelor hardware
  - Nu există consens asupra sistemelor de operare
  - Nu există consens asupra protocoalelor de reţea
  - Nu există consens asupra limbajelor de programare

- Totuşi trebuie să existe concens asupra modelelor, interfeţelor şi interoperabilităţii!

Model Driven Architecture

- Iniţiativa Model Driven Architecture (MDA™) a grupului OMG urmăreşte integrarea bunurilor deja existente în software-ul dezvoltat
- MDA ajută la integrarea mixului de soluţii deja existente şi oferă o arhitectură pentru a suporta orice schimbări neaşteptate viitoare
- Focusată pe integrarea aplicaţiilor de tip “legacy”
- Asigură integrarea facilă a aplicaţiilor COTS
- Modelele sunt testabile şi simulabile

Ce este MDA?

- Procesul de dezvoltare software nu mai seamănă nici pe departe cu procesul clasic “waterfall”, ci constă într-o permanenţă extindere şi rafinare a unei soluţii parţiale deja existente, căreia, după un număr de iteraţii, i se adaugă plus-valoare din punct de vedere al beneficiarului

- Def: O modalitatea de a specifica şi construi sisteme
  - Bazată pe modelarea folosind UML
  - Suportă între ciclul de dezvoltare: analiză, proiectarea, implementare, deployment, mentenanţă, evoluţie & integrarea cu sistemele viitoare
  - Interoperabilitate şi Portabilitate
  - Foloseşte standarde deschise
  - Costuri iniţiale scăzute şi maximizarea ROI

  - Se aplică direct în mixul pe care astăzi îl avem în faţă:
    - Programming language
    - Operating system
    - Network
    - Middleware

Categorii de instrumente MDA

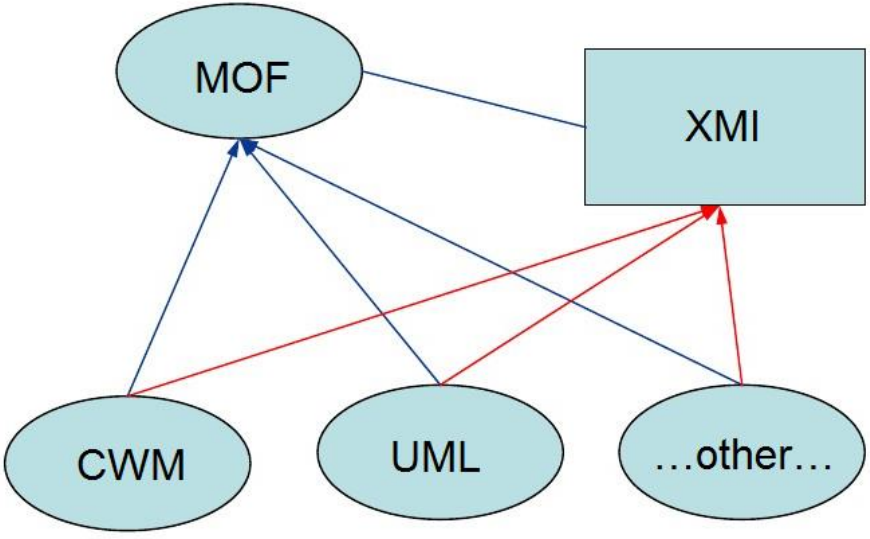
- Instrumentele MDA se împart în trei categorii:

- comerciale sau open source;
- parţial sau complet de implementare;
- generatoare de cod din modele sau executarea modelelor.

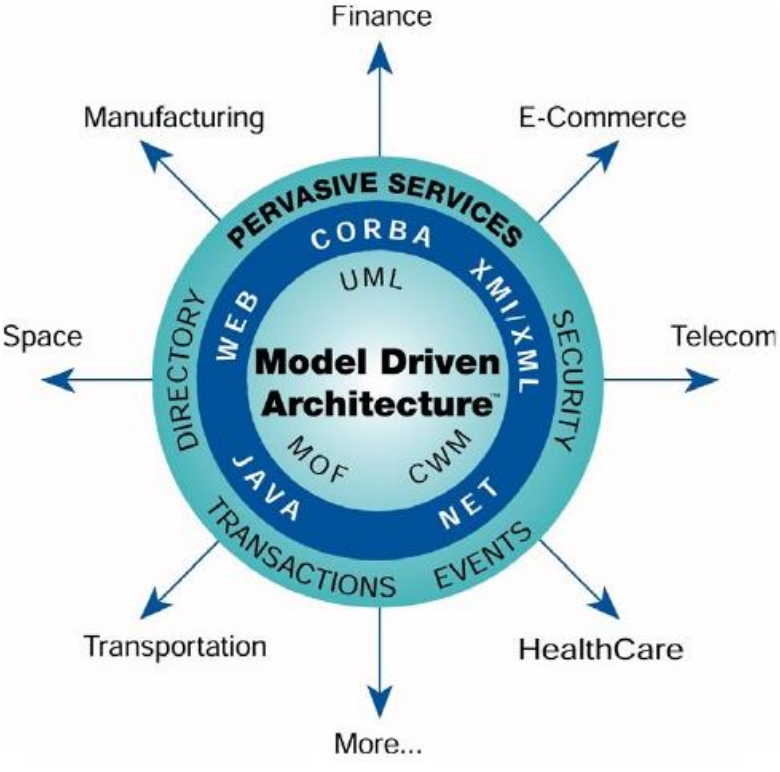
Suita de modelare a OMG

- Unified Modeling Language
  - UML™ rămâne singurul limbaj standardizat de modelare orientat obiect, cea mai bine cunoscută parte a standardului
- Common Warehouse Metamodel
  - CWM™, integrarea de iniţiative data warehousing
- Meta-Object Facility
  - MOF™, standard de integrare a metadatelor
  - Defineşte metadatele şi serviciile de metadate
- XML Metadata Interchange
  - XMI™, standardul XML-UML
  - Interoperabilitate a instrumentelor UML
  - Colecţie de reguli de mapare XML/MOF

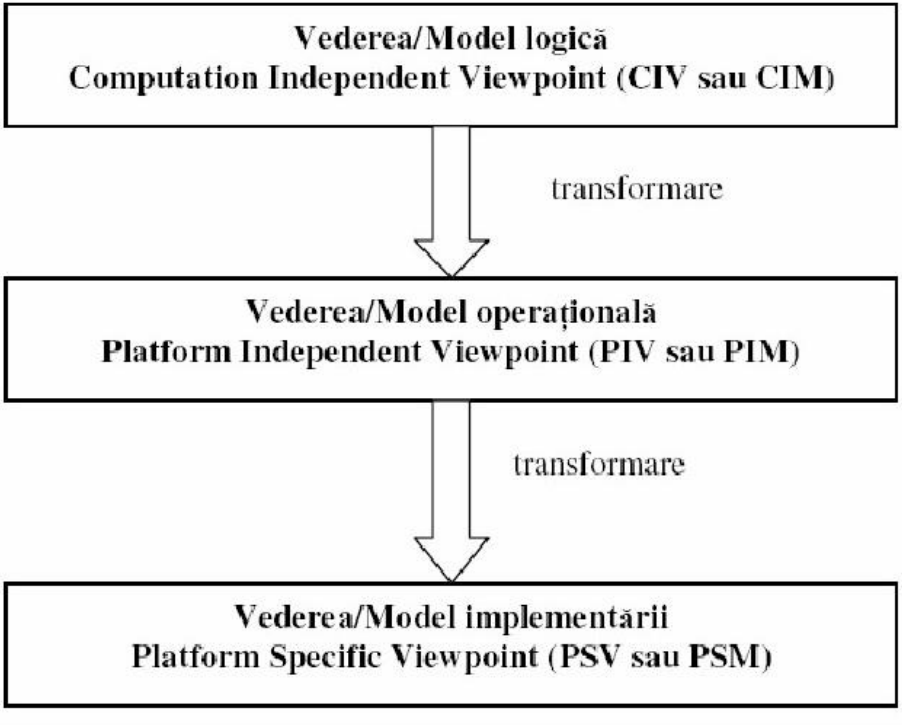
OMG Modeling Suite



Model Driven Architecture



3 vederi ale arhitecturii MDA



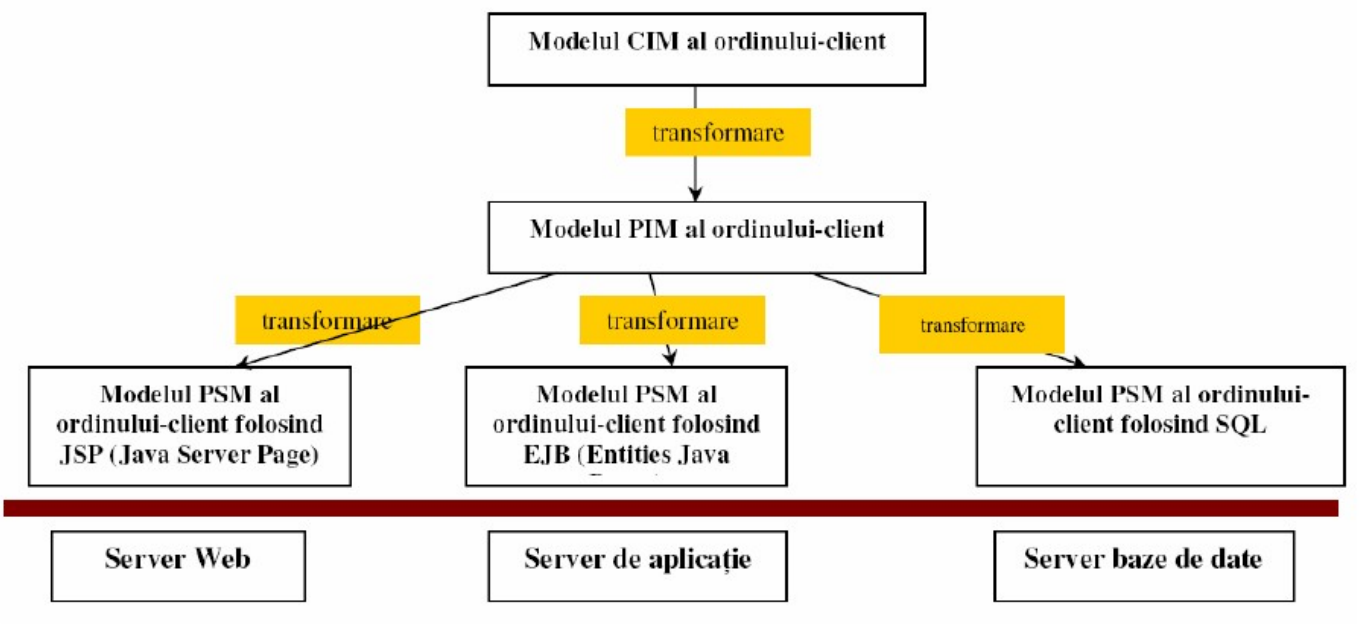
Vederi MDA

- Computation Independent Viewpoint – CIV
  - separă modelarea logică a sistemului informatic de specificaţiile de implementare;
- Platform Independent Viewpoint – PIV
  - se focalizează pe modelarea operaţională a sistemului, dar fără detalierea specificaţiilor de implementare;
- Platform Specific Viewpoint – PSV
  - conţine specificaţiile de implementare (platforma hardware, platforma software, produse middleware, tehnologii IT folosite).

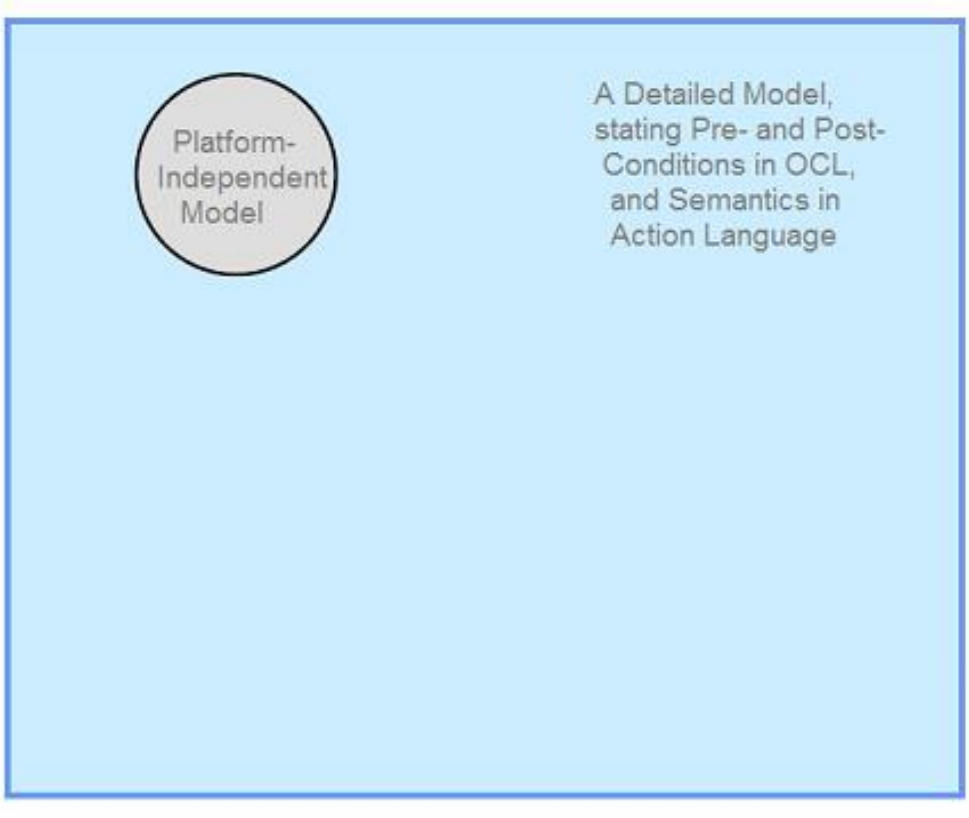
Exemplu (1)

- integrarea în sistemul informatic al unei firme a ordinelor de la clienţi transmise prin Interne
  - vederea CIM va consta din diagrame UML la nivel conceptual prin care se arată ce va face sistemul informatic
  - vederea PIM descrie funcţiile şi structura acestuia prin diagrame UML detaliate
  - mai multe vederi PSV

Exemplu (2)

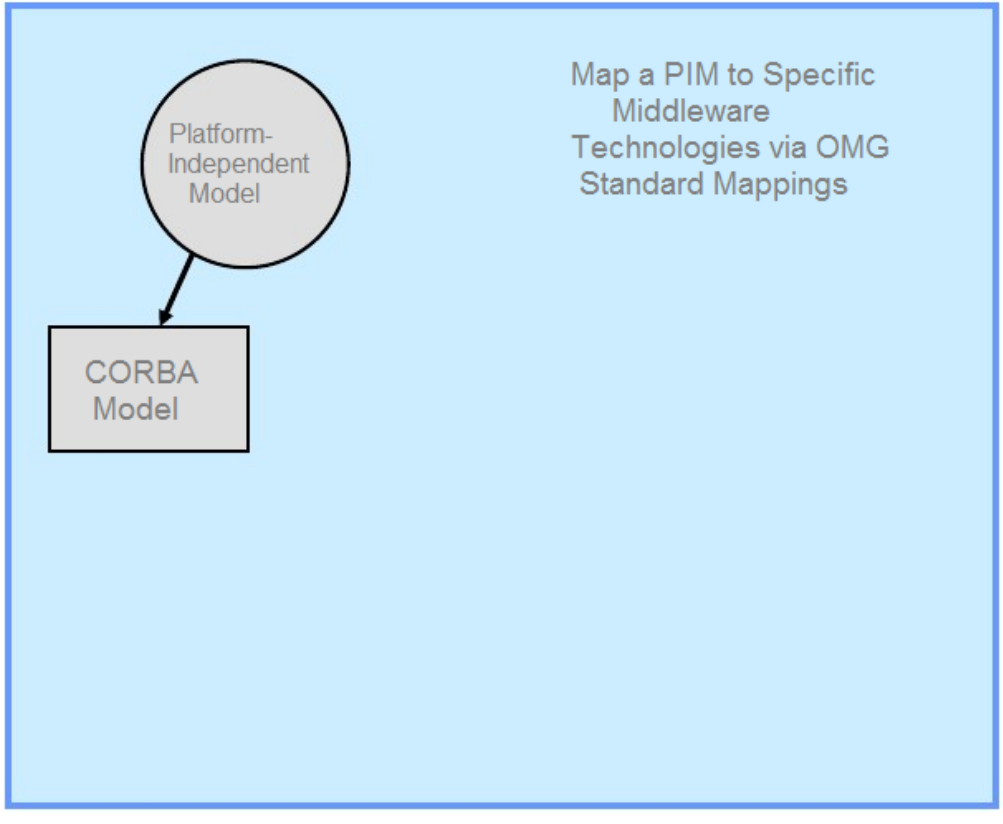

Modelul CIM al ordinului-client
transformare
Modelul PIM al ordinului-client
transformare
transformare
transformare
Modelul PSM al ordinului-client folosind JSP (Java Server Page)
Modelul PSM al ordinului-client folosind EJB (Entities Java
Modelul PSM al ordinului-client folosind SQL
Server Web
Server de aplicație
Server baze de date

Construirea unei aplicaţii MDA



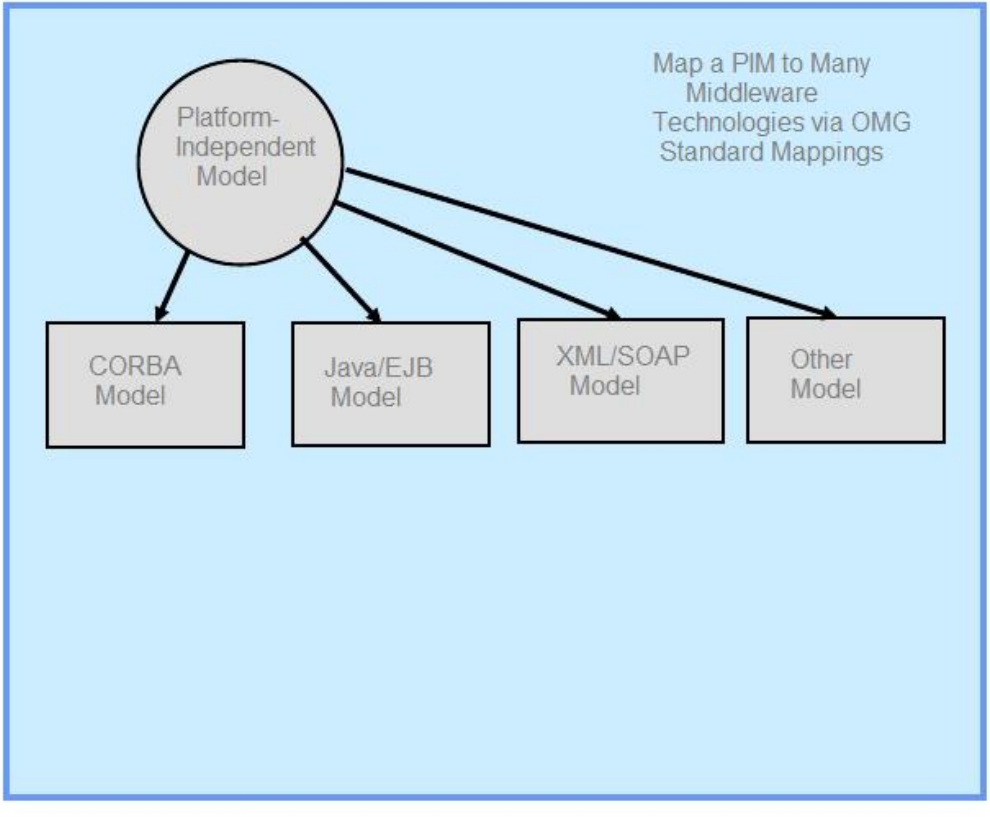
Start with a Platform-Independent Model (PIM) representing business functionality and behavior, undistorted by technology details.

Generarea modelului specific platformei



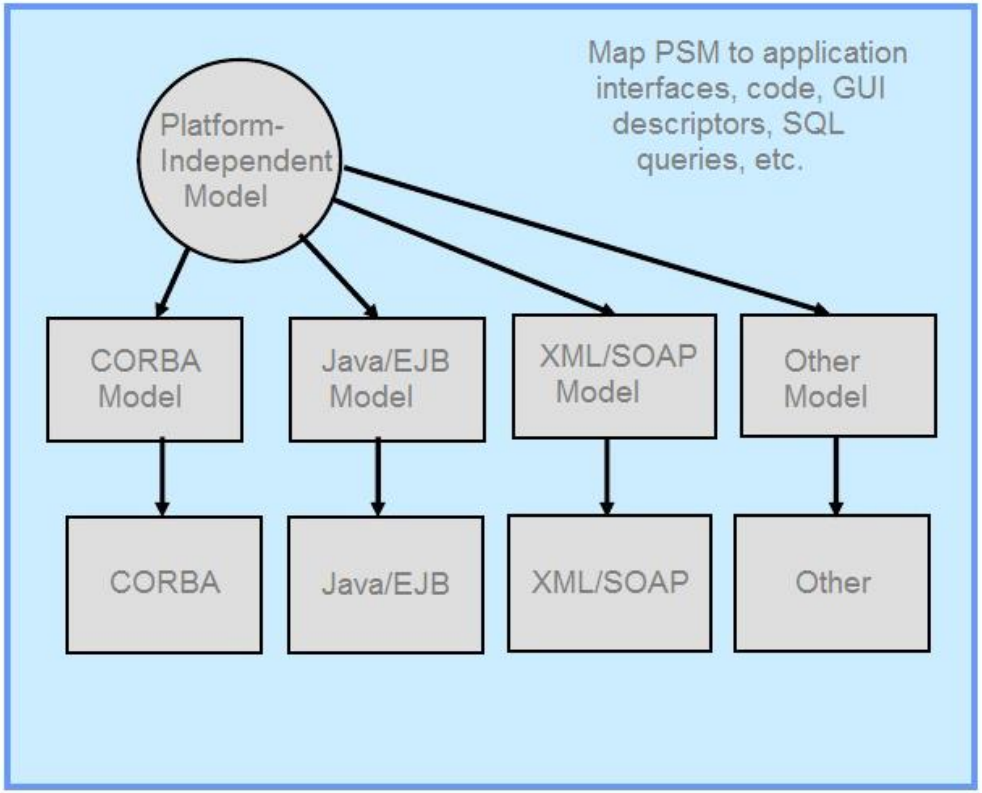
MDA tool applies a standard mapping to generate Platform-Specific Model (PSM) from the PIM. Code is partially automatic, partially hand-written.

Maparea pe multiple tehnologii de deployment



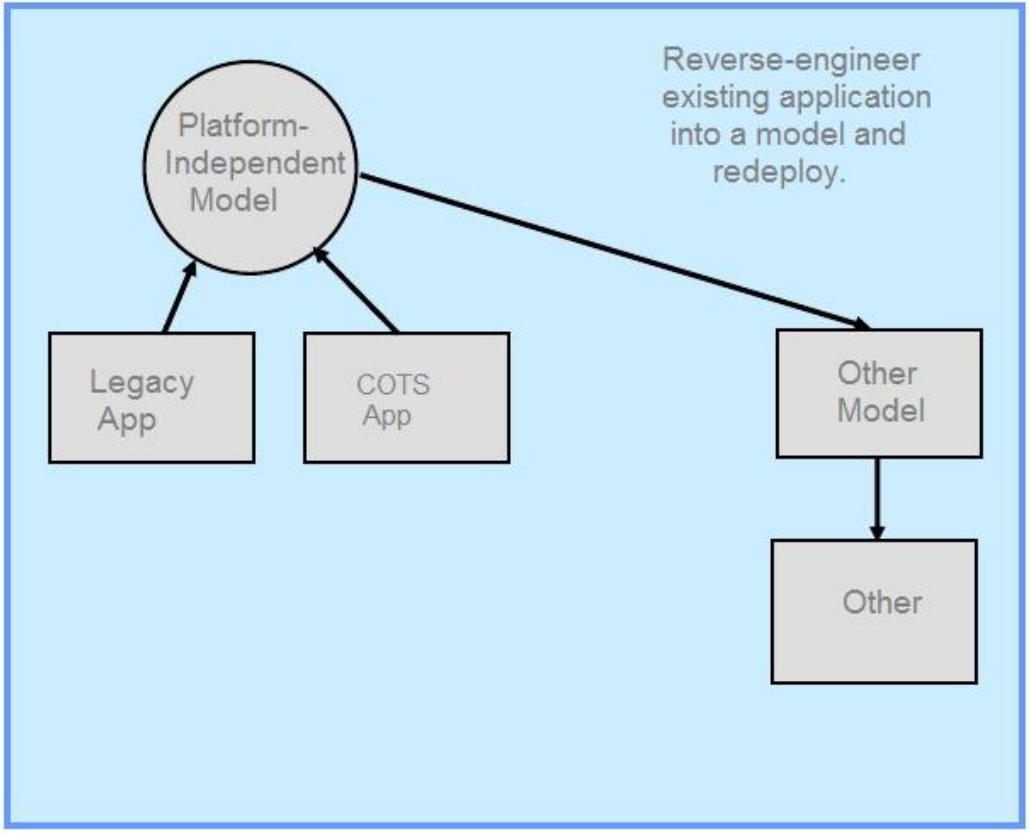
MDA tool applies an standard mapping to generate Platform-Specific Model (PSM) from the PIM. Code is partially automatic, partially hand-written.

Generarea implementărilor



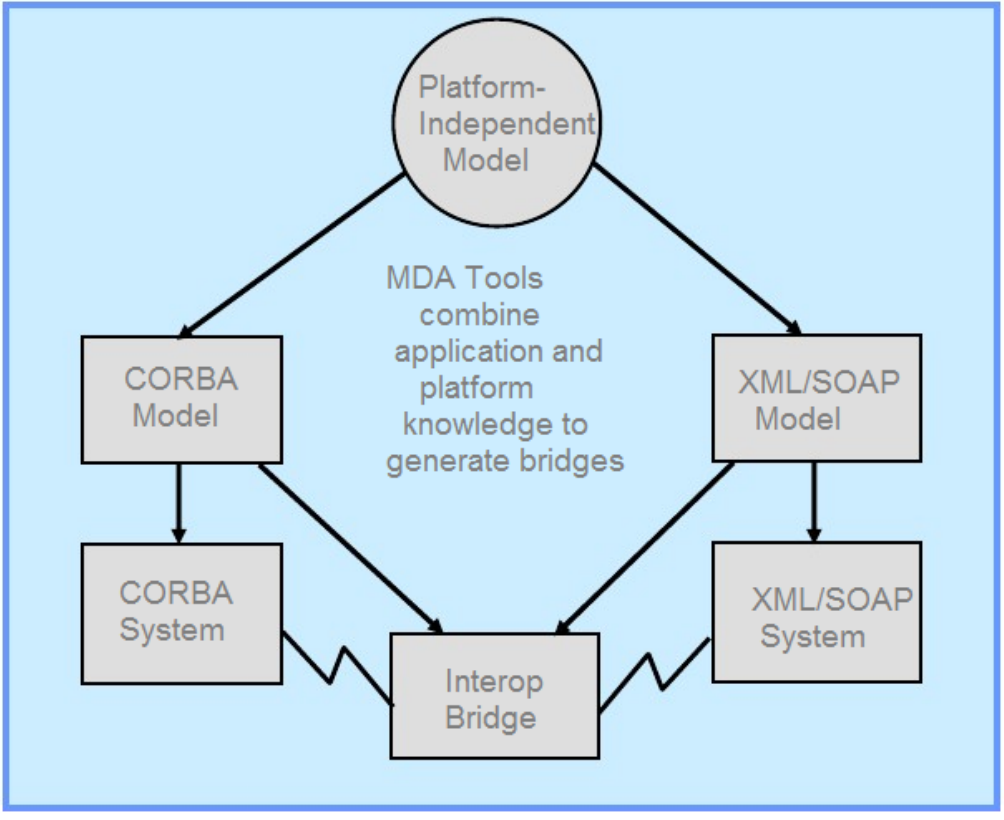
MDA Tool generates all or most of the implementation code for deployment technology selected by the developer.

Integrarea Legacy & COTS



MDA Tools for reverse engineering automate discovery of models for re-integration on new platforms.

Automating Bridges



Bridge generation is simplified by common application models, simplifying creation of integrated applications both within and across enterprises.

MDA în Standarde Industriale

- MDA promovează standarde ce sunt funcţionale independent de tehnologie
  - Aplicabil pentru deploymenturi mari & mici, noi aplicaţii, legacy şi COTS
  - Aplicabil pentru CORBA, DCOM, .Net, etc.
- MDA a fost adoptat de grupurile de standardizare OMG
- Grupuri din marketing văd valoarea adusă de abordara MDA:
  - Legacy Transformation
  - Financial Services
  - Healthcare
  - Life Sciences Research
  - Manufacturing
  - Space & Ground Systems
  - Telecommunications

MDA în Practică

- Several excellent proofs-of-concept:
  - Wells Fargo (an architecture that has already been resilient through a decade of change)
  - Lockheed Martin Aeronautics
  - GCPR in US government
  - Deutsche Bank Bauspar
  - Defense Information Systems
  - Merrill Lynch
  - Österreichische Bundesbahn
  - Thales Training & Simulation
  - Zuercher Kantonal Bank
  - CGI
  - Chubb and Son

Mai multe informaţii

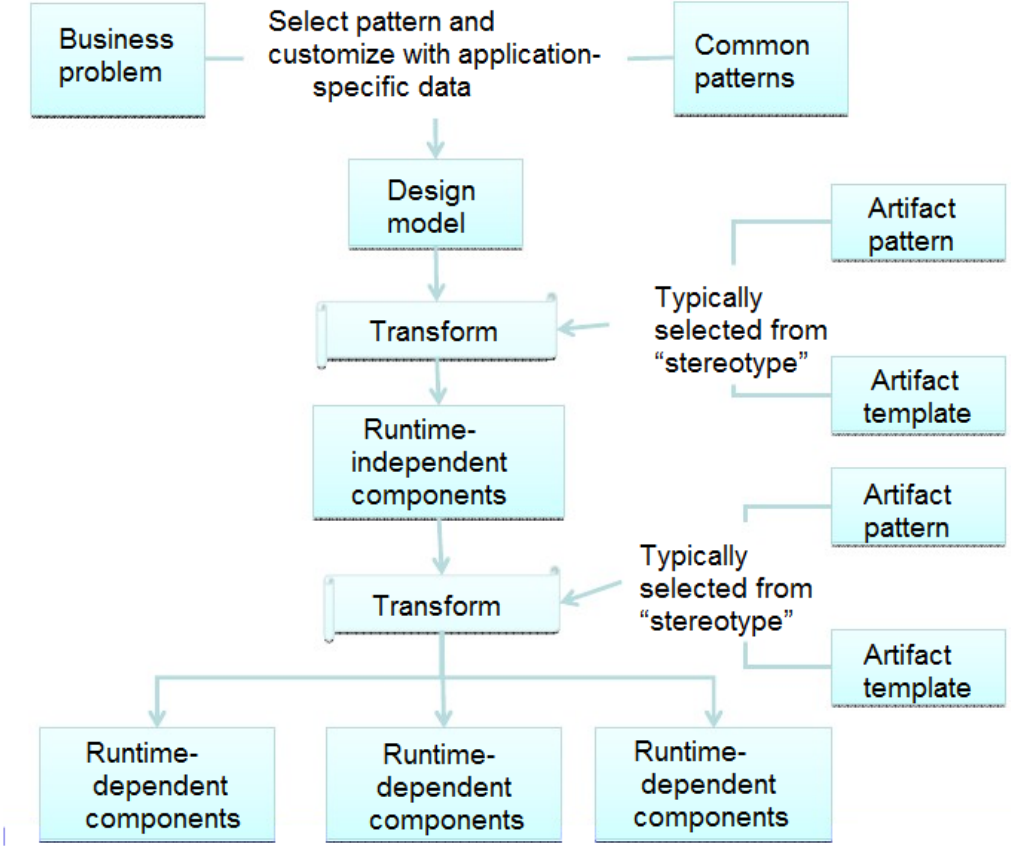
- MDA Information Page
  - http://www.omg.org/mda/
- OMG General Information
  - http://www.omg.org/

http://www.omg.org/~soley/ooi s03.ppt

MDD

- Model Driven Development (MDD) = dezvoltare dirijată prin modele
  - metodă de proiectare a sistemelor informatice aplicabilă în cadrul metodologiei MDA.
- Se construieşte un ansamblu de modele ale sistemului de analizat cât şi ale noului sistem, pe baza cărora se generează alte modele sau codul sursă al sistemului.
- Totul se centrează pe transformarea modelelor sistemului de realizat şi generare de cod sursă.
- Această metodă necesită un mediu integrat de dezvoltare (IDE) care să suporte:
  - limbajul UML, şabloane, transformarea modelelor UML şi generare de cod sursă.
- Rational Software Architect (RSA) este un astfel de instrument integrat de proiectare şi dezvoltare de sisteme informatice.

Transformarea problemă reală – soluţie IT aplicând metoda MDD



Şabloane

- Şabloanele sunt modalitati de descriere a motivatiilor, a ceea ce se va întâmpla si de ce.
- Motivele studierii si aplicarii acestora sunt:
  - reutilizarea,
  - stabilirea unei terminologii comune ce îmbunatateste comunicarea echipei software,
  - trecerea la o perspectiva de nivel înalt asupra problemei de rezolvat,
  - îmbunatatirea posibilitatii de mdificare a codului sursa,
  - facilitarea adoptarii unor alternatice de proiectare mai bune,
  - ofera alternative la ierarhiile laborioase de clase pentru probleme complexe cât si posibilitatea unei proiectari mai bune si nu numai a uneia functionale.

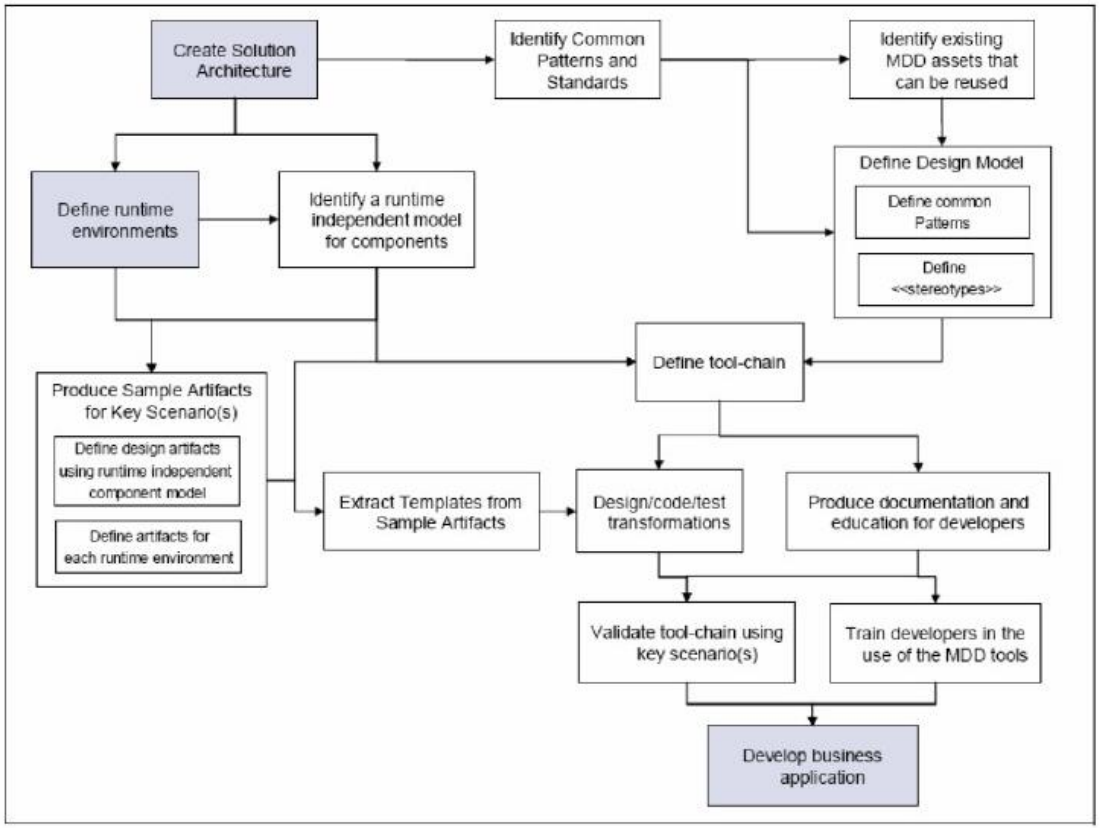
Şabloane

- Integrarea şabloane-MDD
  - creşte productivitatea procesului de dezvoltare de aplicaţii soft
  - se îmbunătăţeşte calitatea sistemului livrat şi întreţinerea acestuia,
  - reutilizarea componentelor sistemului informatic,
  - reducerea costurilor şi a riscurilor unui proiect de investitii,
  - creşte flexibilitatea sistemului dezvoltat,
  - se îmbunătăţeşte comunicarea în cadrul echipei de dezvoltare cât şi dintre aceasta şi beneficiarii sistemului informatic.

- Metoda MDD se fundamentează pe abstractizare
  - permite realizarea vederii conceptuale a sistemului informatic, centrarea pe funcţiunile sistemului fără specificaţii de implementare.

- Este recunoscut faptul că este mai rapidă crearea modelelor unui sistem informatic decât scrierea de cod-sursă, motiv pentru care se agrează această metodă care permite lucrul la un nivel ridicat de abstractizare, permite includerea specificaţiilor de proiectare în modele, urmat de generarea codului sursă prin aplicarea de transformări automate între modele.

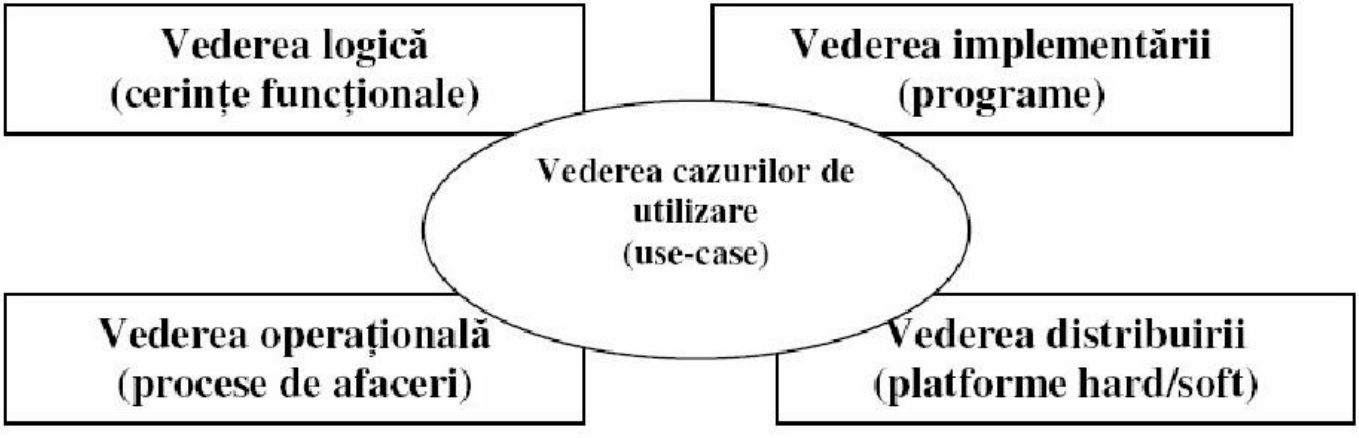
Etapele de realizare a sistemelor informatice în cadrul metodei MDD



Etape MDD

- Primele etape sunt de creare a arhitecturii sistemului şi de definire a mediului de implementare (platforme soft/hard, tehnologii IT).
- Pe baza arhitecturii sistemului informatic sunt identificate şabloanele de analiză ce se pot aplica, cât şi a modulelor de aplicaţii dezvoltate în alte proiecte MDD.
- Pe baza acestora se creează modelul de proiectare al sistemului şi modelul de implementare pe componente, pe baza căruia se produc un număr de artefacte iniţiale ce vor reprezenta scheme pentru următoarele transformări MDD.
- Se definesc instrumentele ce vor fi utilizate şi o planificare a întregului proiect de dezvoltare a sistemului informatic.
- Sunt generate modele de programe (template-uri) pe baza artefactelor obţinute anterior,care sunt testate, evaluate, sunt create module funcţionale pentru care se furnizează şi documentaţia aferentă.
- În acest moment se poate trece la dezvoltarea aplicaţiilor, precedată de instruirea în prealabil a echipei de dezvoltare.

Perspective arhitecturale MDD

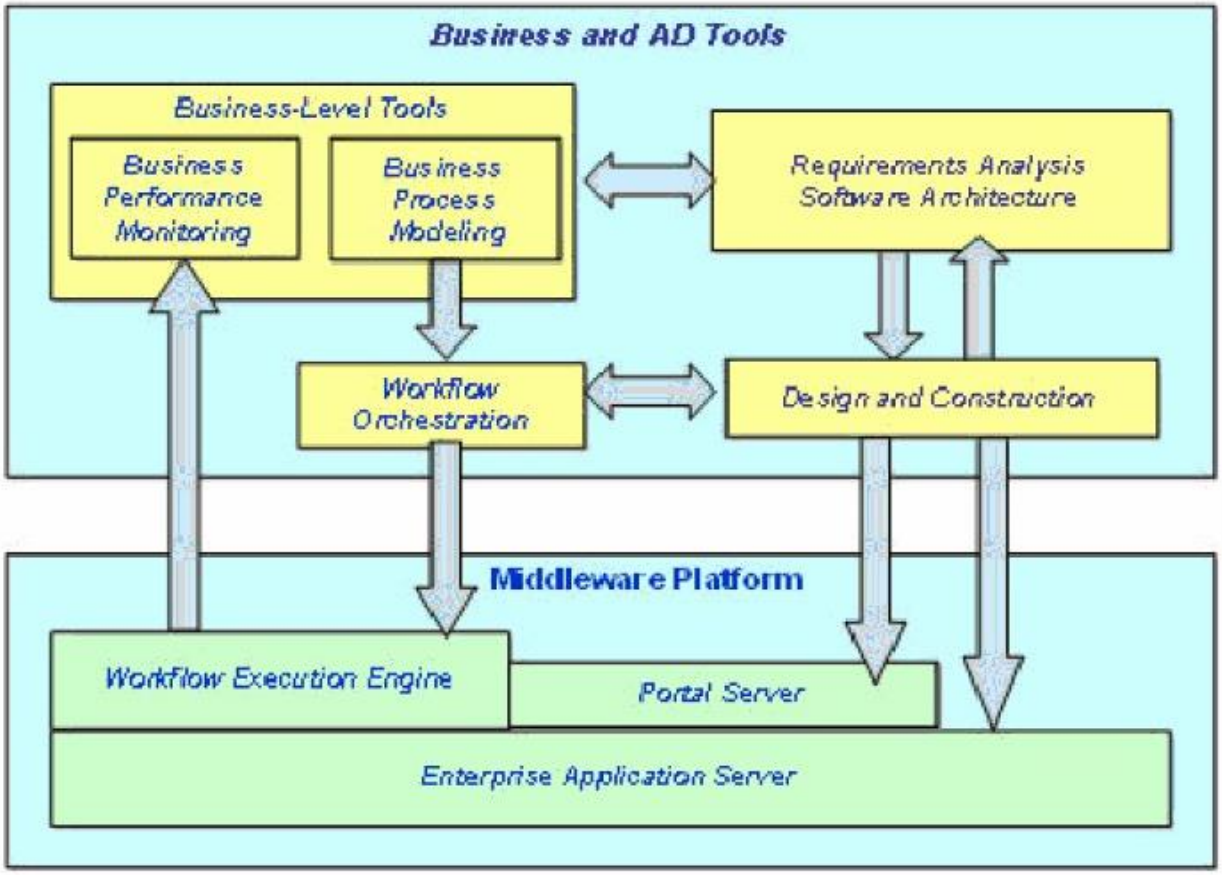


- vederea implementării - corespunde structurării programelor ce formează sistemul în componente
- vederea logică - descrie cerinţele funcţionale ale sistemului
- vederea distribuirii - defineşte aspectul spaţial al sistemului (echipamente hardware, noduri de retea)-
- vederea operaţională sau a proceselor, ce corespunde structurii de exploatare a programelor şi componentelor executabile.

Analiza comparativă a procesului tradiţional de dezvoltare a sistemelor informatice şi cel bazat pe metoda MDD

Dezvoltarea tradiţională a sistemelor informatice	Dezvoltarea sistemelor bazată pe metoda MDD
Centrat pe codificare şi testare	Centrat pe analiză şi proiectare, modelare
Codificarea manuală a programelor	Generarea automată a codului
Generarea manuală a documentaţiei	Generarea automată a documentaţiei
Testarea software-ului continuu	Validarea automată
Specificaţiile sunt bazate pe hârtie	Prototipizare interactivă rapidă

Noua viziune privind instrumentele de proiectare şi realizare de soluţii IT



Componente în noua viziune

- Platforma middleware – reprezintă o platformă de execuţie (runtime) formată din aplicaţii server robuste capabile să administreze soluţii IT dezvoltate pentru platforme software eterogene sau soluţii IT noi bazate pe standarde open;
- Model de execuţie – interfeţele platformei middleware definesc un model de programare ce trebuie înţeles de arhitecţii IT. Orice utilizator al platformei middleware trebuie să aibă deja modelul conceptual al soluţiei IT şi proiectarea fizică pe componente executabile;
- Fluxurile de lucru – reprezintă procesele de afaceri cât şi relaţiile dintre acestea şi posibilităţi de configurare;
- Instrumente de modelare a proceselor de afaceri – includ instrumente de modelare a acestor procese cât şi posibilitatea monitorizării acestora prin simularea impactului asupra evenimentelor declanşate în platforma middleware;
- Arhitectura sistemului prin componente – include instrumente care să permită gestionarea de aplicaţii moştenite, aplicaţii noi dezvoltate şi asamblarea lor.

Rational Software Architect

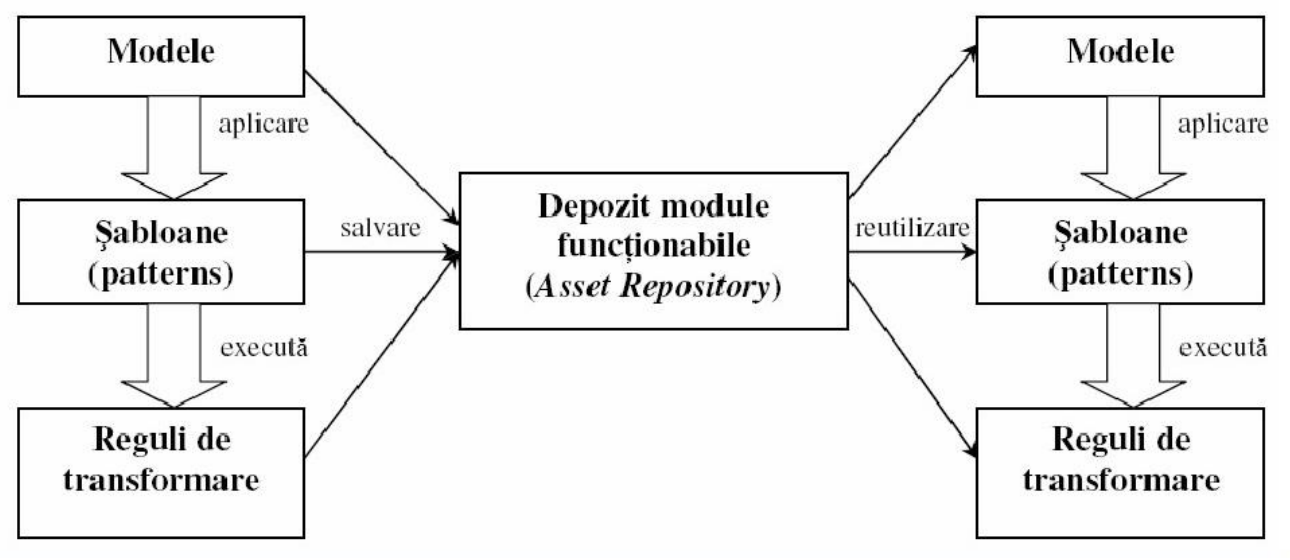
- octombrie 2004 - firma IBM a lansat platforma IBM Software Development Platform introducând o nouă generaţie de instrumente ce permit aplicarea metodei MDD.
  - Un loc central îl ocupă instrumentul integrat Rational Software Architect (RSA) ce suportă această metodă bazată pe limbajul UML.
- Instrument CASE ce se bazează pe metodologia MDA şi oferă suport în dezvoltarea aplicaţiilor Web statice sau dinamice ce pot rula pe platforma J2EE, include capabilităţi JSF, de utilizare a şabloanele (depozitul RAS – Reusable Asset Specification).

Obiectivele RSA

- separarea proiectării de implementare
- reducerea timpului şi costului de proiectare şi dezvoltare a sistemelor informatice
- simplificarea şi îmbunătăţirea procesului de testare
- integrarea activităţilor de proiectare şi dezvoltare
- standardizarea acestui proces
- creşterea calităţii sistemului realizat
- realizarea automată a unei documentaţii de calitate
- reutilizarea modulelor aplicaţiilor şi a documentaţiei
- specificarea corectă şi completă a cerinţelor sistemului
- Simplitate
- flexibilitate.

Componentele mediului RSA

- La arhitectura mediului integrat RSA se adaugă componente specifice aplicării şabloanelor

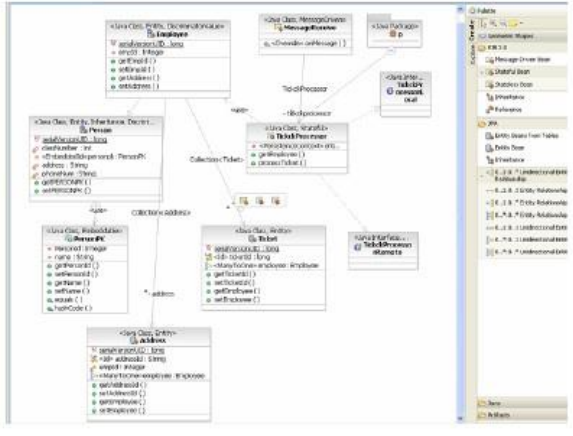


IBM Rational Software Architect

An integrated platform for innovation and collaboration

Best of breed, comprehensive modeling tools that facilitate communication and collaboration

- With the power of abstraction, automation and simplification
- DoDAF
- UML Profile-based Integrated Architecture
- Profile for Software Services
- Leveraging Jazz platform integrated with Rational Team Concert



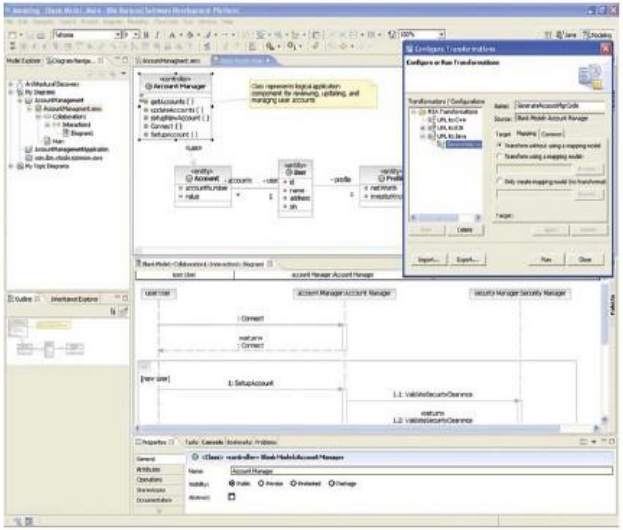
Exploit the latest in modeling language technology and leverage an open and extensible modeling platform

Simplify and unify Java and C++ design and development by integrating with other facets of the lifecycle such as:

- Rational Data Architect, Rational Requisite Pro
- Rational Asset Manager, Rational Team Concert
- Rational ClearCase, Rational ClearQuest
- Telelogic Synergy and Change

What's New: Rational Software Architect

- Custom modeling environment for your business
  - Modeling with Domain Specific Language (DSL)
  - Work with reduced subset of UML
- Rational Deployment Architecture Platform
  - Rich tools for deployment architecture definition
  - Verification tools for deployment architecture
- Enhanced transformations and visualizations
  - Extensive Java and C++ support
  - Increased visibility into existing source code
- Integrations
  - Rational Asset Manager
  - Rational Team Concert on Jazz
  - Telelogic Change and Synergy



Getting Started: User Assistance

User Assistance model to enable users of all skill levels

Leverages Product Tours to assist with the discoverability of capabilities

Tutorial Gallery leverages tutorials as learning aids

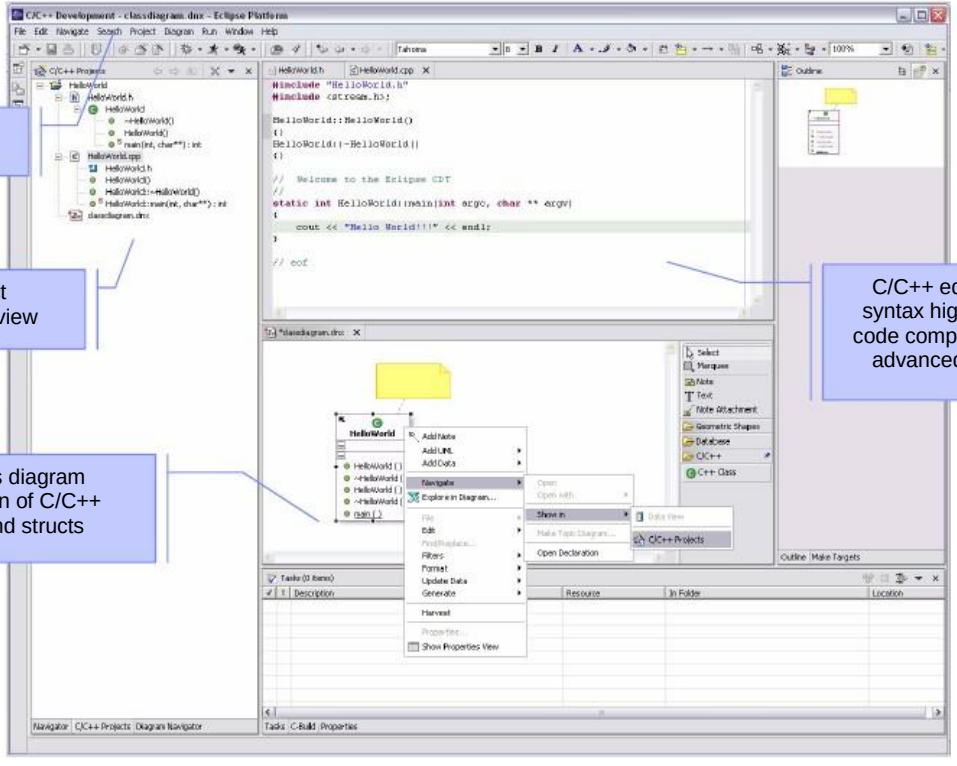
- “Watch and Learn”
- “Play and Learn”
- “Do and Learn”

Samples gallery provides completed assets for reference purposes

- Showcase
- Application
- Technology

All user assistance can be launched from a Welcome perspective

Key Feature: C/C++ Development Environment



Key Feature: C/C++ Development Environment

Editing and Navigation

- C/C++ Syntax Highlighting, Outline View
- C++ Class Browser (Hierarchy View)
- C/C++ Search
- C/C++ Content Assist

Project Import

- Automated assistance in setting up CDT for search and content assist.

UML C/C++ Code Editor

Debug

- GDB Integrated
- Extensible Debug Interface

Build

- Standard Make for projects with existing build infrastructure
- Managed Build
  - Automatic makefile generation
  - GNU tools supported out of box
  - Managed build is extensible, build tools can be plugged-in and build tools options selectable

Meets Internationalization and Accessibility requirements

Extensibility

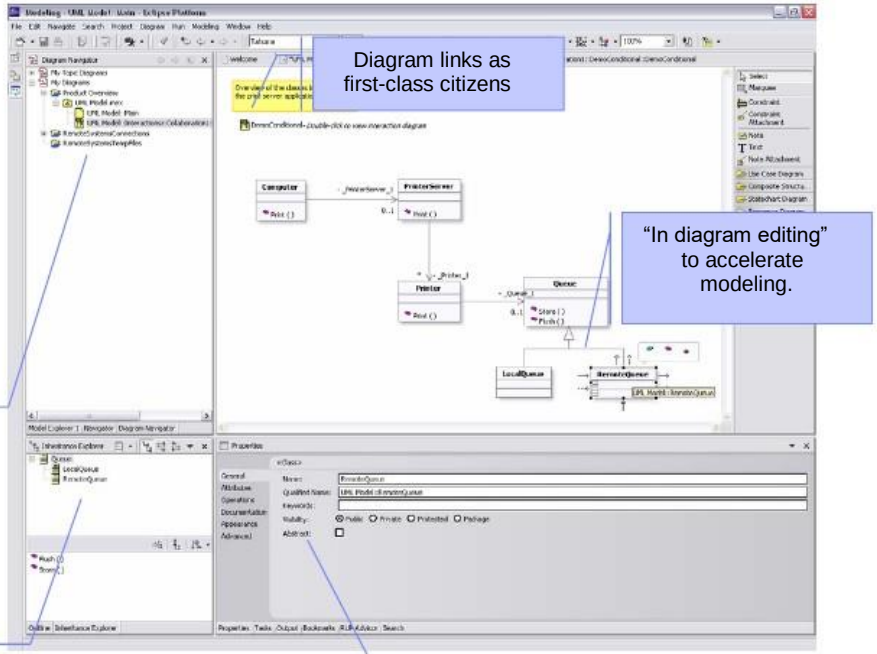
- Provides extension points for managed build, debuggers, …

Key Feature: Modeling assistance

Simplify the capture of UML models during Analysis and Design

Make modeling more accessible to a broader audience

New custom views improve the editing experience



Key Feature: Patterns

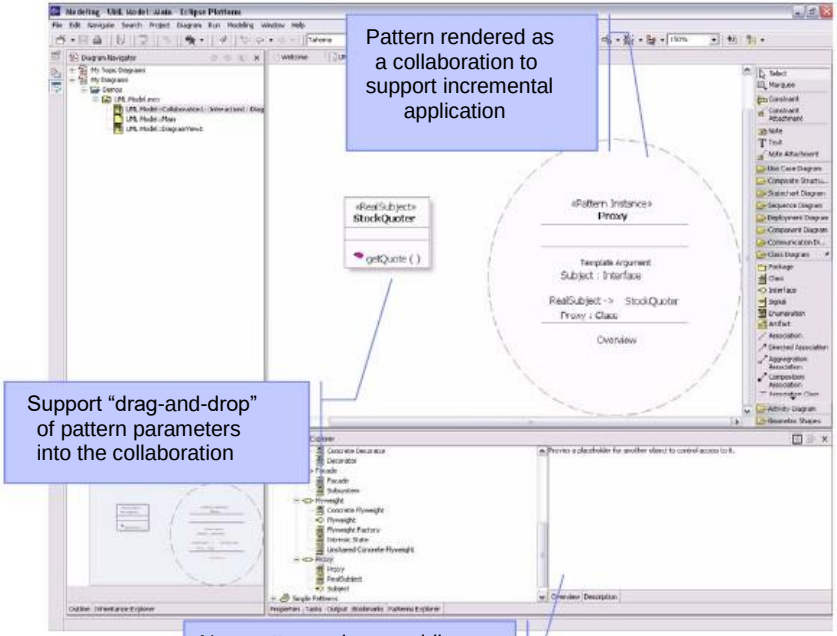
Applying Patterns is very simple

- Evolution of pattern experience based on lessons learned

Pattern-authoring provides greater flexibility using Open API

All Gang of Four design patterns provided

Additional patterns provided via RAS repository on IBM developerWorks



Key Feature: Transformations

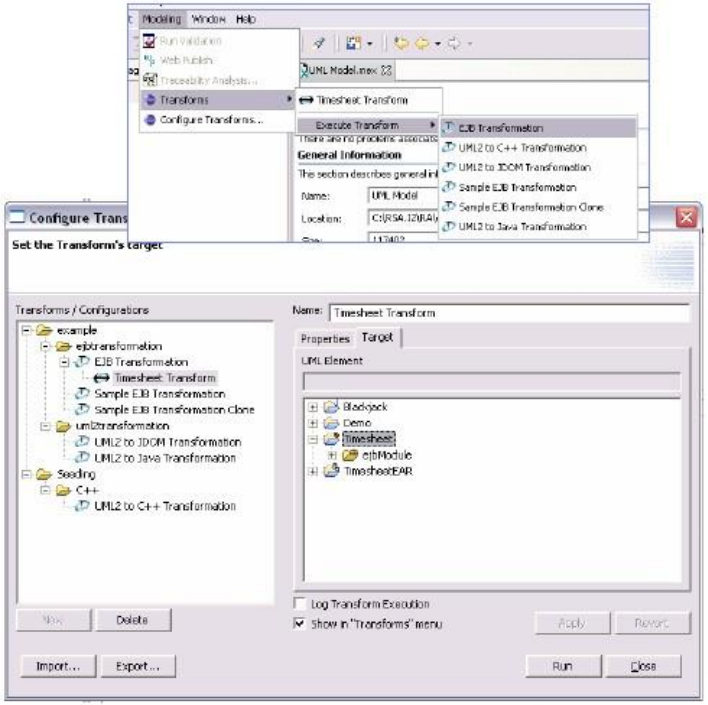
Transformations are optimal for “batch” style computationally intensive operations

- Model-to-model
- Model-to-code
- Code-to-model

Out-of-the box code transforms

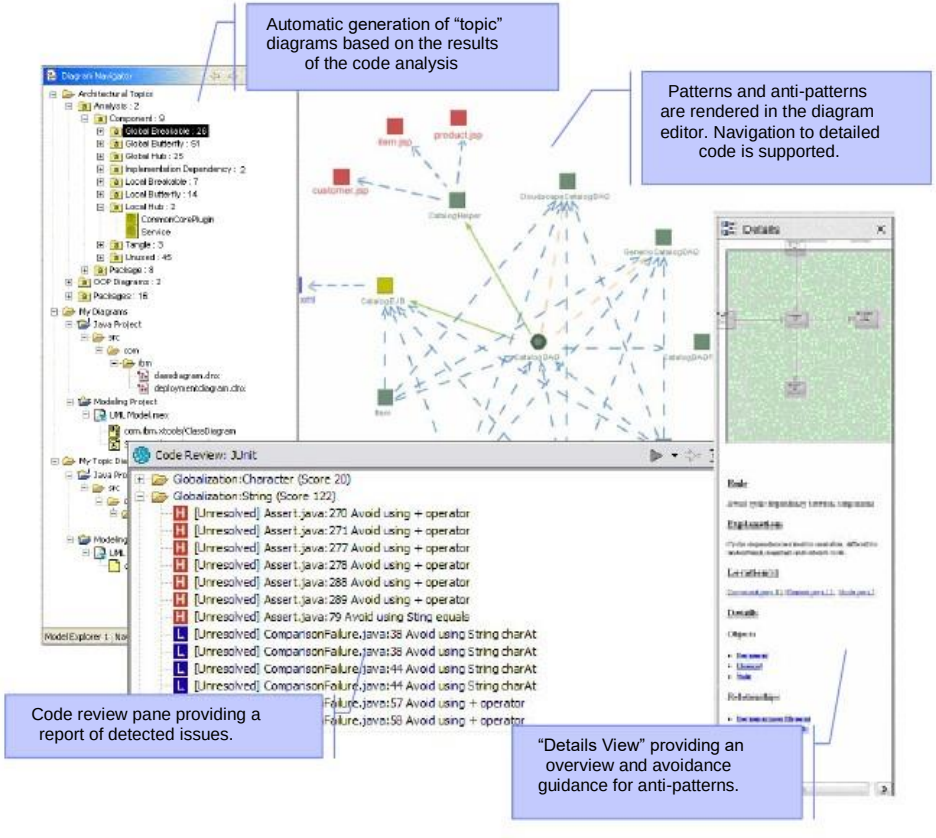
- UML-to-Java/JSE
- UML-to-C++
- UML-to-CORBA IDL
- Plus sample model-to-model transforms

Transformations may be updated via RAS repository hosted on IBM developerWorks



Key Feature: Architectureal Analysis, Discovery and Control

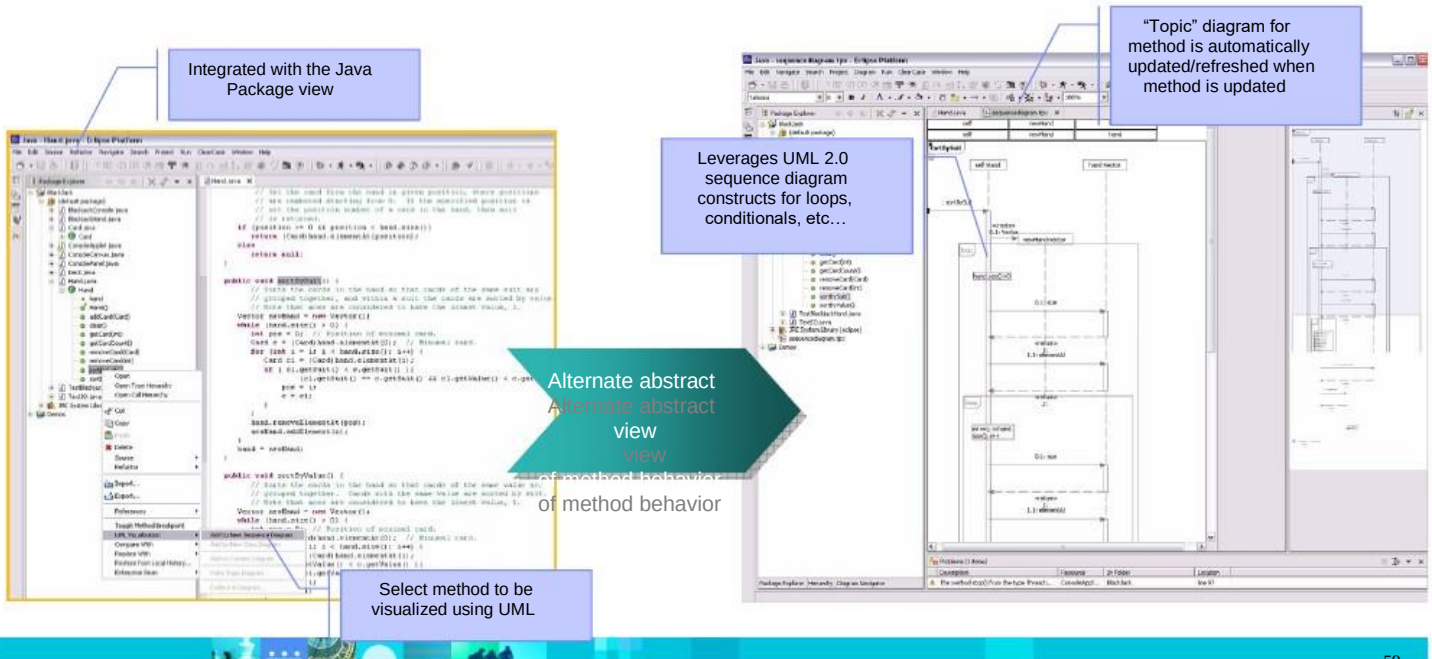
- Architecture discovery for J2SE
  - High-level software visualization
- Application architecture is reflected in the running code
  - Analyzing code can help assess its maintainability
- Govern the architecture with the assistance of rules
  - Template-based rule authoring
- Anti-pattern and pattern detection
  - Detection of cyclic dependencies, hubs, breakable, etc.
  - Wizard assisted automated quick-fix



Key Feature: Visualize method bodies

Facilitates understanding and application's behavior by providing visualization of detailed code

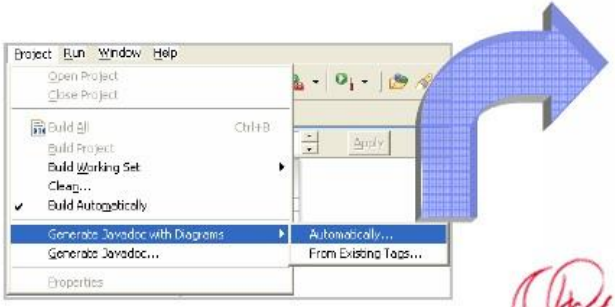
Diagrams can be integrated in Javadoc reports

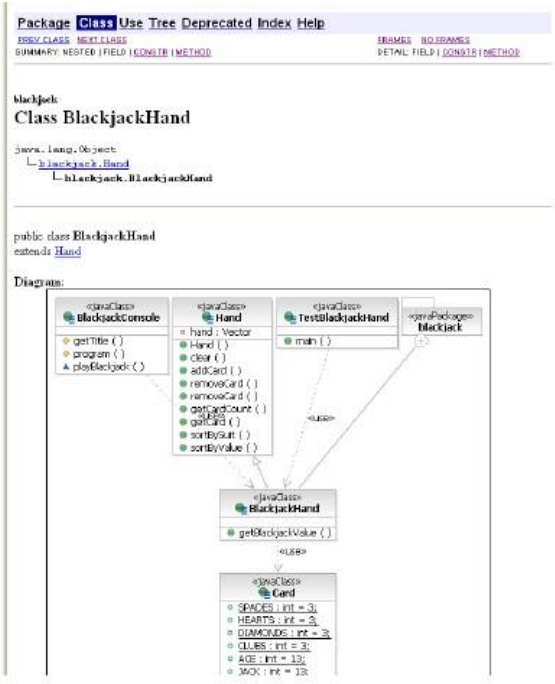


UML Enhancements: JavaDoc with Embedded UML Diagrams

Produce enriched JavaDoc

- UML diagrams right on the pages
- Completely integrated with hyperlinks





UML Enhancements: Interaction modeling

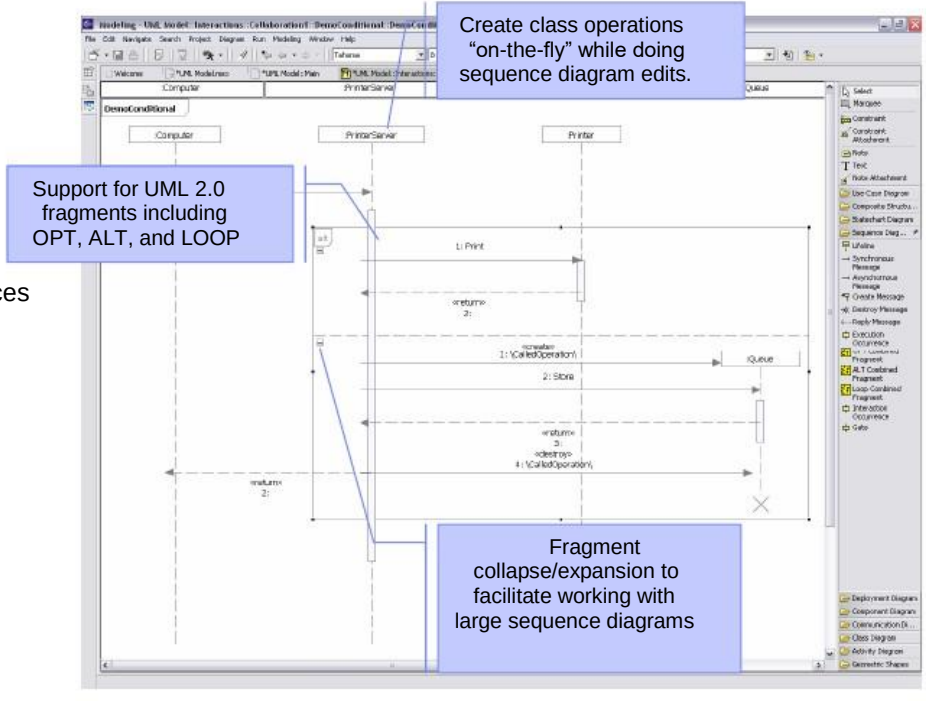
Interactions are expressed more effectively using UML 2.0 constructs

- Support specification of test scenarios
  - Loop, alt, opt
  - Interaction fragment references

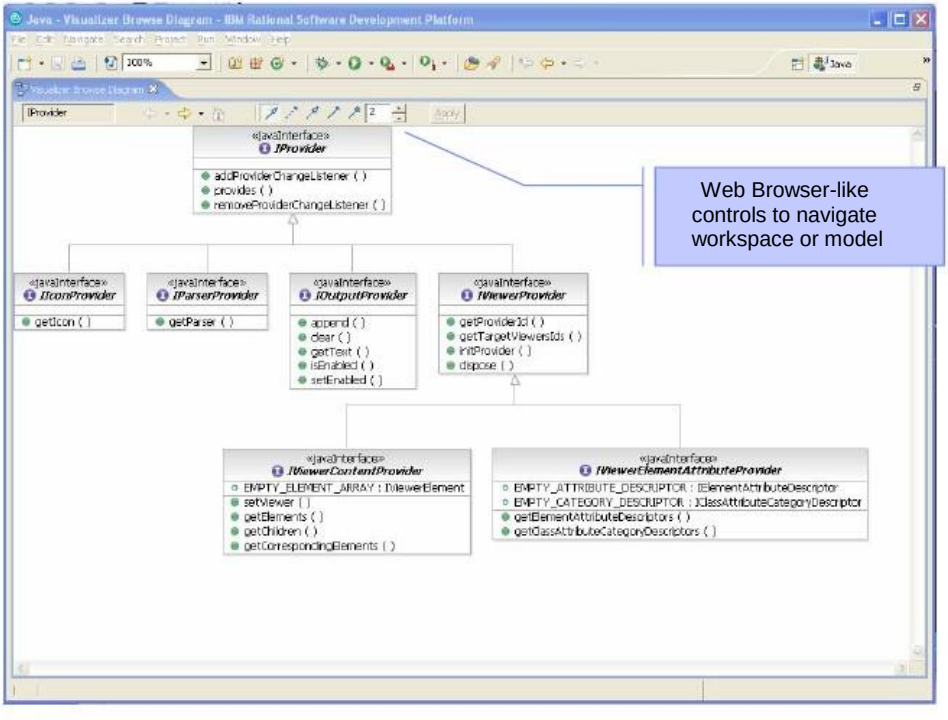
Interactions can be rendered as either sequence or communication diagrams

Sequence diagram editing improvements

- Ordering and reordering



UML Enhancements: Browse Diagrams



Enables users to understand and discover models and applications without having to create or maintain diagrams

Team: RequisitePro integration

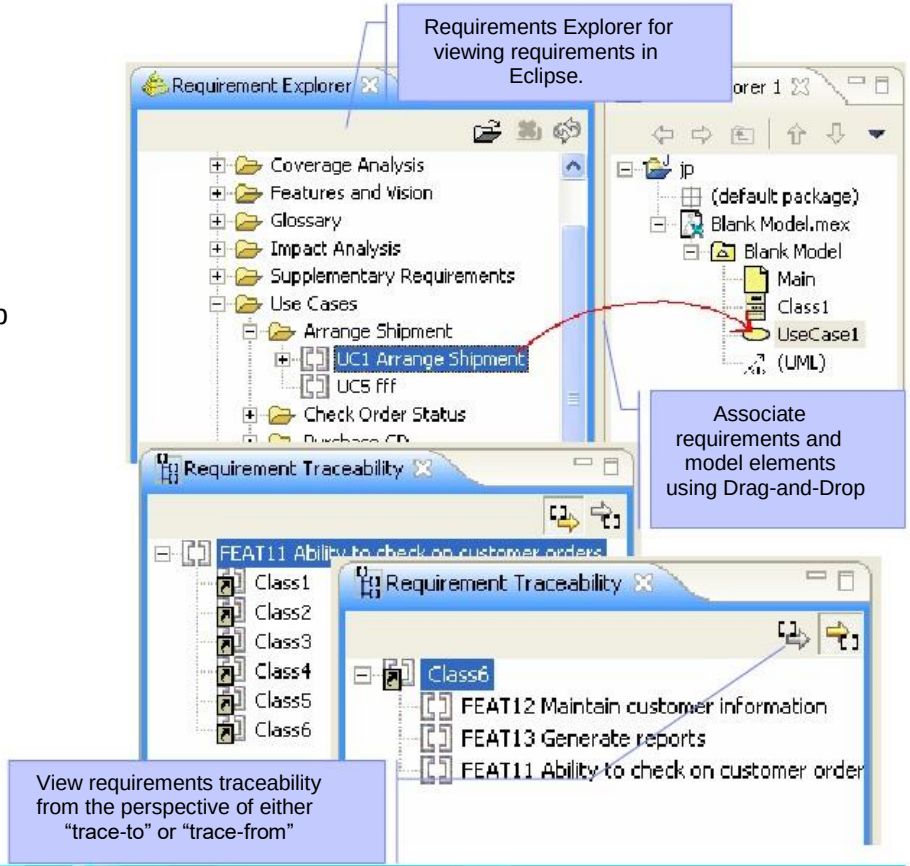
Open and browse multiple RequisitePro projects

See requirements, packages, and views

Associate requirements with model elements via drag and drop

Create model elements from requirements

Customizable synchronization



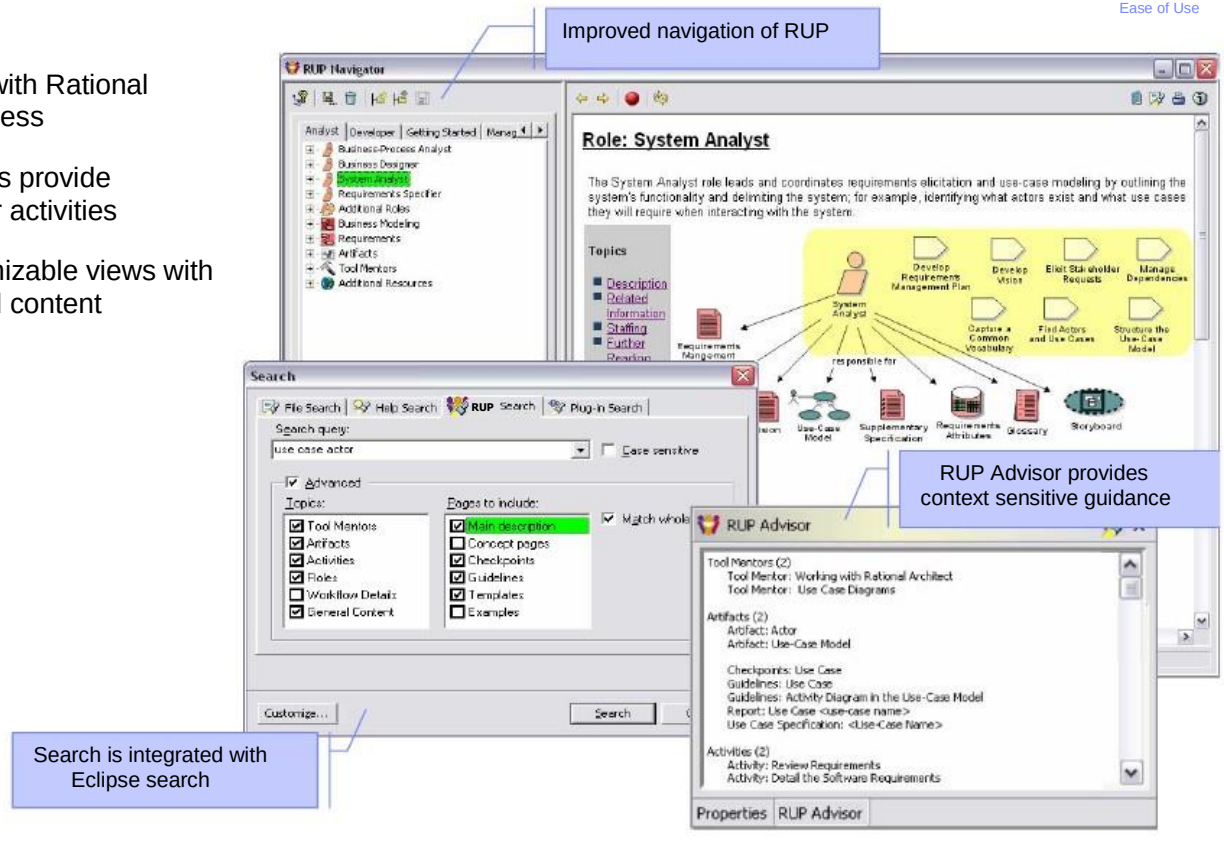
Team: Process Guidance

Ease of Use

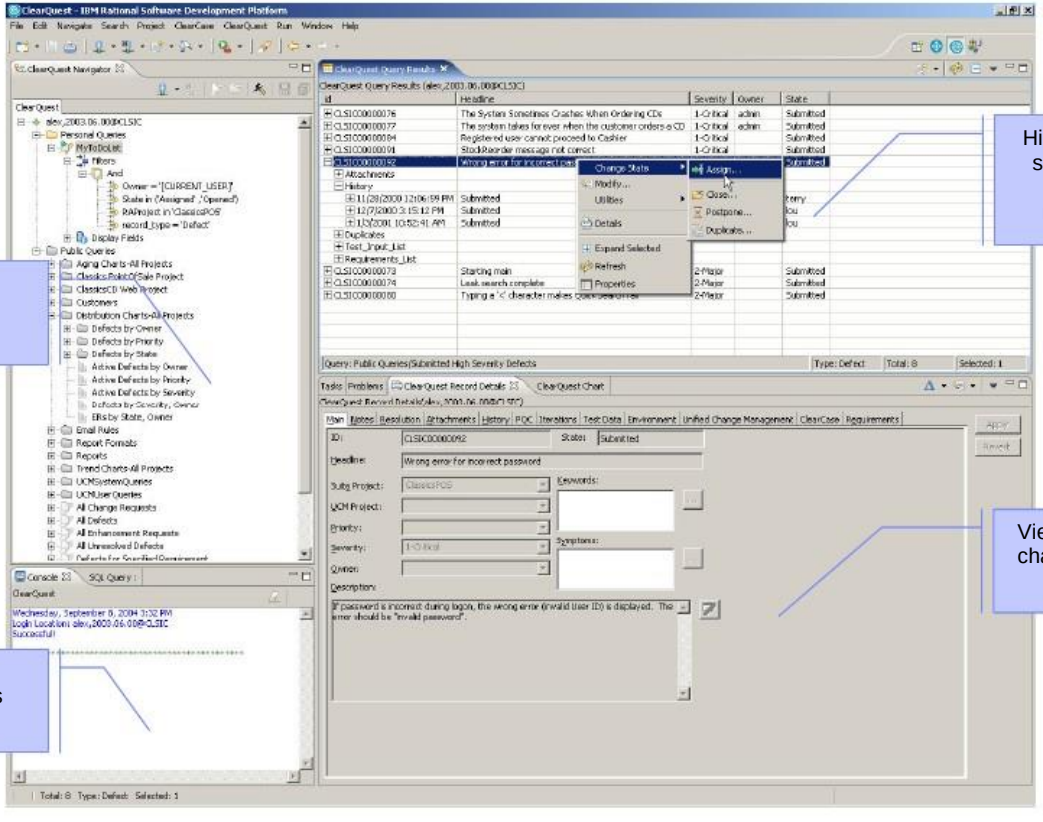
Integration with Rational Unified Process

Tool Mentors provide guidance for activities

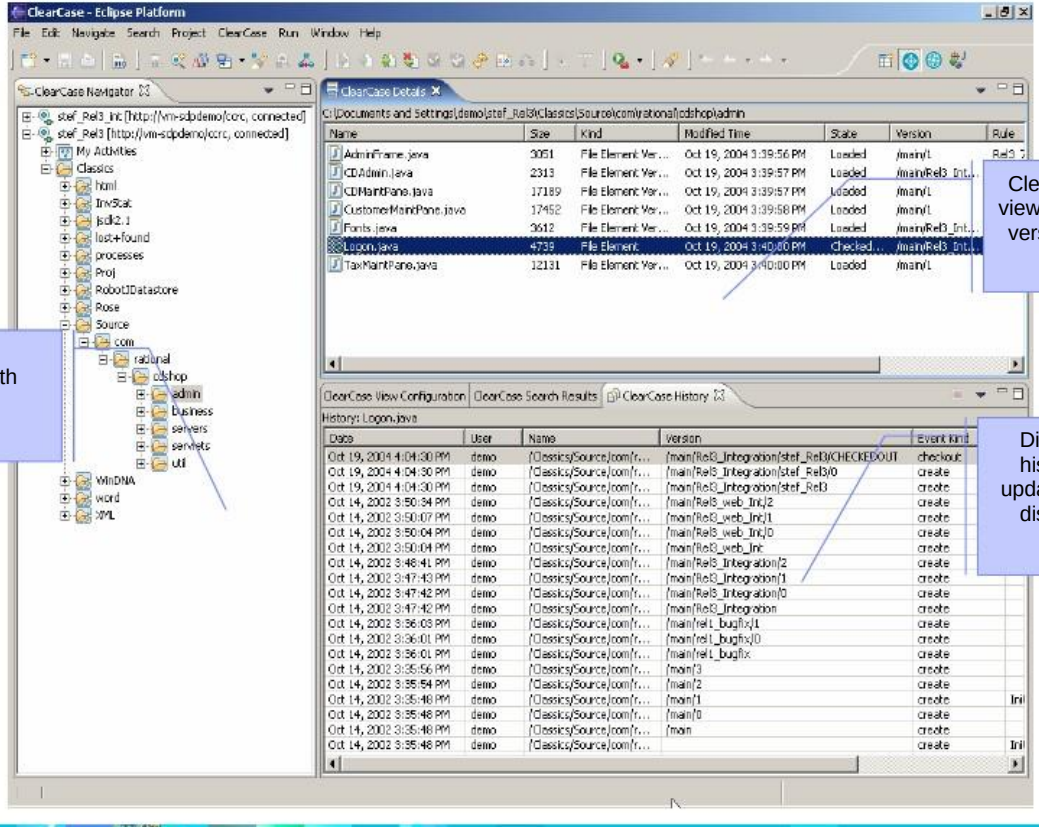
User customizable views with user defined content



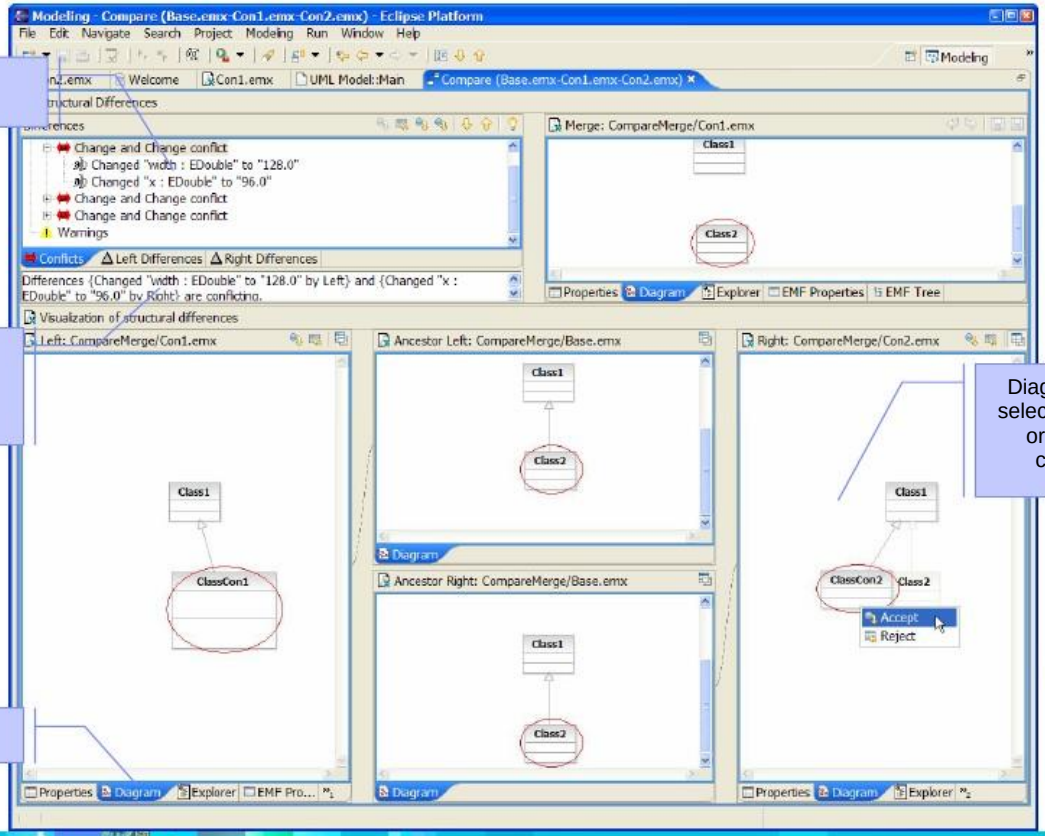
Team: ClearQuest integration



Team: ClearCase integration



Team: Model compare & merge



Summary: Key Features Rational Software Architect SE

Architecture Support

- Java, J2SE, C++
- UML2 Modeling
- Architecture Discovery via Application Analysis
- Patterns and Transformations

Team Environment

- Enhanced Compare / Merge
- Integrated RequisitePro Views
- Process Advisor
- CC and CQ fully integrated

Open Platform

- Based on Eclipse 3.4 Shell
- Testing and Team tools work together