

Effektives Software-Design und -Entwicklung mit IBM Rational Software Architect, UML und Model Driven Architecture



Michail Matjuchin
IBM Software Group, Rational
Austria

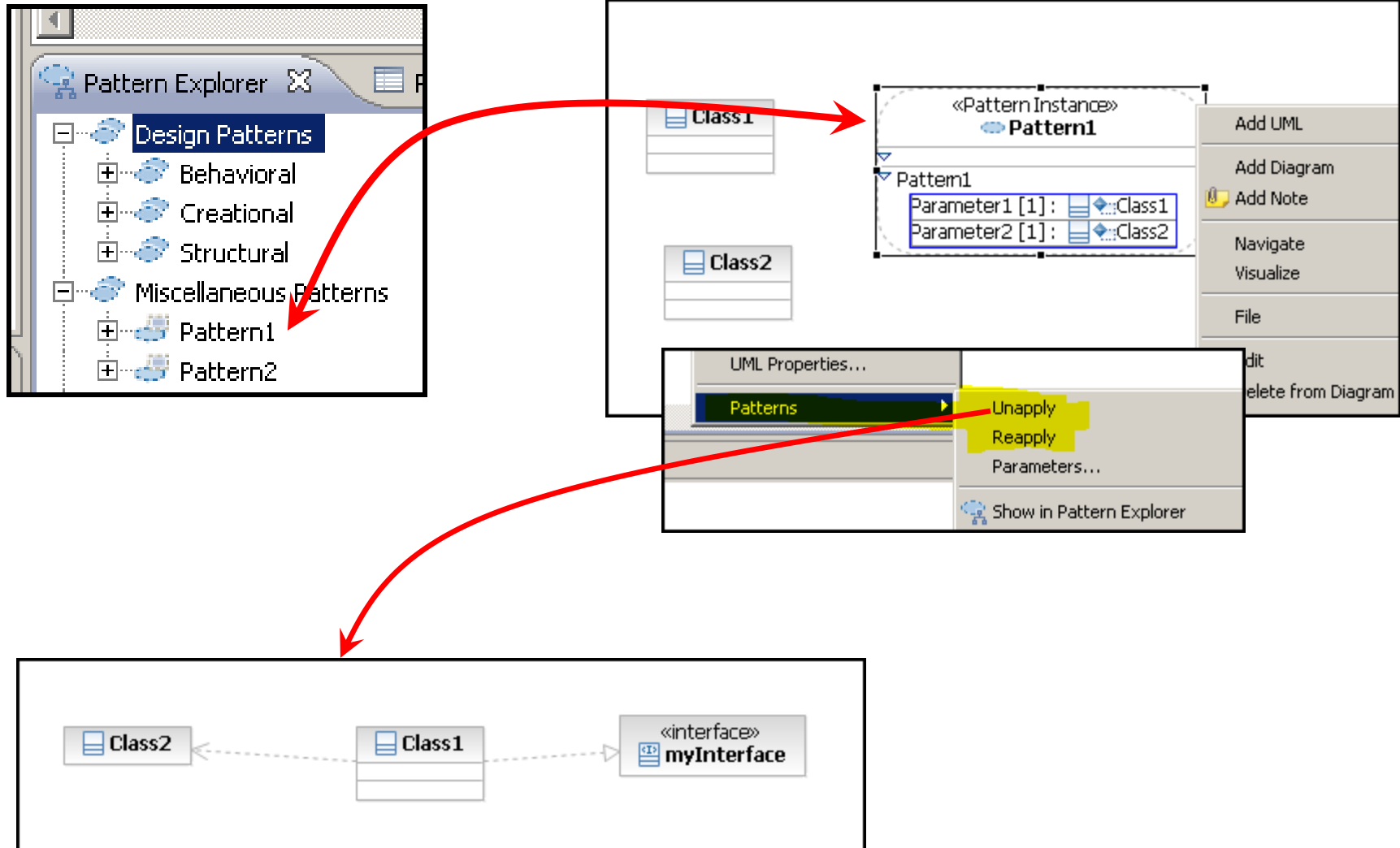
Rational. software

Agenda

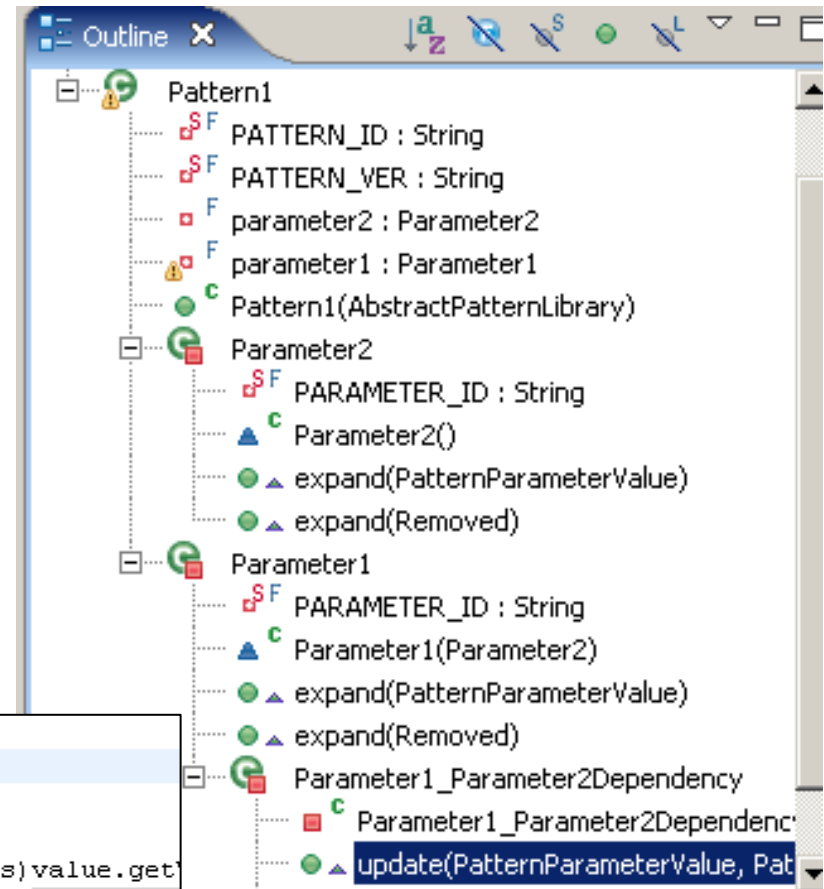
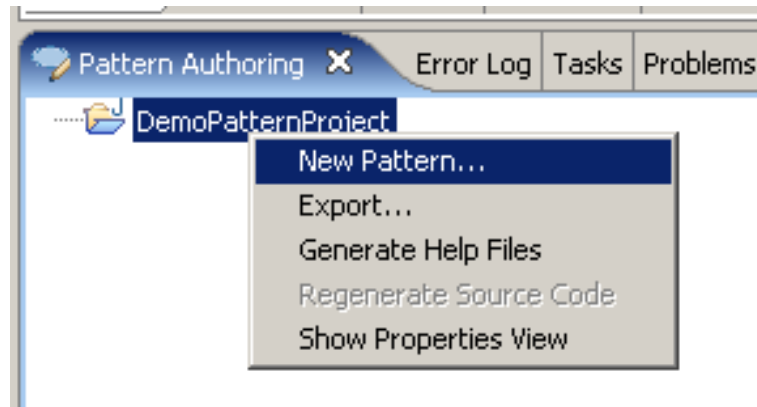
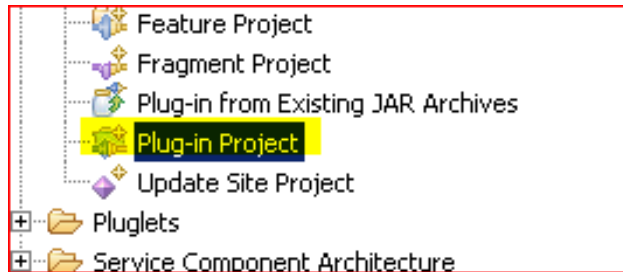
- UML Pattern
- Domain-Specific Languages
- RESTful Service Modeling
- IBM Deployment Planning and Automation
- Out-of-the-Box Transformations
 - EJB 3 & JPA
 - Web Services
- Questions



UML Pattern rise Modeling Productivity



New Patterns can be developed



```

public boolean update(PatternParameterValue value,
    PatternParameterValue dependencyValue) {
    //TODO: implement the dependency's update method

    org.eclipse.uml2.uml.Class a=(org.eclipse.uml2.uml.Class) value.get
    org.eclipse.uml2.uml.Class b=(org.eclipse.uml2.uml.Class) dependenc
    a.createDependency(b);

    return true;
}

```

UML Pattern



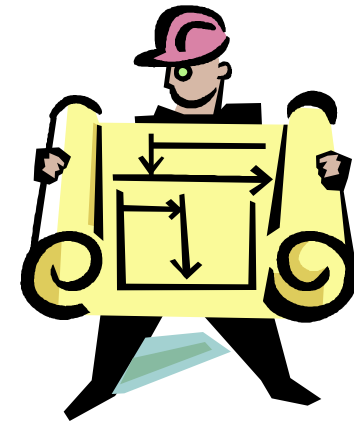
DEMO

Domain-Specific Languages and Modeling



Domain-specific modeling (DSM)

- DSM involves using a graphical DSL to visualize, specify, construct, and document a software system
- Goals of DSM:
 - Raise the level of abstraction on programming languages
 - Focus solution models
 - Hide or eliminate unnecessary details
 - Show or accentuate key details
 - Better utilize the skills of each team member



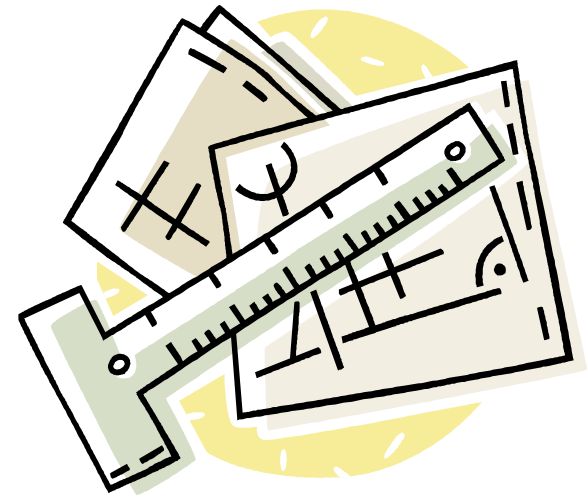
Domain-specific languages

- A **DSL**:
 - Is used to solve problems in a particular problem domain
 - Provides a restricted and common vocabulary of terms
 - Provides common approaches to problem-solving

- In contrast, **general-purpose languages** are created to solve problems across many domains
 - Examples
 - UML
 - Java
 - C++

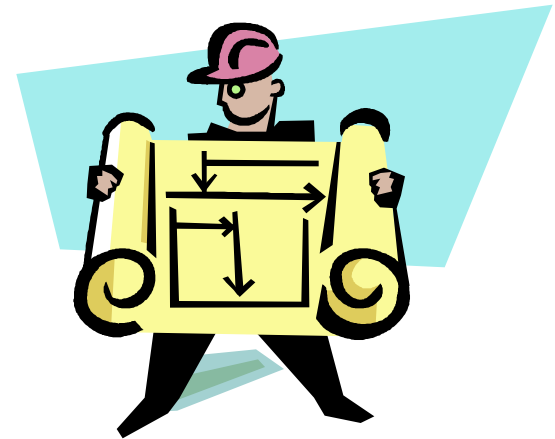
Value proposition for architects

- From the architect's point of view, DSLs and domain-specific modeling:
 - Improve communication between experts with different roles and skills
 - Provide reuse opportunities, with the right tooling
 - Constrain the solution space



Value proposition for developers

- From the developer's point of view, DSLs:
 - Improve consistency
 - Provide a better understanding of the artifacts that they work with
 - Help developers leverage their skills efficiently
 - Insulate the developer from irrelevant technology details
 - Make it possible to automate transitions between models

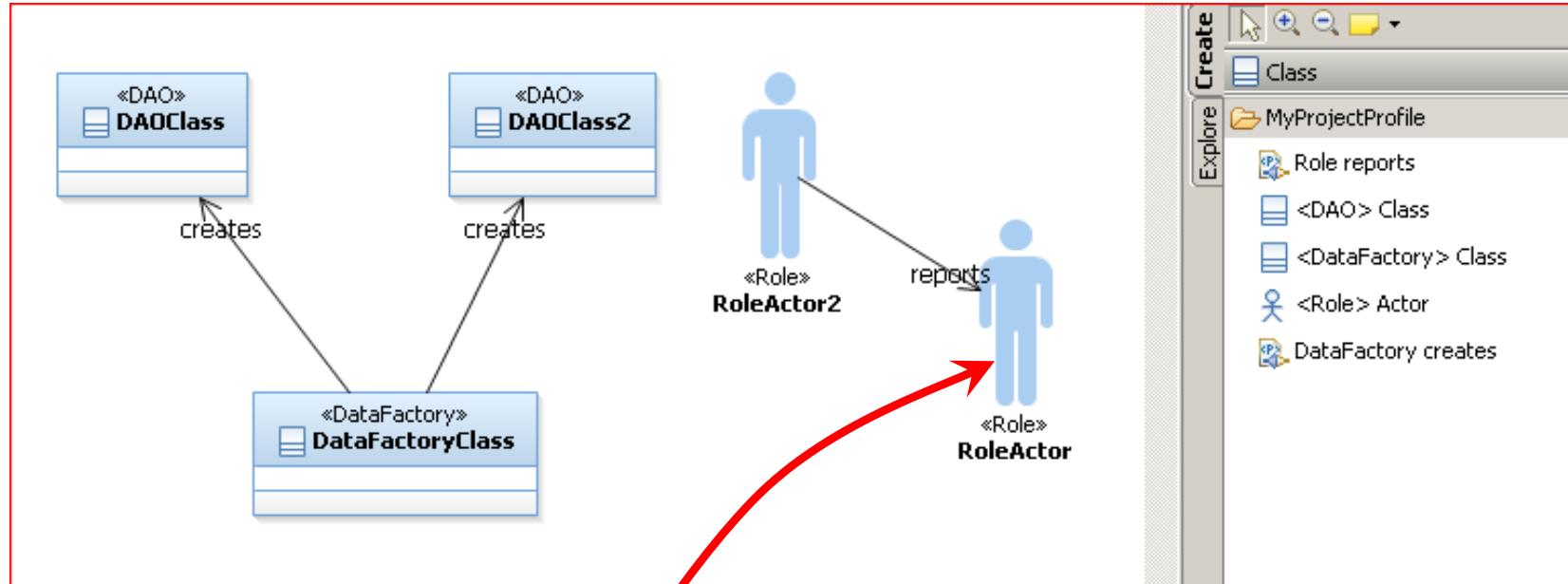


Value proposition for business stakeholders

- From the business stakeholder's point of view, DSLs provide:
 - A clearer view into the details of the solution
 - More precise validation of requirements to design
 - Insulation from irrelevant technology details
 - Efficient leverage of skills



DSLs simplify the Modeling



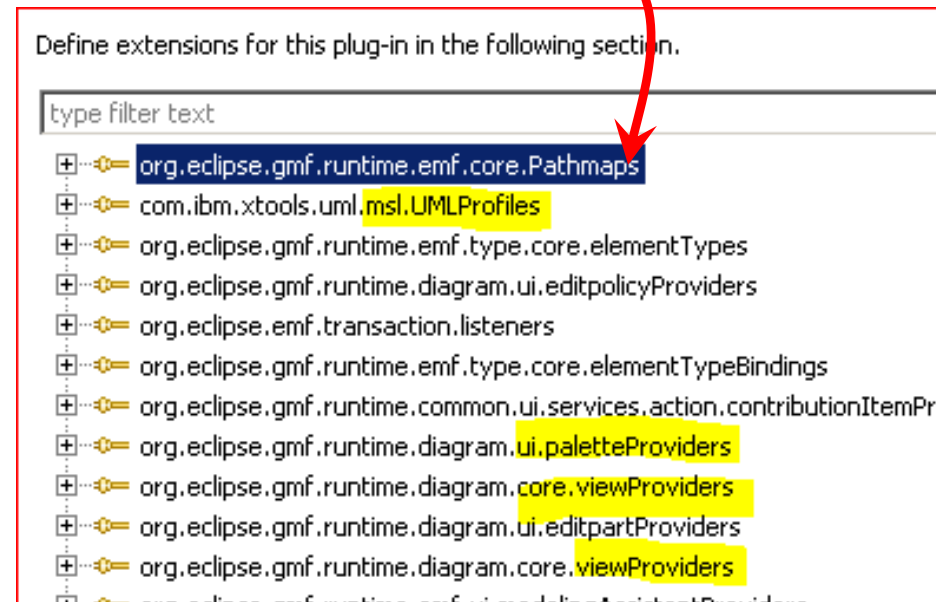
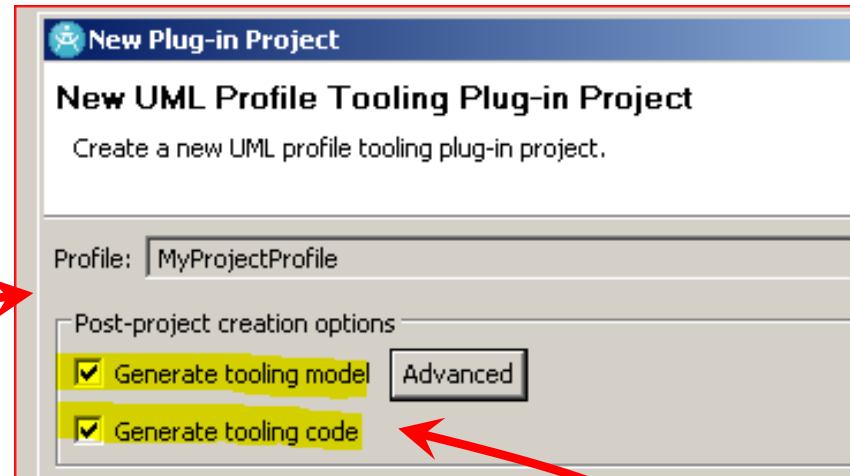
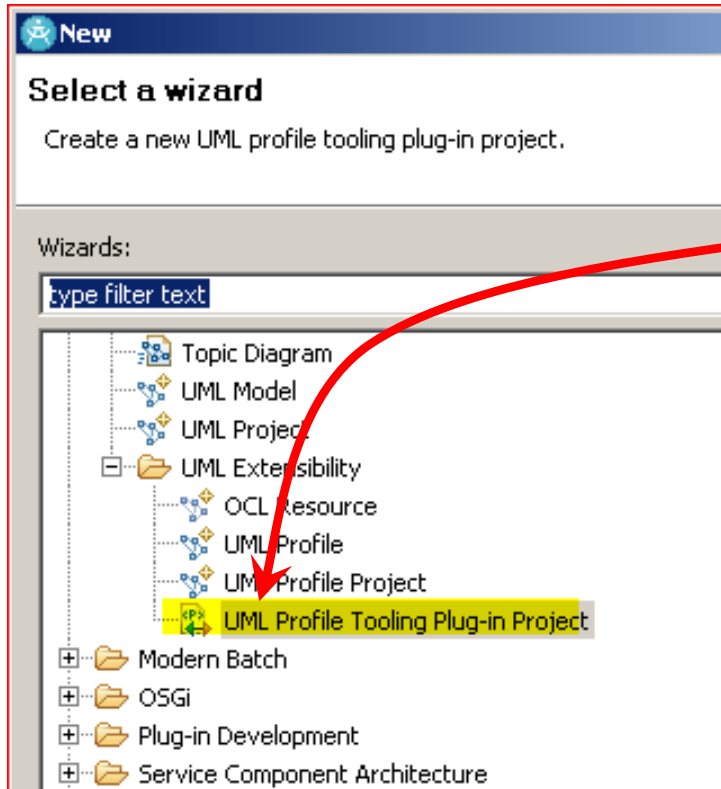
Stereotypes

Apply Stereotypes... Unapply Stereotypes

Stereotype Properties:

Property	Value
[-] Role	
department	1 - Finance
designation	2 - CFO
reports	0 - CEO 1 - HR manager 2 - CFO

UML Profile Tooling generates all necessary artifacts



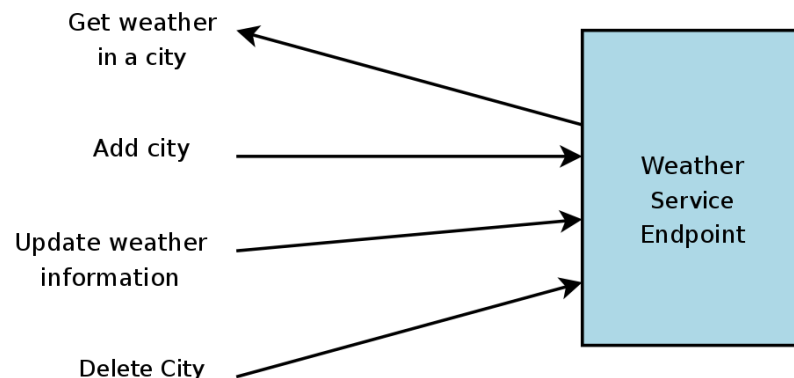
Domain-Specific Languages



DEMO

REST : REpresentational State Transfer

- REST defines a set of architectural principles for designing Web services
 - Focus on resources, including how resource states are addressed and transferred over HTTP.
- A simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web services
- Has gained widespread acceptance across the Web
 - Adoption of REST by mainstream Web 2.0 service providers—including Yahoo, Google, and Facebook
- REST Web service follows four basic design principles:
 - Use HTTP methods explicitly.
 - Be stateless.
 - Expose directory structure-like URIs.
 - Representation of resource state



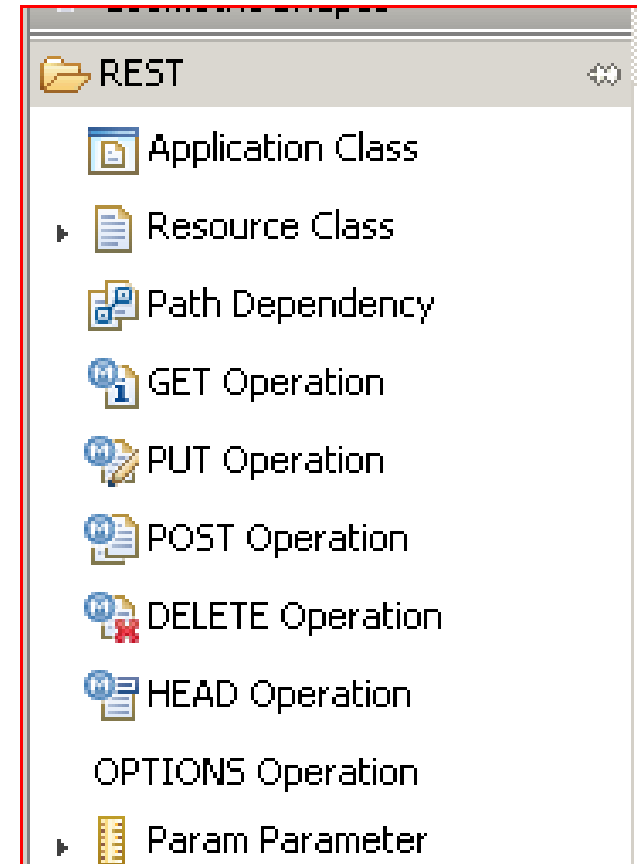
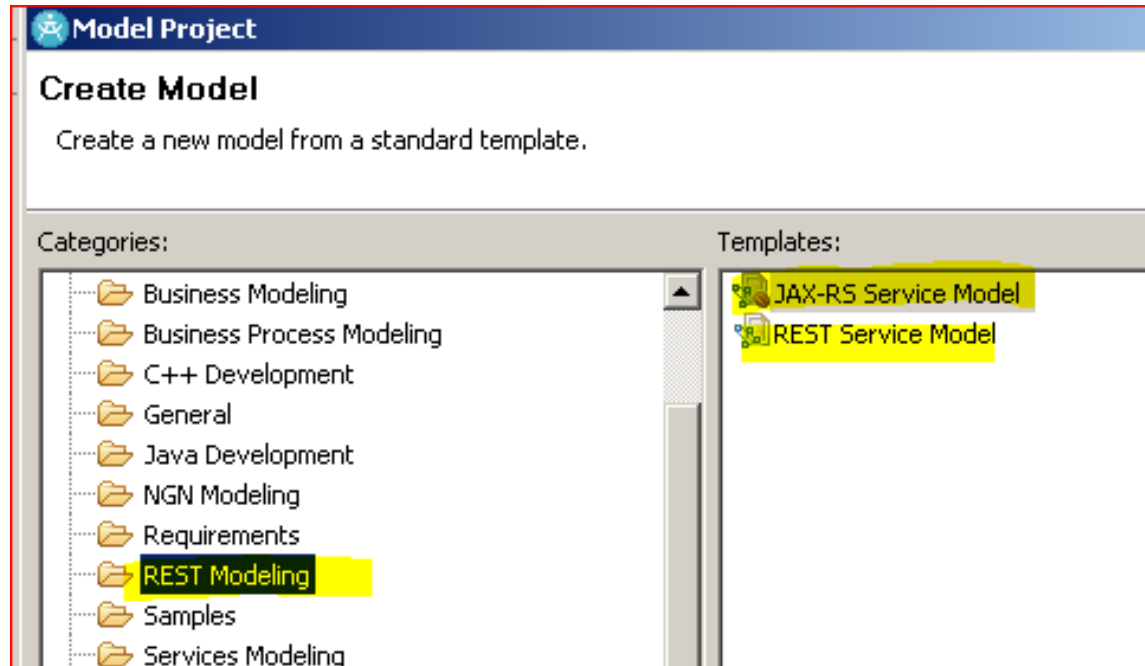
REST Concepts

- REST design principle establishes a one-to-one mapping between create, read, update, and delete (CRUD) operations
 - To create a resource on the server, use POST.
 - To retrieve a resource, use GET.
 - To change the state of a resource or to update it, use PUT.
 - To remove or delete a resource, use DELETE.
- REST suggests the design of web services be stateless
 - A client includes within the HTTP headers and body of a request all of the parameters, context, and data needed by the server-side component to generate a response
- Expose directory structure-like URIs
 - hierarchical, rooted at a single path, and branching from it are subpaths
 - *http://www.bookmarkservice.com/bookmarks/users/{john}*
 - *http://www.bookmarkservice.com/bookmarks/2008/12/10/{john}*
- Resource Representation
 - A resource representation typically reflects the current state of a resource
 - Has to do with the format of the data that the application and service exchange in the request/response payload or in the HTTP body

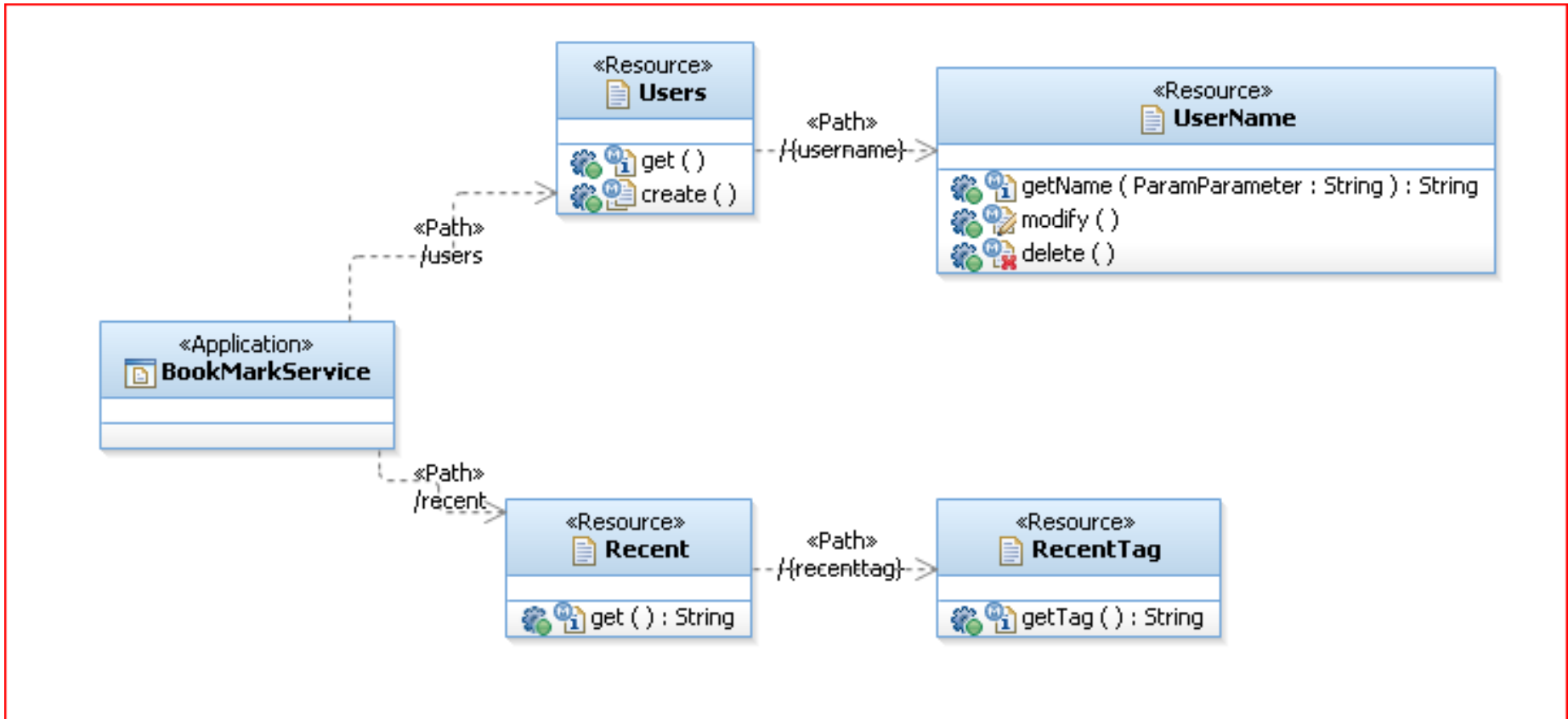
Modelling REST Services

- The key abstraction of information in REST is a **RESOURCE**
 - Any information that can be *named* can be a resource: a document, a home page of a weblog, or a search result
 - A conceptual mapping to Data Entity (or actual resource)
- Resource **Path** is a URI that identifies a particular resource
- Uses the concepts HTTP and URIs to retrieve or modify the state of a resource
 - Resource can have methods to handle HTTP request GET, PUT, POST, DELETE etc
- A **RESOURCE** can define **Input/Output** types
 - Produces/Consumes types can be specified at the individual method level as well
- **Path** between resource specifies the navigation path from a resource to a sub-resource

REST Tooling in RSA



REST Domain Model



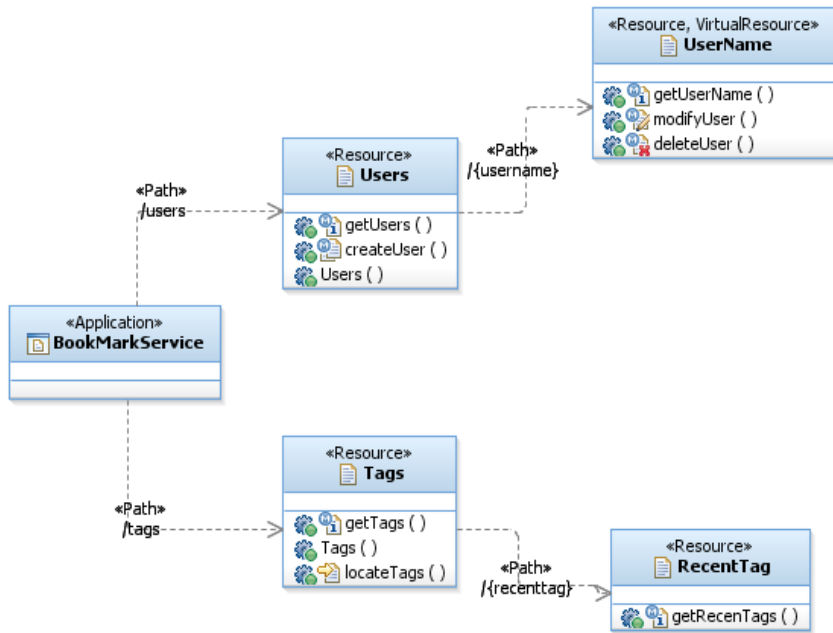
BIRT reports for REST services

- A BIRT report will be provided which will list out details for all Resources in a model

REST Resource Report

<i>Resource</i>	ChangeRequestResource
<i>URL</i>	/[CR URI]
<i>Description</i>	Change Management resources define the change requests, activities and tasks of the software delivery lifecycle. They represent individual change requests, activities and tasks, along with their relationships to other shared resource types such as project, category, release and plan. The intent of this specification is to define the set of HTTP-based RESTful interfaces in terms of HTTP methods: GET, POST, PUT and DELETE, HTTP response codes, mime type handling and resource formats. The capabilities of the interface definitions are driven by key integration scenarios and therefore don't represent a complete setup of operations on resources or resource types. The resource formats and operations may not match exactly the native models supported

JAX-RS Code Generation



```

@Path("/users")
@Produces("application/xml")
public class Users {

    public Users() {
        // TODO Auto-generated constructor stub
    }

    @GET
    @Produces("text/html")
    public List getUsers(){
        return null;
    }

    @PUT
    @Consumes("text/plain")
    public void modifyUsers(String name){

    }

    @Path("/{username}")
    public UserName getUserName(){
        return new UserName();
    }

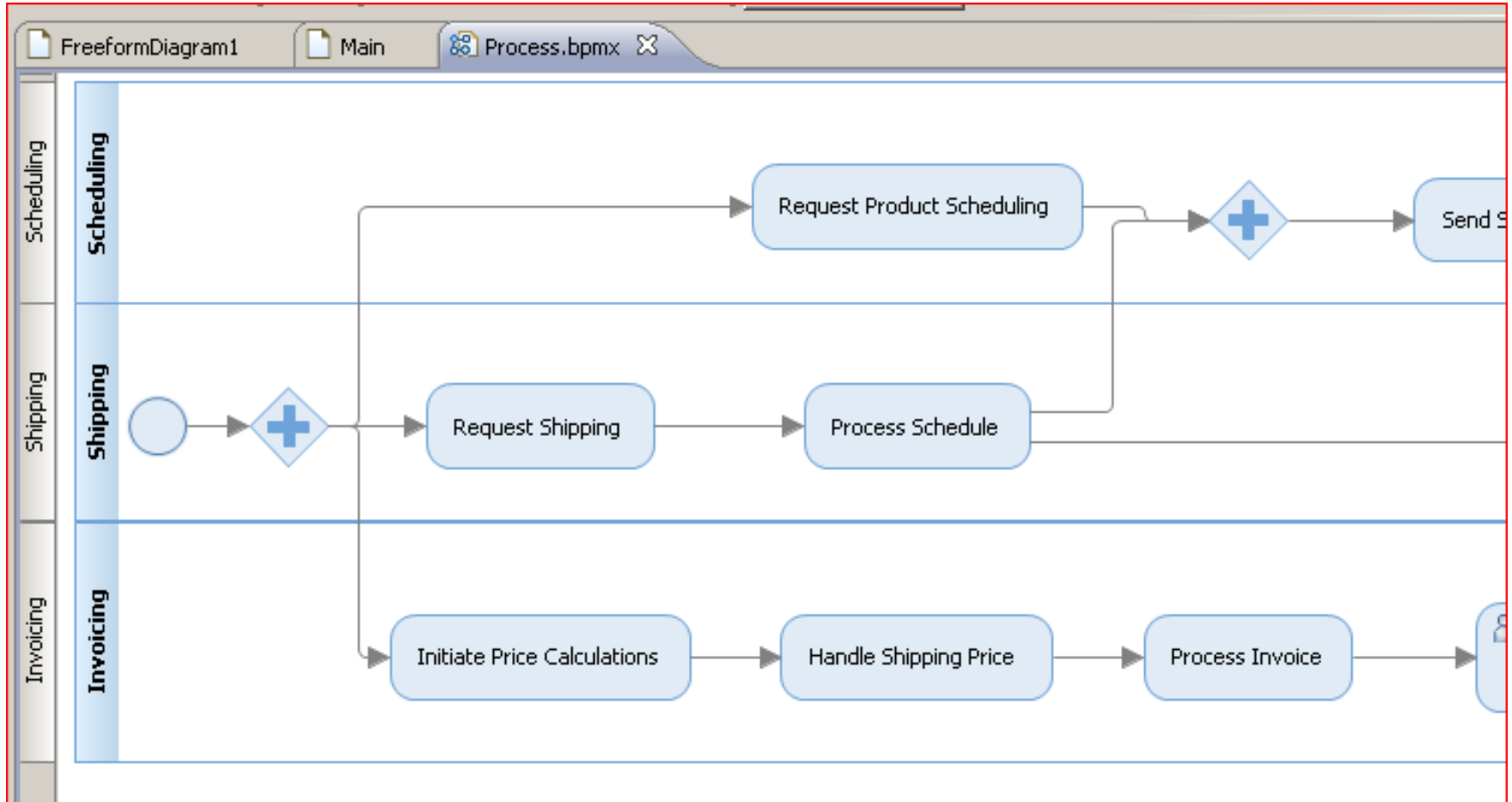
}
  
```

RESTful Service Modelling



DEMO

BPMN Modeling



IBM Deployment Planning & Automation



Introducing Deployment Planning & Automation



- Problems
 - Organizations spend too much time and too many resources on deployment
 - Manual deployment is unreliable and not repeatable
 - Inconsistent methods of hand-off between development and operations
- Which Causes
 - Concerns about the risk of deploying delays customers from getting the latest software for days, weeks, months, even years
- Solution
 - IBM tools provide process, automation and collaboration to address the silos, time and resource constraints
- Benefits
 - Automation significantly reduces cost, time and human error
 - Staged and incremental adoption allows benefits to be achieved from the outset

Deployment is a complex problem

- Development and Operations teams collaboration challenges

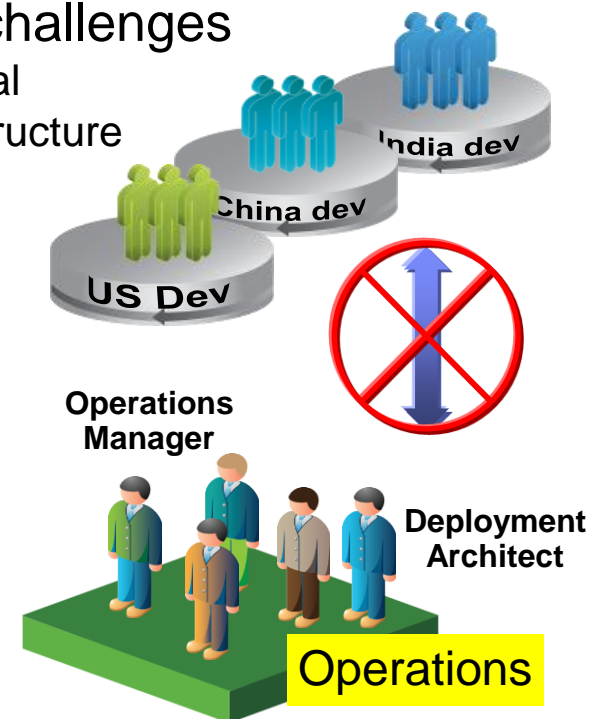
- Hand-off from development teams is inconsistent and manual
- Application component requirements do not match IT infrastructure

- Deployment requirements are difficult to validate

- Enterprise, Software & IT architects all use different formats
- No standardization or templates for reuse

- Complex series of steps

- Deployment engineers often execute manual steps
- Not repeatable, prone to error
- Automations are hard to build, maintain and reuse
- Hard to tell what if the right things were installed



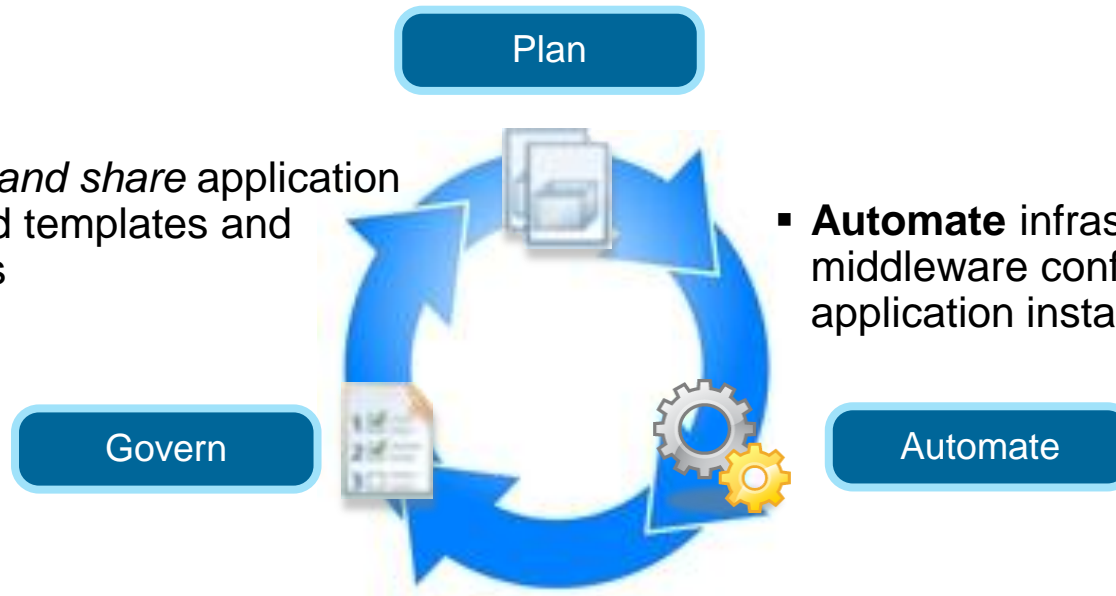
✓ 50% of applications put into production are later rolled back (*Gartner*)

✓ 60% - 80% of an average company's IT budget is spent on maintaining existing applications
(*Intelligent Enterprise.com*)

✓ Software related downtime cost industries almost \$300 billion annually (*CENTS - Comparative Economic Normalization Technology Study*)

Introducing IBM Deployment Planning and Automation

- **Plan** your desired deployment using discovered resources and standard configuration templates

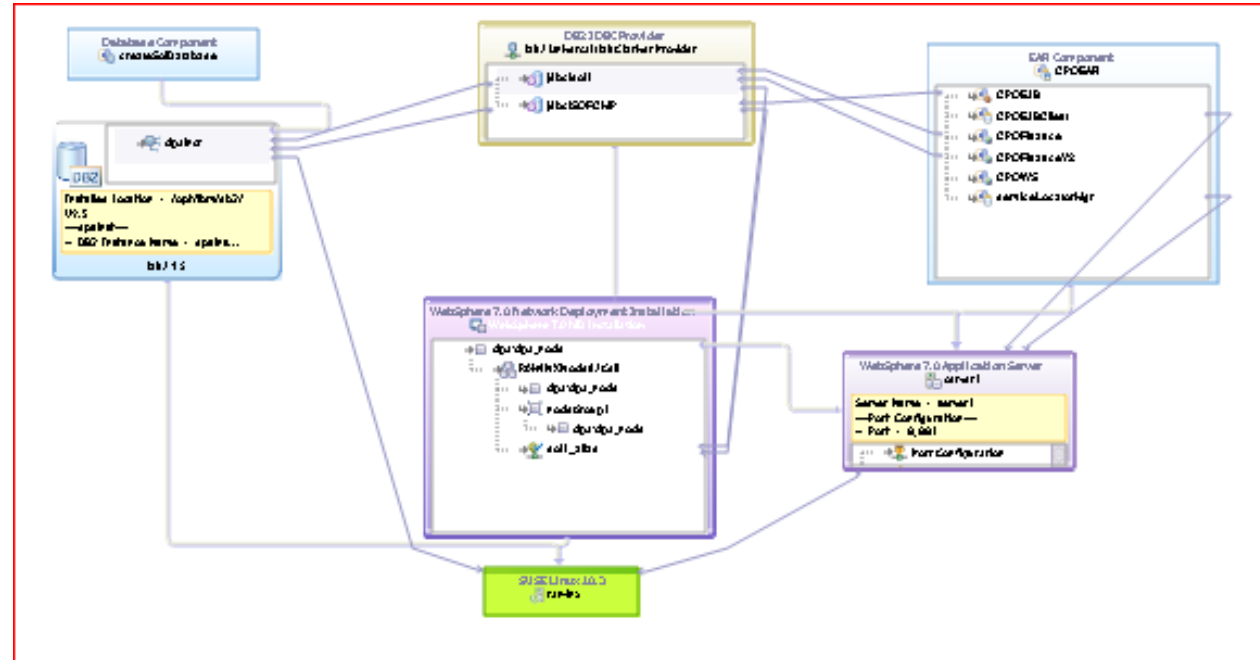
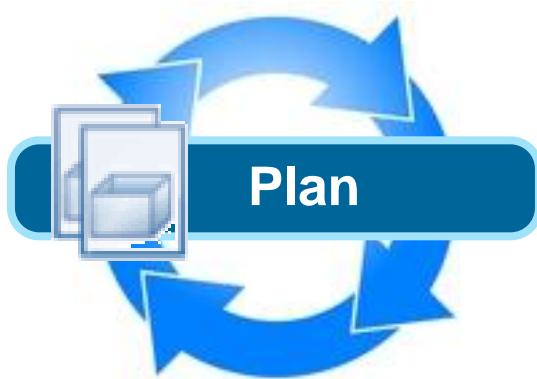


- **Govern**, *catalog and share* application artifacts, standard templates and deployment plans

- **Automate** infrastructure provisioning, middleware configuration, and application installation

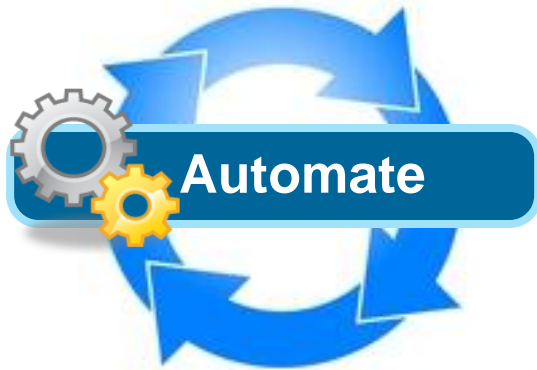
Speed the delivery of high quality applications to physical environments, virtual environments, and cloud environments

IBM Deployment Planning and Automation lifecycle



- **Rational Software Architect (RSA)** allows you to plan and validate deployment of applications and infrastructure as well as generate and publish workflows to drive automation.

IBM Deployment Planning and Automation lifecycle



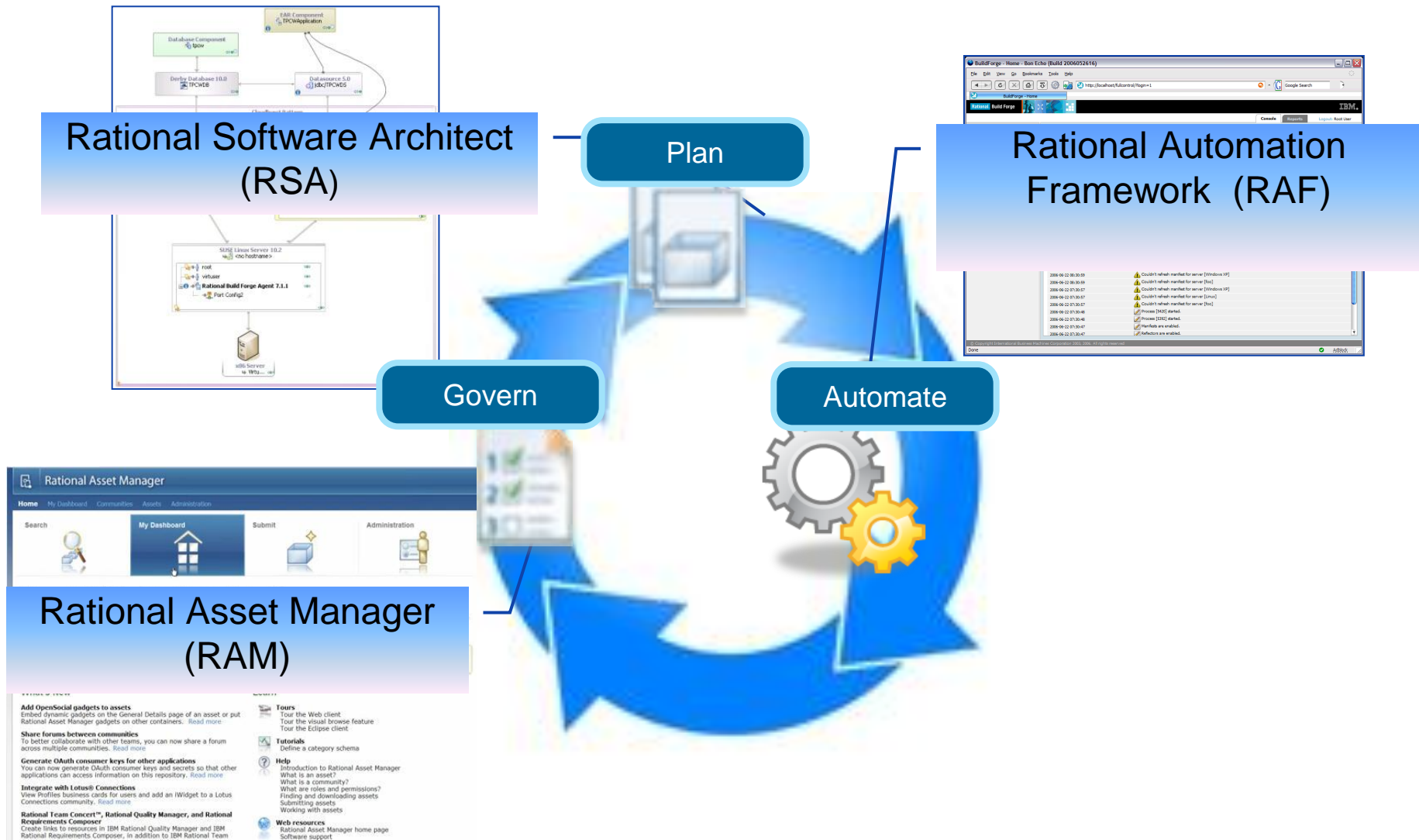
The screenshot shows the BuildForge web console interface. The browser address bar indicates the URL is `http://localhost:fulcontrol/login=1`. The console displays a table of build history under the "Last Builds Run" tab.

Tag	Project	State	Status	Date	Runtime	Owner
BUILD_10	Project 1	Complete	✓	2006-06-21 14:06:27	0:10:20	Root User
BUILD_9	Project 1	Complete	✓	2006-06-20 16:22:04	0:20:28	Root User
BUILD_8	Project 1	Complete	✓	2006-06-20 16:11:48	0:20:28	Root User
BUILD_7	Project 1	Complete	✓	2006-06-20 16:11:42	0:41:06	Root User
BUILD_6	Project 1	Complete	✓	2006-06-20 16:11:36	0:10:24	Root User
BUILD_5	Project 1	Complete	✓	2006-06-19 12:00:57	0:10:05	Root User
BUILD_4	Project 1	Complete	✗	2006-06-19 11:58:51	0:00:01	Root User
BUILD_3	Project 1	Complete	✗	2006-06-19 11:57:31	0:00:01	Root User
BUILD_2	Project 1	Complete	✗	2006-06-19 11:54:46	0:00:01	Root User

Below the table is the "System Messages" section, showing a list of messages with timestamps and severity levels. The messages include warnings about manifest refresh failures and successful process starts.

- Within **Rational Automation Framework**, you can work from the published deployment workflow from RSA. The RAF automation engine will then perform automation activities to configure the middleware and deploy the application.

IBM Deployment Planning and Automation Product Mapping



Benefits of Rational Software Architect

- Topology notations supports Operational Modeling
- It is a model
 - Element changes are reflected throughout the model
 - Provides a standard notation
 - Can be validated and harvested
- Topology Status view identifies problems & potential fixes!!!
 - Tracks requirements, realizations and links
 - Out of box technology domain knowledge for WebSphere
 - Possibility to create and share other domains
- Out of box reports

IBM Deployment Planning and Automation



DEMO

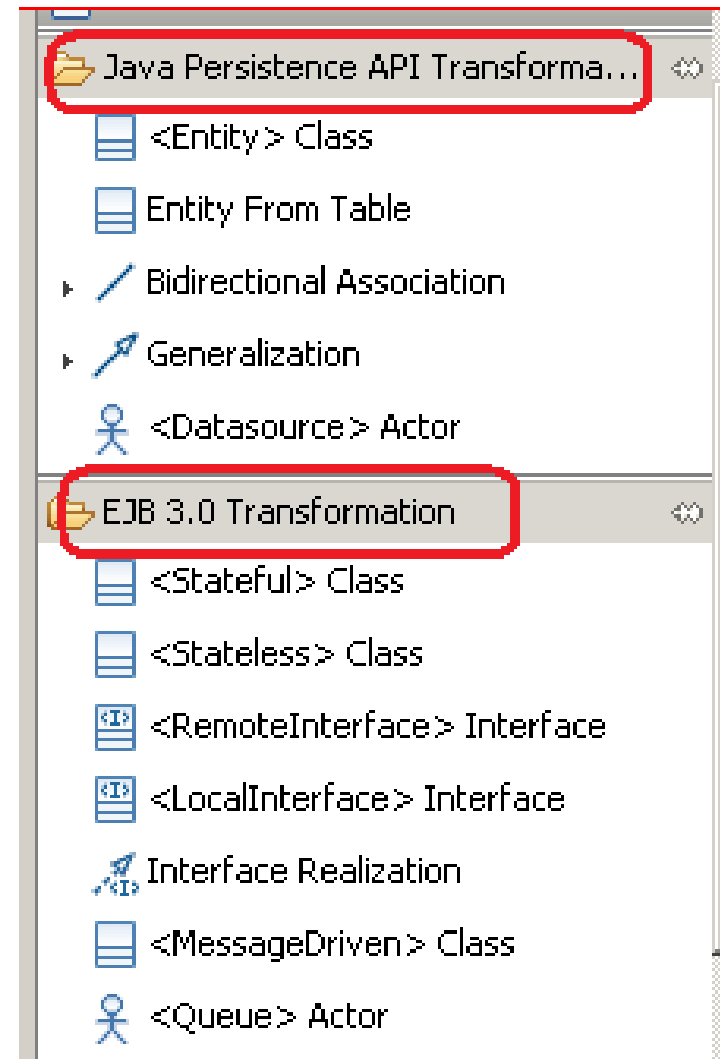
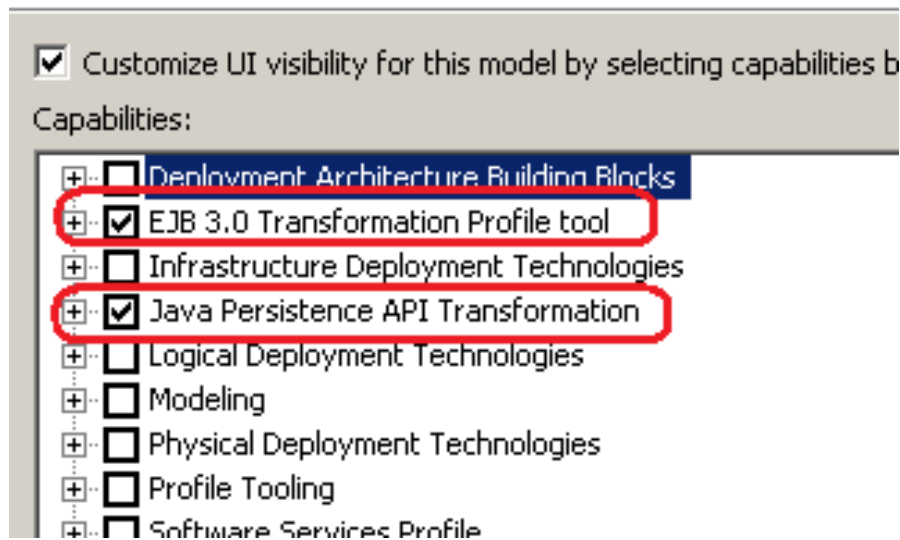
Out-of-the-Box UML Transformations



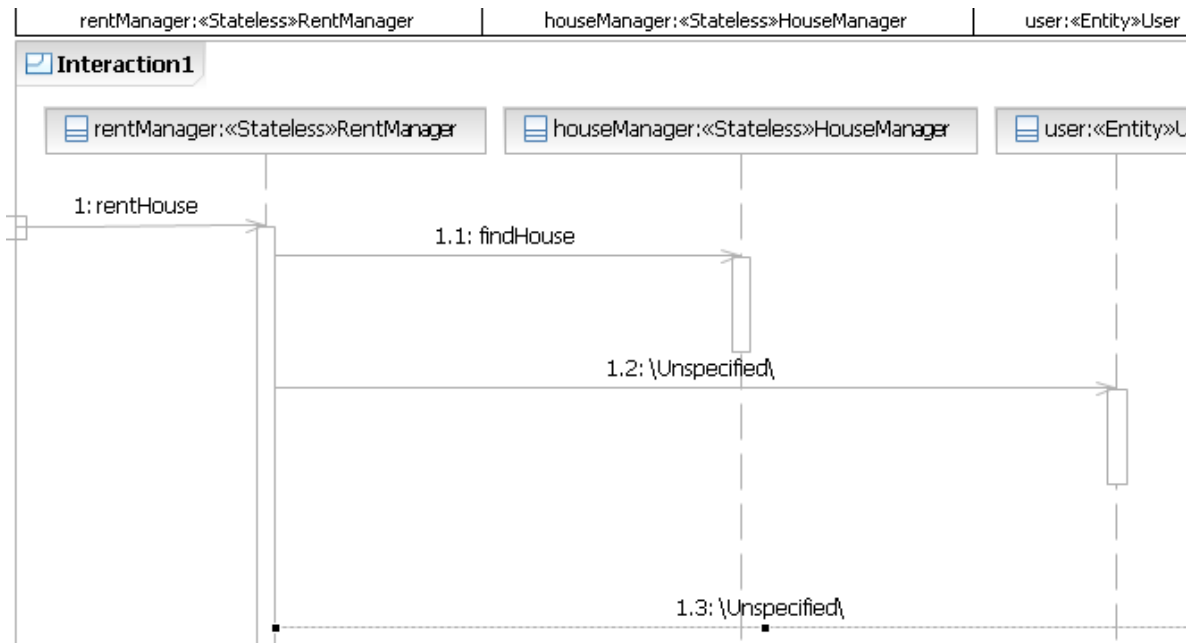
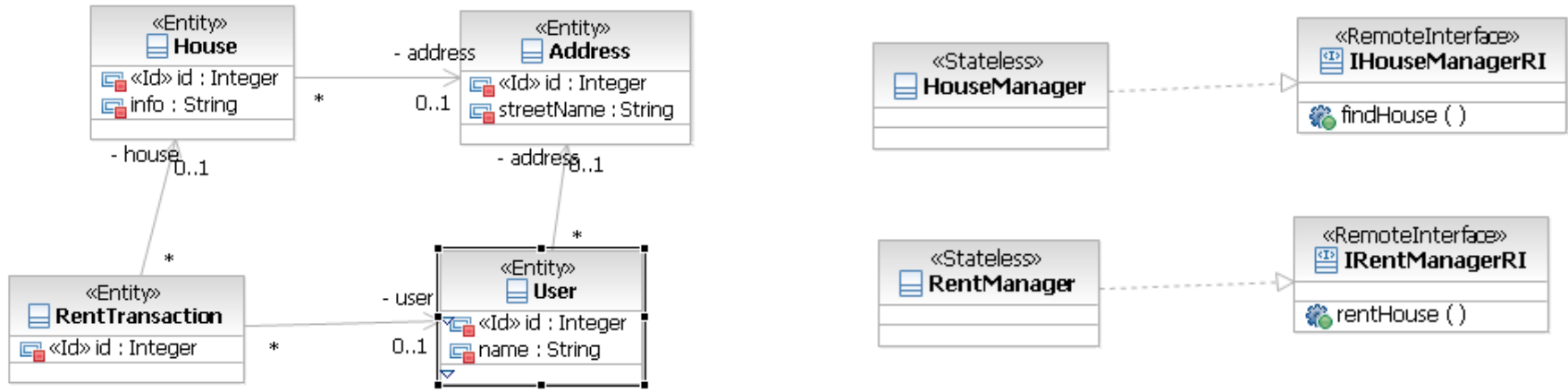
EJB 3 Transformation

Model Capabilities

Select the capabilities to associate with the new model.



EJB 3.0 Example

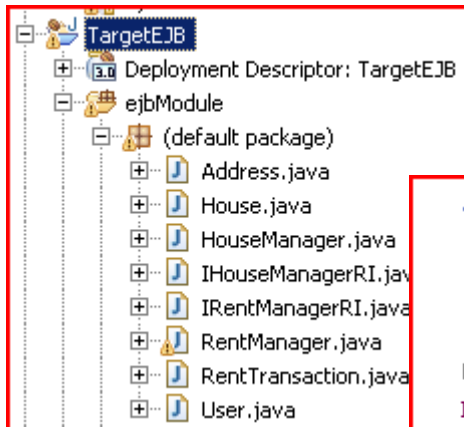


EJB 3.0 Example

```

*/
@Entity
@NamedQueries( {
    @NamedQuery(name = "House.findById", query = "select obj from House where obj.id = :id"),
    @NamedQuery(name = "House.findByinfo", query = "select obj from House where obj.info = :info")
    @NamedQuery(name = "House.findByaddress", query = "select obj from House where obj.address = :")
})
public class House implements Serializable {
    /**

```



```

/**
 * <!-- begin-UML-doc -->
 * <!-- end-UML-doc -->
 * @generated "UML-to-EJB 3.0"
 */
@ManyToOne
private Address address;

/**
 * @return the address
 * @generated "UML-to-EJB 3.0"
 */
public Address getAddress() {
    // begin-user-code
    return address;
    // end-user-code
}

```

```

public void rentHouse() {
    //IHouseManagerRI.findHouse();
    //TODO Add method parameters

    Integer id = null; //TODO initialize and set the v
    User entity = entityManager.find(User.class, id);

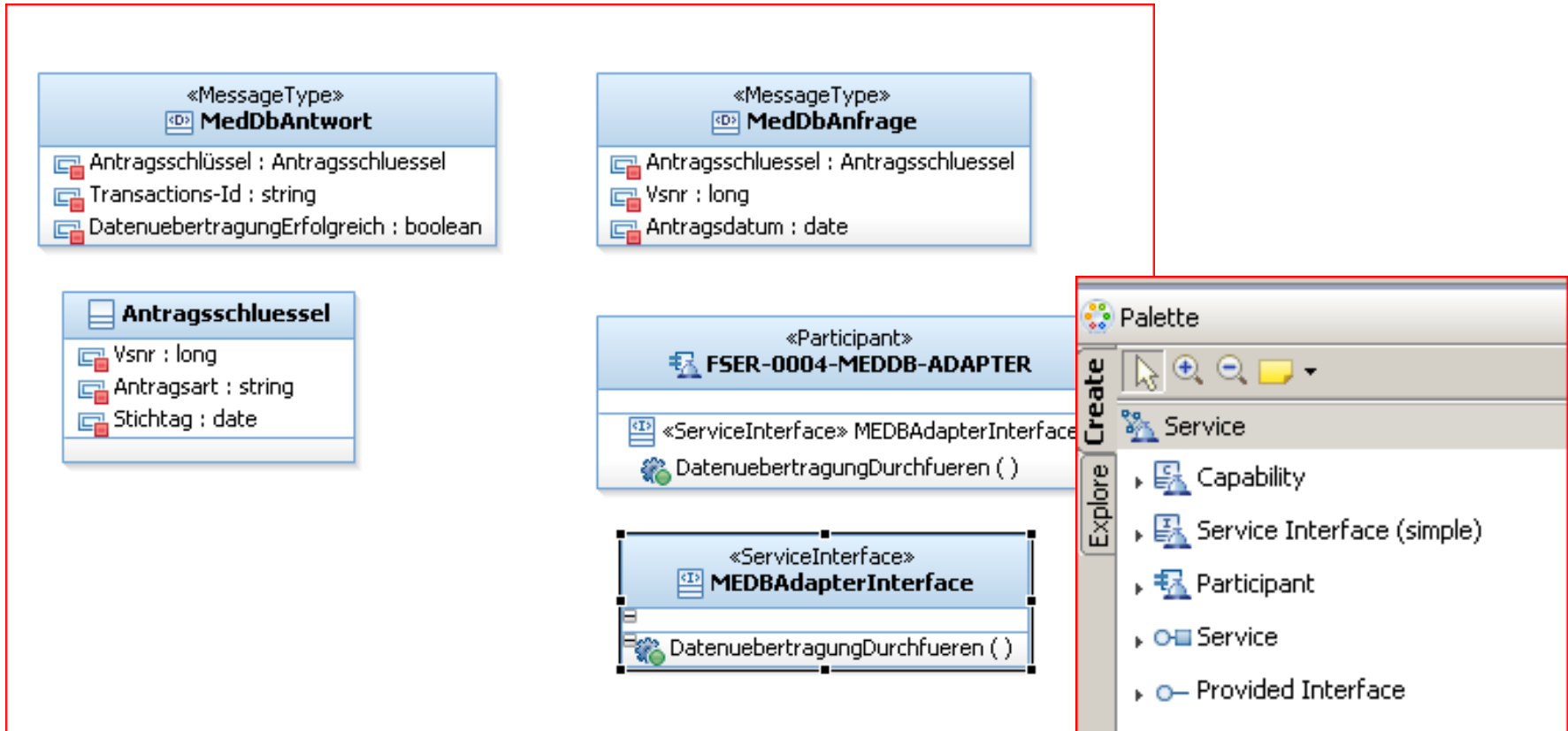
    RentTransaction entity1 = new RentTransaction();
    //TODO set the entity properties
    entityManager.persist(entity1);

    // begin-user-code
    // TODO Auto-generated method stub

    // end-user-code
}

```

Web Services Transformation



Palette

Tools: [Mouse], [Zoom In], [Zoom Out], [New]

Create

- Service

Explore

- Capability
- Service Interface (simple)
- Participant
- Service
- Provided Interface
- Part
- Service Channel
- Message Type (Data Type)

- Services Architecture
- Service Contract

Web Services Transformation

UML to WSDL Transformation: sample_conf.tc ▾

Binding options

Interface	Binding ▾	SOAP Vers
MEDDB-ADAPTER::demo.pva::service::MEDBAdapterInterface	SOAP-DOCUMENT-LITERAL ▾	1.1
	HTTP-POST	
	SOAP-DOCUMENT-LITERAL	
	SOAP-RPC-ENCODED	
	SOAP-RPC-LITERAL	
	WRAPPED-DOCUMENT-LITERAL	

FSER0004MEDDBADAPTER.wsdl ✕

The diagram illustrates the transformation of a service port into a WSDL interface. On the left, a service box contains a port named 'MEDBAdapterInterfacePort' with the URL 'http://demo.pva/service/...'. An arrow points from this port to a central icon representing the transformation. On the right, the resulting WSDL interface 'MEDBAdapterInterface' is shown with the operation 'DateneruebertragungDurchfueren'. The interface has two messages: an input message 'MEDBAdapterInterfaceDateneruebertragungDurchfuerenRequest1' with the body 'MedDbAnfrage', and an output message 'MEDBAdapterInterfaceDateneruebertragungDurchfuerenResponse1' with the body 'MedDbAntwort'.

UML Transformations



DEMO

QQQ

- QQQ



Questions

Thank You