Developing a J2EE Architecture with Rational Software Architect Using the Rational Unified Process

Jean-Louis Maréchaux

August 16, 2005

This article shows how you can use IBM Rational Software Architect to define a software architecture. It assumes that you are familiar with the IBM Rational Unified Process methodology, and with iterative development.

Introduction

This article illustrates the elaboration of a software architecture for a fictitious company, Yummy Inc. Using the IBM® Rational Unified Process® (RUP®) methodology, you will focus on the *Analysis and Design* discipline to create a Software Architecture Document, from which you can then define the architecture with IBM® Rational® Software Architect (IRSA).

The RUP methodology is a software engineering process that provides a set of customizable best practices and documented activities that help businesses reliably deliver quality software. The goal of this article is not to describe the RUP approach, but rather to focus on architecture from a RUP perspective. Therefore, the reader should already be proficient in RUP activities and iterative development.

The main purpose here is to illustrate how the IRSA tool can help in architectural analysis and documentation. That’s why this article will focus exclusively on the Analysis and Design discipline of the RUP process.

IRSA is a design and development tool that leverages Model-Driven Architecture principles (MDA -- see the Related topics section), in which modeling is the foundation of software development. IRSA helps you define models at various levels of abstraction, and can of course be used to support activities of an engineering process like the RUP methodology.

Architectural analysis: the RUP Elaboration phase

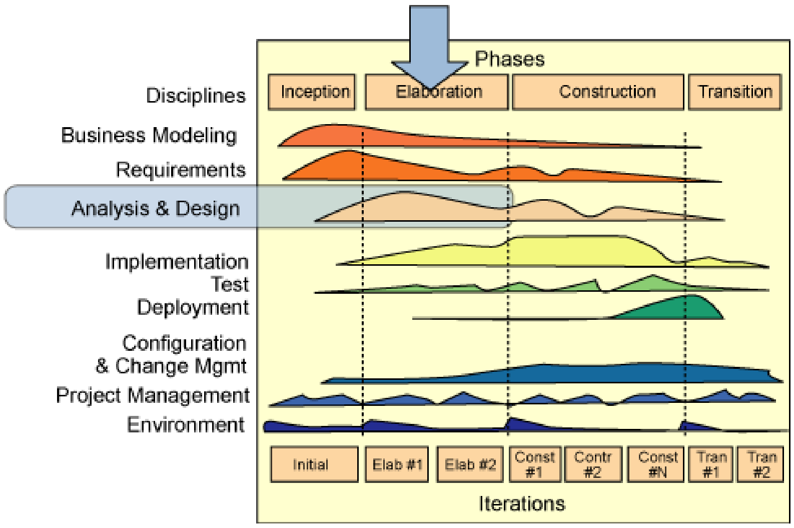
The RUP Elaboration phase's purpose is to baseline the architecture of the system you are developing. At the end of this phase, the main milestone is to have a stable architecture that will be used as the basis for system development. The *Analysis and Design* discipline contains a set of activities to help you to achieve this objective. The principal ones include:

- Defining a candidate architecture

- Analyzing the behaviors identified in the requirements
- Designing the components of the system

Of course, there are a lot of different ways to conduct architectural activities. The RUP approach is a configurable process, and you will only produce what seems to be relevant for your project, focusing on the *Analysis and Design* discipline of the Elaboration phase, as highlighted in Figure 1 following.

Figure 1. RUP activities



As a RUP Software Architect, you will typically follow these steps:

1. **Identify and prioritize significant Use-Cases:** The key requirements
2. **Define the candidate architecture:** The high level architecture of the system
3. **Define the initial Deployment Model:** The nodes onto which to deploy the system
4. **Identify key abstractions:** The key data elements used in the system
5. **Create an Analysis Model:** The key components to implement the requirements
6. **Create the Design Model:** The layers, the subsystems, and the components
7. **Document concurrency mechanisms:** The infrastructure to support the system concurrency requirements
8. **Create the Implementation Model:** The source code and source file structure

Through these modeling activities, you will dive into each aspect of the system architecture, from the abstract level to detailed design of the components.

Later in this article, you will go into more detail for each of these steps while defining the architecture of the Yummy Inc project.

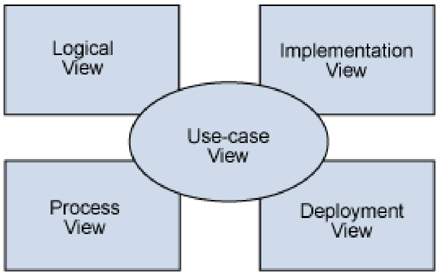
The Software Architecture Document

In the RUP approach, the main artifact used to document the architecture of a software solution is called the Software Architecture Document (SAD). It is used to gather and communicate all of the architectural decisions made during the Elaboration phase. You can Download the sample SAD that supports this article.

The SAD provides a comprehensive architectural overview of the system, using a number of different views to depict different aspects of the solution. It is intended to capture and convey the significant architectural decisions which have been made regarding the system.

In order to describe the software as accurately as possible, the structure of the SAD is based on the 4+1 view model of architecture (see the Related topics section for more information). Shown in Figure 2, this model has been slightly modified to fit the RUP specification.

Figure 2. 4+1 architectural view model



The RUP SAD template is organized around these 5 main sections (it is sometimes enriched with an optional data view). In this 4+1 View Model (and so the SAD), each view addresses the concerns of a specific set of stakeholders, allowing various stakeholders to find what they need in the software architecture.

The Table below gives, for each view, its audience, its area, and the RUP model it summarizes.

Table 1. Architecture views

View	Audience	Area	RUP Model
Logical	Designers	Use-Case realization	Analysis Model Design Model
Process	Integrators	Performance Scalability Concurrency	Deployment Model Design Model
Implementation	Programmers	Software components	Implementation Model
Deployment	Deployment Managers	Physical nodes	Deployment Model
Use-Case	Everyone	Functional requirements	Use-Case Model
(Data) - optional	Data Specialists Data Administrators	Data persistence	Data Model

Note: Traditionally, the process view is not that important for a application as the platform deals natively with the threading aspects. Nevertheless, you can choose to document some of the concurrency issues in a deployment model, and some of the communication mechanisms (synchronous versus asynchronous) in a design model.

IRSA: Defining the architecture

The SAD is nothing but a document you can create with your favorite text editor. But the design information comes from a modeling tool. Actually, if you have chosen to produce a RUP Software

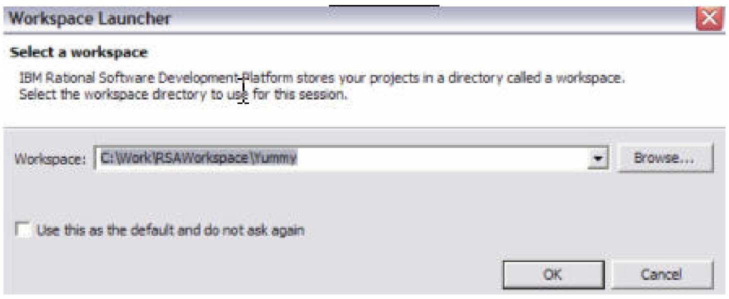
Architecture Document, you need to fill each section with the appropriate information, that is to say, with the appropriate UML diagrams from the UML model.

But before documenting the architecture you need to build it. That is where IRSA can add value to a project. To demonstrate the strengths of the product in a RUP environment, we will now define and describe the architecture of an online catering service of a fictive company: Yummy Inc. This web application provides to each online user the ability to order catering menus, and Yummy Inc takes care of the delivery. Quite simple isn't it?

Preparing the IRSA workspace

The first step is to create a new workspace. To do it, launch IRSA and specify a storage directory (Figure 3).

Figure 3. Workspace Launcher


Workspace Launcher

Select a workspace

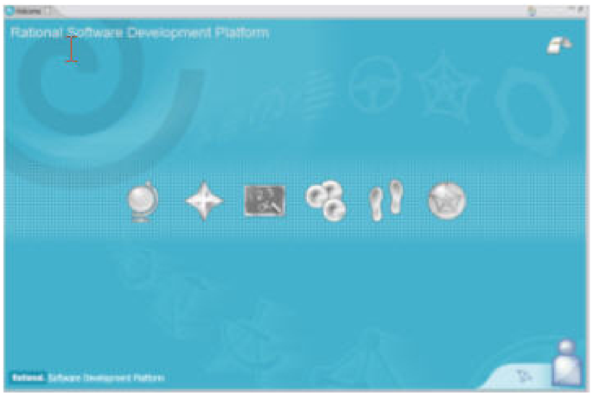
IBM Rational Software Development Platform stores your projects in a directory called a workspace.
Select the workspace directory to use for this session.

Workspace: C:\Work\RSAWorkspace\Yummy

Use this as the default and do not ask again

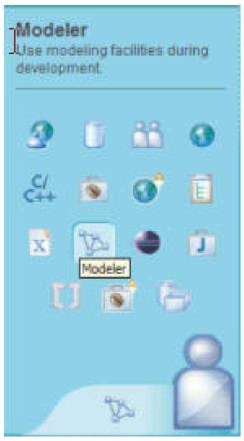
For a new workspace, IRSA shows the *Welcome* window (Figure 4).

Figure 4. Welcome window



To be able to perform modeling activities, make sure the *Modeler* role is enabled. Click the role icon in the bottom left corner and verify the *Modeler* role is selected, as shown in figure-5.

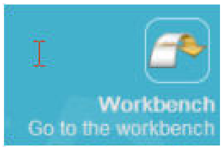
Figure 5. Modeler role selected



The role icon is something really powerful. Depending on the role you have to play in a project, you select a profile (Modeler, Web Services Developer, Tester). Then IRSA automatically enables the set of capabilities relevant to the role for you. In our example, the Modeler role will activate the Modeling perspective to facilitate all the activities a software architect has to perform.

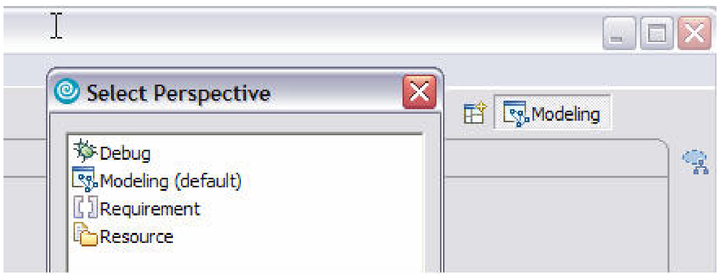
You can now go to the workspace using the icon in the right top corner (Figure 6).

Figure 6. Workspace icon



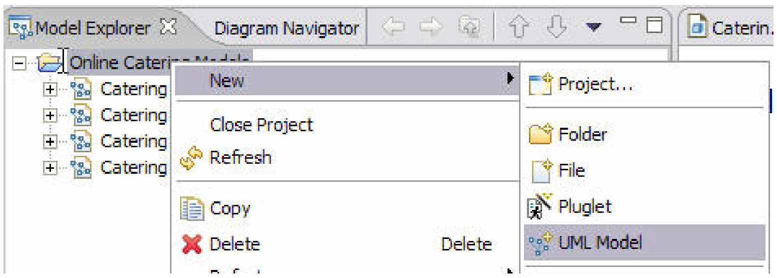
IRSA, as any Eclipse based product, is based on the notion of perspective. As you have enabled the *Modeler* role, the default active perspective is called *Modeling*, but you can of course add new ones if needed (Figure 7).

Figure 7. Modeling Perspective



IRSA comes with different pre-defined UML model templates. In the *Modeling Perspective*, you can add as many models as you need in order to document the architecture of your system (Figure 8).

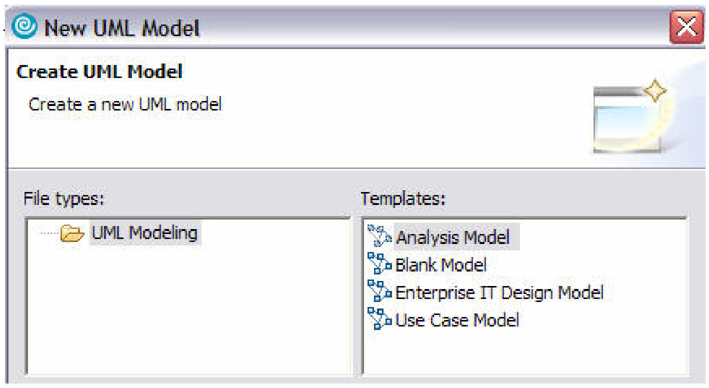
Figure 8. Modeling Perspective



Each template is dedicated to a specific architectural purpose. Typically, to document each aspect of the system (remember the "4+1" view model), you will need the Use-Case, Analysis and Design models (Figure 9). In our example, the Blank Model will also be used and configured to become the deployment model.

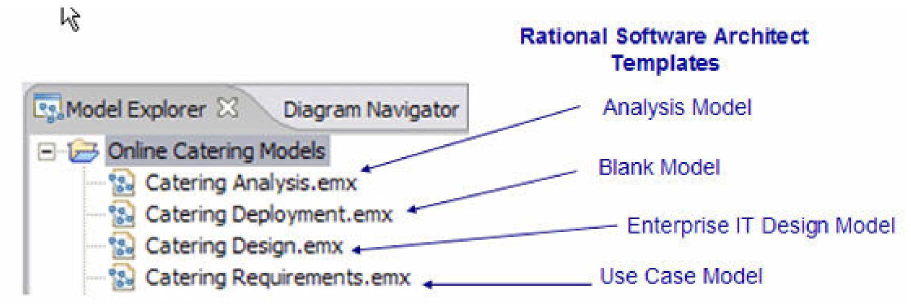
Note: In a real life project, the Use-Case Model is the responsibility of a system analyst. So the architect will rather import an existing model than create a new one from scratch.

Figure 9. UML modeling templates



So Yummy Inc.'s workspace will contain the following models (Figure 10). Each one is based on a different IRSA template.

Figure 10. Variety of models



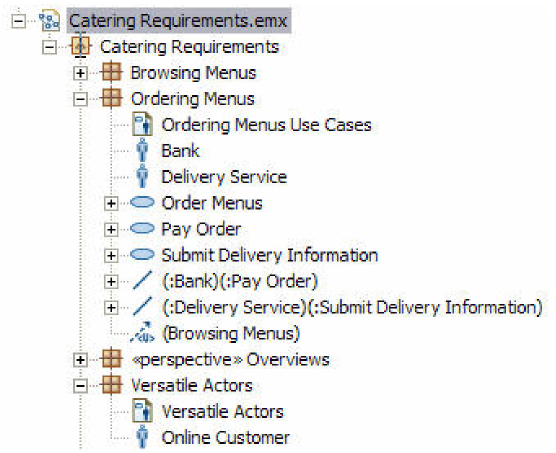
As a RUP software architect, you need to perform several activities we have already listed before. Now let's illustrate how IRSA will assist you in your step-by-step architectural analysis.

Step 1: Identify the use-cases that are significant for the architecture

One of the first tasks you will be doing during the Elaboration phase is to identify the functional requirements that are significant for the architecture. Typically, a *Vision* document and a high level *Use-Case Model* have been produced by a system analyst during the inception phase. These key artifacts are a prerequisite for any architectural activity. They define the scope of the project and allow the software architect to identify the requirements that will influence the architecture.

In IRSA, the pre-defined Use-Case Model will contain the identified key requirements (Figure 11).

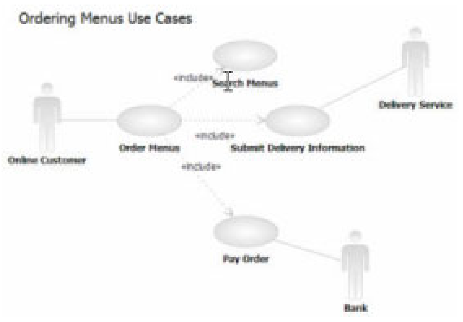
Figure 11. Key requirements



```
Catering Requirements.emx
  Catering Requirements
    Browsing Menus
    Ordering Menus
      Ordering Menus Use Cases
      Bank
      Delivery Service
      Order Menus
      Pay Order
      Submit Delivery Information
      (:Bank)(:Pay Order)
      (:Delivery Service)(:Submit Delivery Information)
      (Browsing Menus)
    «perspective» Overviews
    Versatile Actors
      Versatile Actors
      Online Customer
```

The model is subdivided into functionally-oriented packages. Each package contains the related use-cases and actors (cross-cutting actors are grouped in the versatile package). A use-case diagram illustrates graphically the requirement (Figure 12).

Figure 12. Use-case diagram

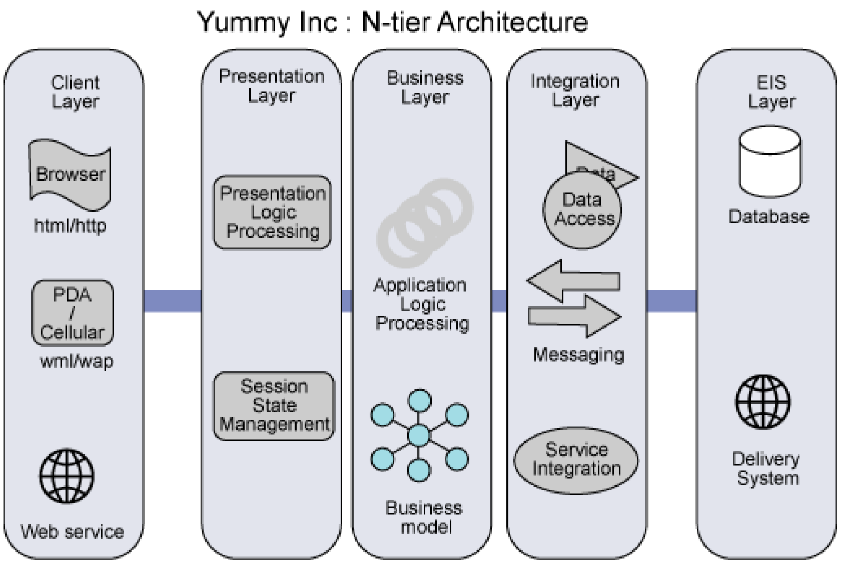


Step 2: Define the candidate architecture

In this step, the software architect creates an overview of the architecture. Most of the time, it relies on past experience with similar systems and on a *Reference Architecture* already developed. The candidate architecture takes into account the functional requirements (use-cases) but also the non-functional requirements (availability, performance, scalability…) of the system.

In our example, we have chosen an N-tier architecture (Figure 13) where the application is web-based, but will probably be accessed through web services in a near future. (Sun covers the N-tier aspects in the chapter 1 of its J2EE™ tutorial - see Related topics)

Figure 13. N-tier architecture

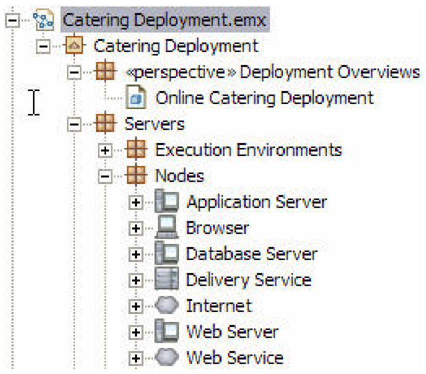


Step 3: Define the initial Deployment Model

Using the above overview of the architecture, the Software Architect can now draw the big picture of the deployment model.

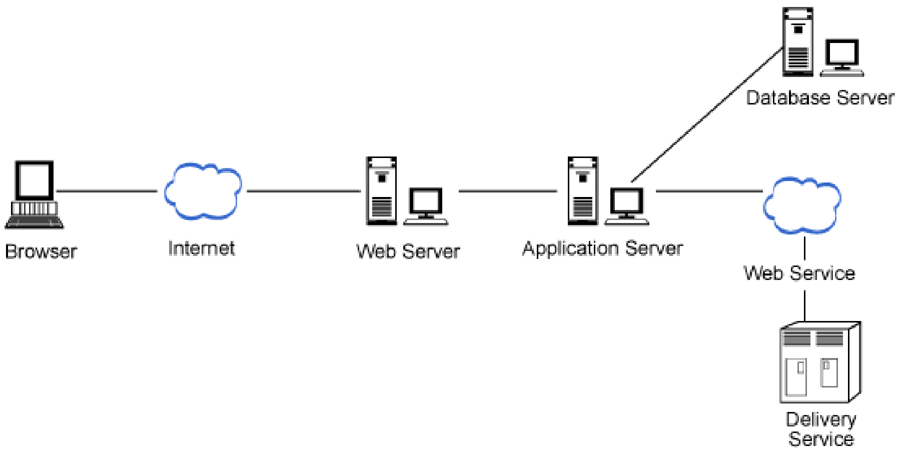
As stated before, we use the blank model template in IRSA to create our initial UML Deployment Model (Figure 14). It allows defining each node of the system, and later each execution environment as the solution is refined.

Figure 14. UML Deployment Model


Catering Deployment.emx
Catering Deployment
«perspective» Deployment Overviews
Online Catering Deployment
Servers
Execution Environments
Nodes
Application Server
Browser
Database Server
Delivery Service
Internet
Web Server
Web Service

Notice that IRSA provides (Figure 15) a rich set of stereotypes to apply graphical patterns to the nodes (desktop, tower, mainframe…).

Figure 15. Nodes

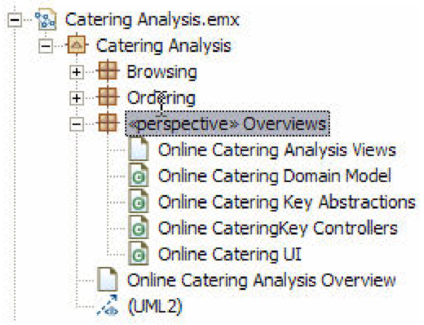


Step 4: Identify key abstractions

From the RUP requirement documents (vision, use-case, glossary, supplementary specification ...), the software architect can extract the key concepts of the solution. At this stage, it is only important to capture the overall vocabulary of the system to implement. This will be the starting point of the Analysis Model created in the next step.

The Analysis Model template provided by IRSA is pre-structured and comes with a special package named *perspective Overviews* to collect information related to key concepts (Figure 16).

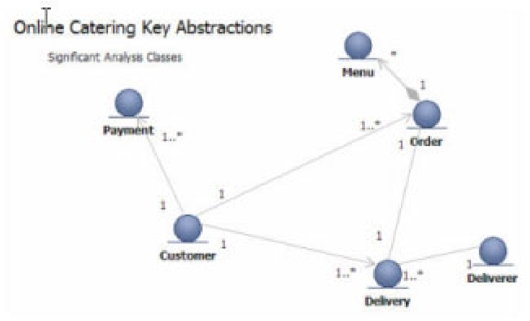
Figure 16. Analysis Model package: perspective Overviews



Catering Analysis.emx
Catering Analysis
Browsing
Ordering
«perspective» Overviews
Online Catering Analysis Views
Online Catering Domain Model
Online Catering Key Abstractions
Online CateringKey Controllers
Online Catering UI
Online Catering Analysis Overview
(UML2)

The class diagram (containing the key abstractions) is then used to document the first draft of the system using analysis stereotypes (Figure 17).

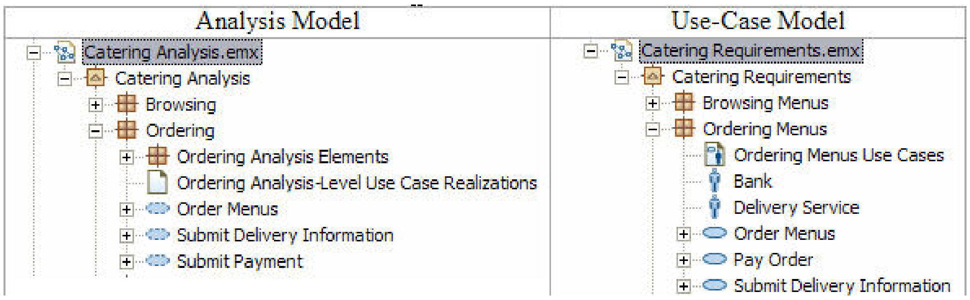
Figure 17. Analysis stereotypes



Step 5: Create an Analysis Model

The objective of this stage is to define the analysis classes necessary to realize the use-cases identified in step 1. So you will typically create the same structure as the one from the use-case model (Figure 18), grouping packages by functionality (Browsing, Ordering).

Figure 18. Analysis and use-case models side by side


Analysis Model
Catering Analysis.emx
Catering Analysis
Browsing
Ordering
Ordering Analysis Elements
Ordering Analysis-Level Use Case Realizations
Order Menus
Submit Delivery Information
Submit Payment

Use-Case Model
Catering Requirements.emx
Catering Requirements
Browsing Menus
Ordering Menus
Ordering Menus Use Cases
Bank
Delivery Service
Order Menus
Pay Order
Submit Delivery Information

The "Analysis Elements" package gathers all the analysis classes involved in the use-case realization (Figure 20). Traditionally, the classes are associated to an analysis stereotype. The *Boundary* stereotype is used to represent a class that acts as an interface to the system. The *Control* class describes a component that exercises control over other classes. The *Entity* stereotype designates a class that carries data.

A graphical representation of these UML stereotypes is shown in Figure 19.

Figure 19. UML stereotypes

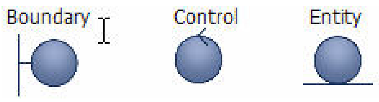
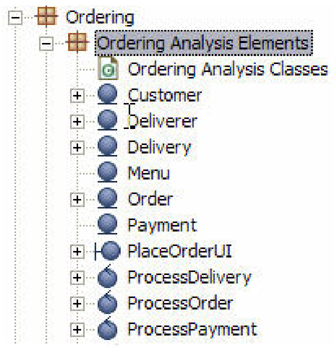


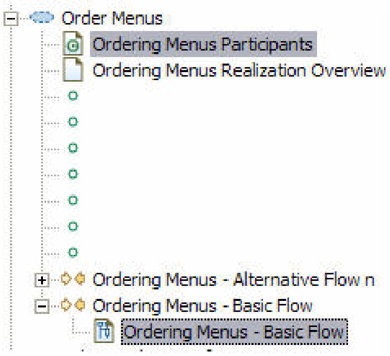
Figure 20. Analysis Elements


Ordering
Ordering Analysis Elements
Ordering Analysis Classes
Customer
Deliverer
Delivery
Menu
Order
Payment
PlaceOrderUI
ProcessDelivery
ProcessOrder
ProcessPayment

Then, for each significant use-case realization, the purpose of this step is to define both their static and dynamic aspects. Once again, IRSA assists the architect in achieving this task by providing in the Analysis Building Blocks a use-case realization template that can be duplicated to create new use-case realizations.

The template (Figure 21) comes by default with a class diagram to record the static aspect (Participants) and a set of sequence diagrams (Basic Flow, Alternative Flow n …).

Figure 21. Use-case realization template


Order Menus
Ordering Menus Participants
Ordering Menus Realization Overview
Ordering Menus - Alternative Flow n
Ordering Menus - Basic Flow
Ordering Menus - Basic Flow

Each diagram addresses a different view of the component. The class diagram (Figure 22) depicts all the classes involved in a use-case realization while the sequence diagram (Figure 23) illustrates the interactions between them.

Figure 22. Class diagram

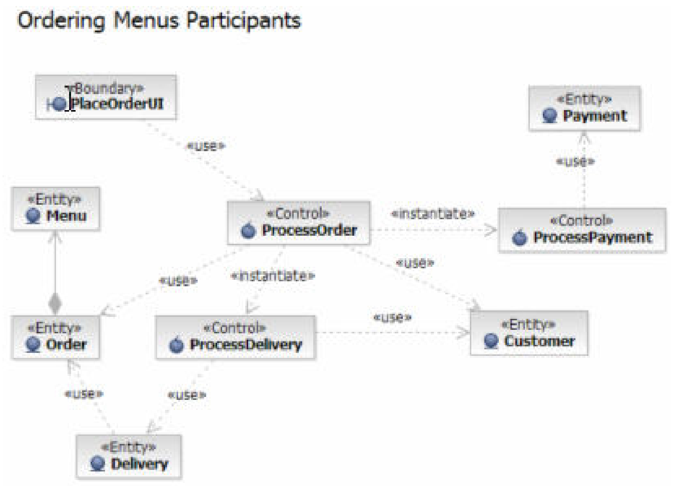
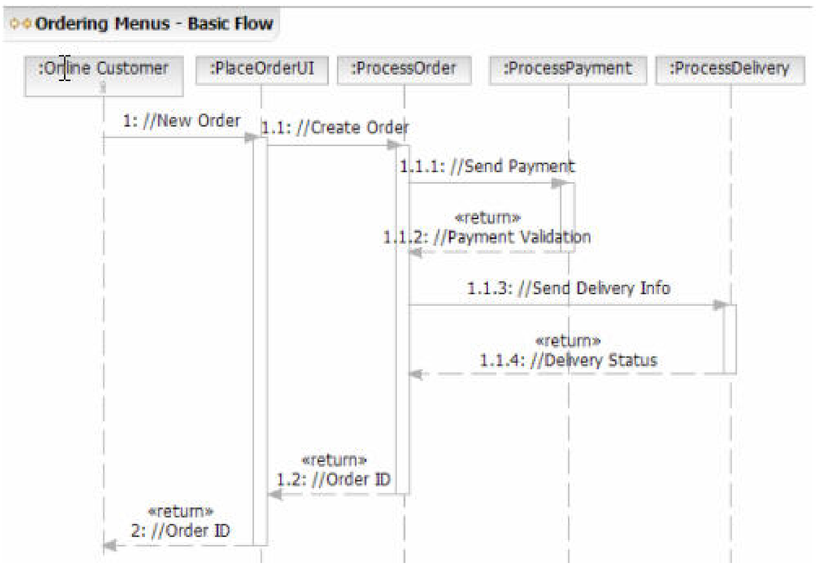


Figure 23. Sequence diagram



At the end of this step, you have developed all the significant use-case realizations of the system with UML models. It is an important part of the architecture as it depicts the structure and the behavior of the software solution.

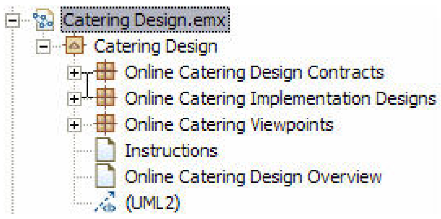
As the *Analysis Model* is still technology-agnostic, it could be reused as the starting point of several implementations.

Step 6: Create the Design Model

Once the use-case realizations have been documented in the analysis model, it is time to go one step further and to dive into the concrete design of the components.

While the *Analysis Model* is abstract, the *Design Model* must be closer to the real implementation (Figure 24).

Figure 24. Design Model


Catering Design.emx
Catering Design
Online Catering Design Contracts
Online Catering Implementation Designs
Online Catering Viewpoints
Instructions
Online Catering Design Overview
(UML2)

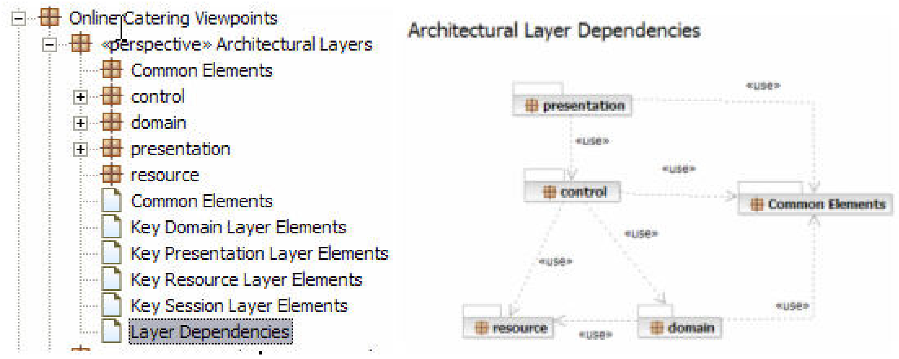
The software architect needs to identify a technology to implement the solution. In our example, Yummy Inc Online Catering will rely on the J2EE™ platform. You are then able to define some design mechanisms and rules, taking advantage of the J2EE™ technology. In the sample application, we have decided to structure the data persistence around the Data Access Object pattern (see Related topics) and to implement the application logic with Session Beans. Access to the delivery service will be done through SOAP web services.

Then, it is important to define the different layers of the solution. The J2EE™ n-tier architecture encourages the separation of concern between layers.

The *Design Model* must ensure consistency throughout the whole application. The developer will follow the rules defined here, so that each part of the system is consistent with the others.

The IRSA design template provides, in the "Architectural Layers" perspective, a view where you can define the dependencies between the layers (Figure 25).

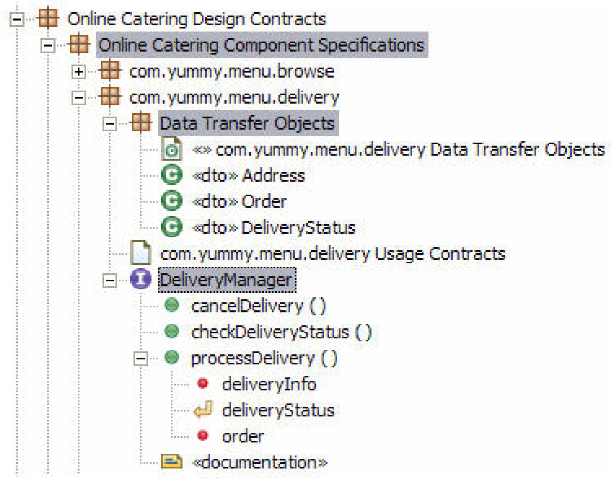
Figure 25. Architectural Layer Dependencies view



```
Online Catering Viewpoints
  «perspective» Architectural Layers
    Common Elements
    control
    domain
    presentation
    resource
    Common Elements
    Key Domain Layer Elements
    Key Presentation Layer Elements
    Key Resource Layer Elements
    Key Session Layer Elements
    Layer Dependencies
```

Finally, the software architect produces the first iteration of the design. The IRSA design template provides a pre-defined structure for this activity (Figure 26).

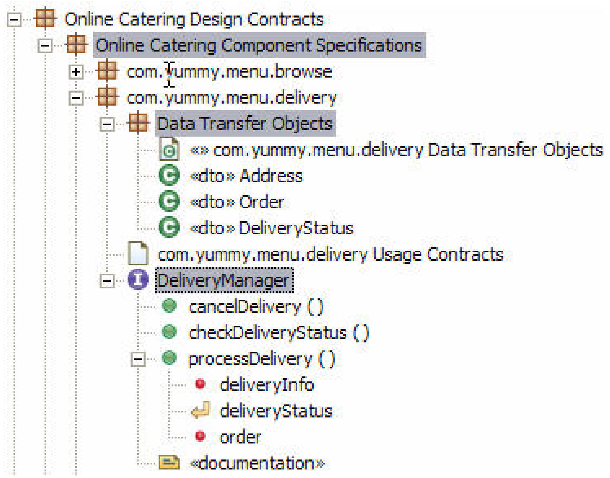
Figure 26. Design template



```
Online Catering Design Contracts
  Online Catering Component Specifications
    com.yummy.menu.browse
    com.yummy.menu.delivery
      Data Transfer Objects
        «» com.yummy.menu.delivery Data Transfer Objects
        «dto» Address
        «dto» Order
        «dto» DeliveryStatus
      com.yummy.menu.delivery Usage Contracts
      DeliveryManager
        cancelDelivery ()
        checkDeliveryStatus ()
        processDelivery ()
          deliveryInfo
          deliveryStatus
          order
        «documentation»
```

The first package is dedicated to the component specifications, so it contains the interfaces and data used to interact with the component (Figure 27).

Figure 27. Component specifications

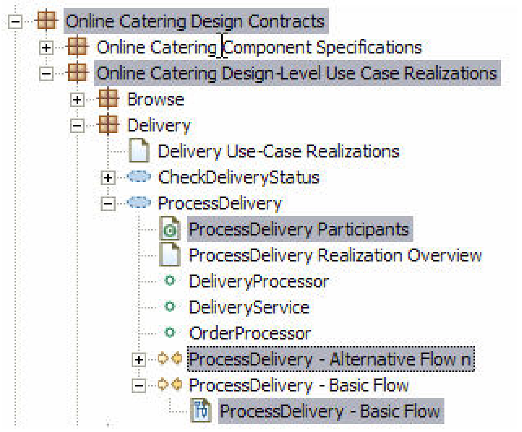


```
Online Catering Design Contracts
  Online Catering Component Specifications
    com.yummy.menu.browse
    com.yummy.menu.delivery
      Data Transfer Objects
        «» com.yummy.menu.delivery Data Transfer Objects
        «dto» Address
        «dto» Order
        «dto» DeliveryStatus
      com.yummy.menu.delivery Usage Contracts
      DeliveryManager
        cancelDelivery ()
        checkDeliveryStatus ()
        processDelivery ()
          deliveryInfo
          deliveryStatus
          order
        «documentation»
```

This is typically the first aspect a software architect will define. For each component, it is important to specify the data it manipulates (data transfer objects) and its interface, including the available operations.

The use-case realization package will be the place to design the static and dynamic views of a component using class diagrams and sequence diagrams (Figure 28).

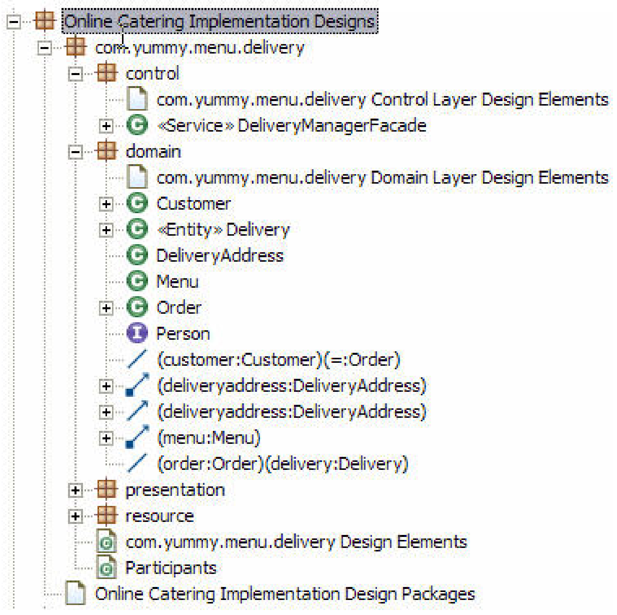
Figure 28. Use-case realization package



```
Online Catering Design Contracts
  Online Catering Component Specifications
  Online Catering Design-Level Use Case Realizations
    Browse
    Delivery
      Delivery Use-Case Realizations
      CheckDeliveryStatus
      ProcessDelivery
        ProcessDelivery Participants
        ProcessDelivery Realization Overview
        DeliveryProcessor
        DeliveryService
        OrderProcessor
        ProcessDelivery - Alternative Flow n
        ProcessDelivery - Basic Flow
          ProcessDelivery - Basic Flow
```

Then it is necessary to define the detailed design of each component. This is done within a specific package called *Implementation Designs* (Figure 29).

Figure 29. Implementation Designs package

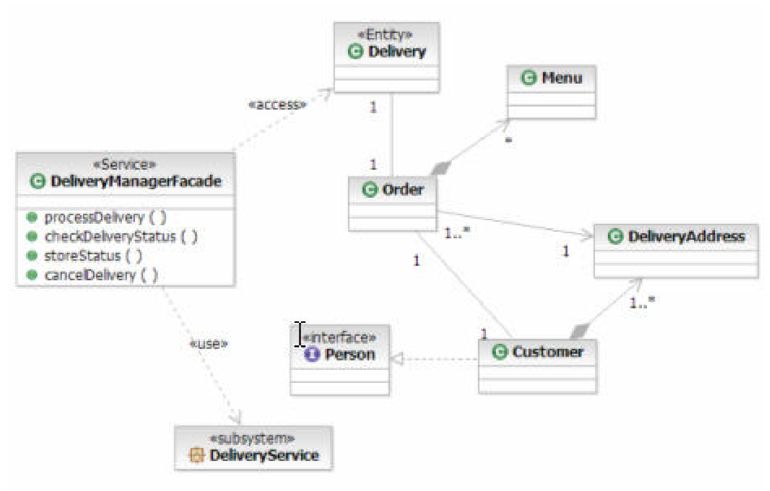


```
Online Catering Implementation Designs
  com.yummy.menu.delivery
    control
      com.yummy.menu.delivery Control Layer Design Elements
      «Service» DeliveryManagerFacade
    domain
      com.yummy.menu.delivery Domain Layer Design Elements
      Customer
      «Entity» Delivery
      DeliveryAddress
      Menu
      Order
      Person
      (customer:Customer)(=:Order)
      (deliveryaddress:DeliveryAddress)
      (deliveryaddress:DeliveryAddress)
      (menu:Menu)
      (order:Order)(delivery:Delivery)
    presentation
    resource
    com.yummy.menu.delivery Design Elements
    Participants
  Online Catering Implementation Design Packages
```

It is important here to define all the classes with their attributes and operations. This task is not always the responsibility of a software architect. Depending on your project team (size, skills…), the software designer role can be assigned to one or many different persons like senior developers (the boundary between architecture and design is sometimes fuzzy…)

At the end of this activity, you should have a class diagram for each component of the system (Figure 30).

Figure 30. Class diagram of system components



There you are. You've realized some important activities of the Elaboration phase leveraging IRSA capabilities. Of course, as RUP is an iterative process, you will have to refine all the artifacts produced as many time as necessary.

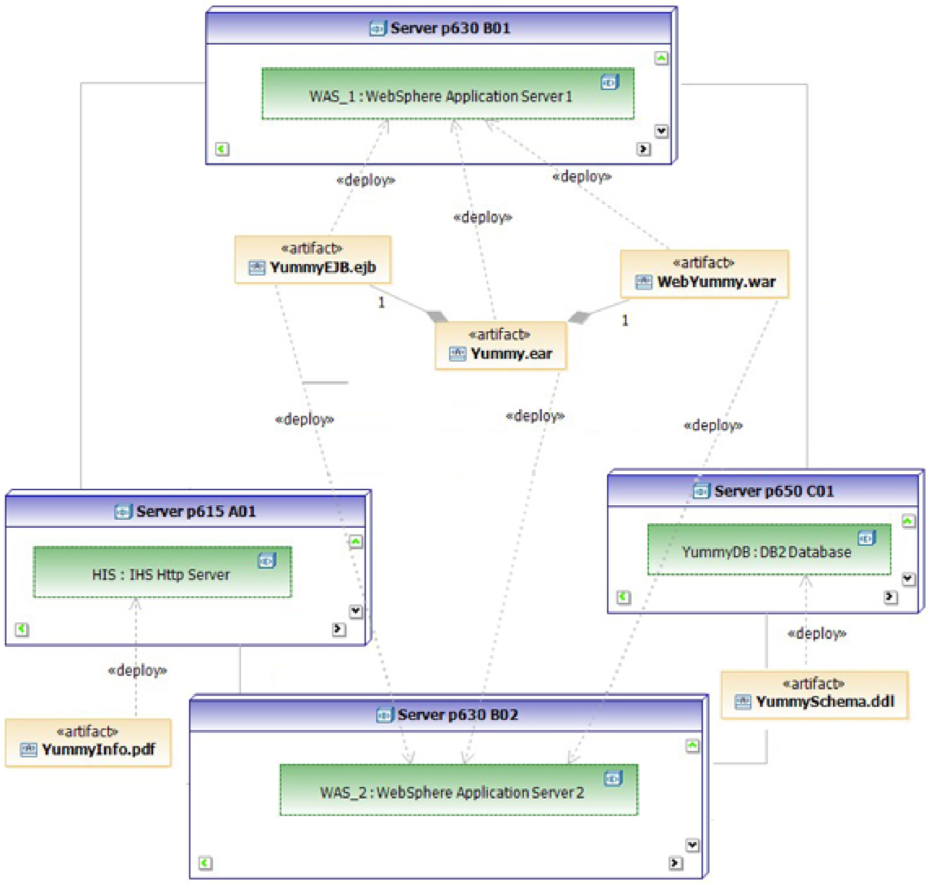
During the same phase, the software architect also completes a working prototype showcasing the architecture. This evolutionary prototype is a baseline and will be reused as a starting point in the Construction phase. I won't cover this aspect as it is far beyond the scope of this article.

Step 7: Design concurrency mechanisms

The purpose of this activity is to analyze the system distribution and concurrency mechanisms. Using several input artifacts like the *Supplementary Specification* (non-functional requirements), the software architect will document how the target architecture addresses concurrency (multi-threading, multi-instances…).

Most of the time, you will also produce a detailed *Deployment Model* (see step 3) where you can illustrate, for instance, that your solution will use a cluster with load balancing. This model can also show how J2EE™ artifacts are deployed (Figure 31).

Figure 31. Sample of detailed deployment model

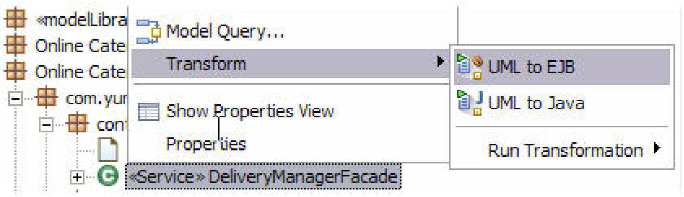


Step 8: Create the Implementation Models

The implementation model contains the source code. In a application, it consists of a set of Java classes organized into Java packages.

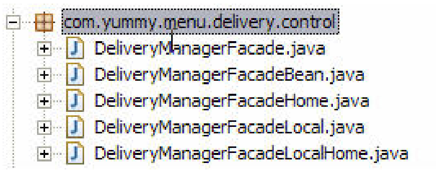
The philosophy behind MDA is to generate lower level models through transformations of more abstract ones. IRSA fully implements this approach by giving you a way to apply and create your own transformations. It also provides you with a set of out-of-the box transformations. As shown in Figure 32 below, one of them can help you to generate your implementation model from the design elements.

Figure 32. Out-of-the box transformations



This will create a Java™ project you can work with in a J2EE™ perspective (Figure 33).

Figure 33. J2EE™ implementation

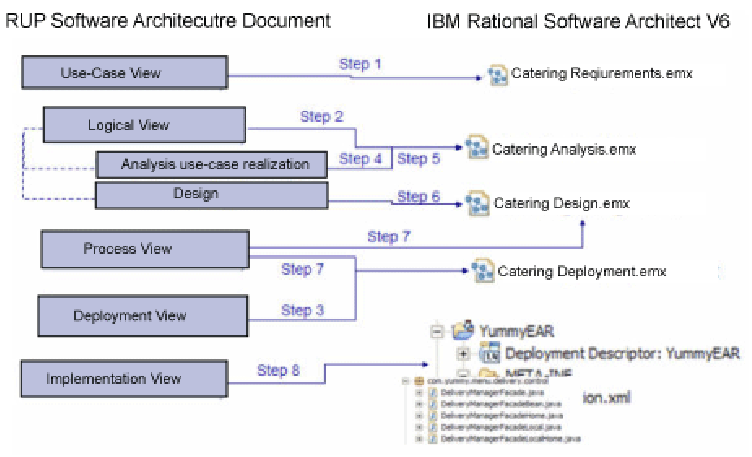


```
com.yummy.menu.delivery.control
  DeliveryManagerFacade.java
  DeliveryManagerFacadeBean.java
  DeliveryManagerFacadeHome.java
  DeliveryManagerFacadeLocal.java
  DeliveryManagerFacadeLocalHome.java
```

Summary

RUP suggests several steps to follow in order to achieve the architectural analysis of a system. The resulting artifact is a *Software Architecture Document* (see the sample Download), which summarizes all the models created during the Elaboration phase. In addition to assisting the architect in its modeling activities, IRSA provides templates to build the RUP recommended views (Figure 34) and facilitates the creation of any UML model required to produce a representation of software architecture.

Figure 34. How RUP maps to IRSA



Moreover, IRSA supports the MDA approach through features like UML model creation, or model and pattern transformation into executable code. It unifies all aspects of software design and development, enforcing a link between the RUP Elaboration and Construction phases.

Downloadable resources

Description	Name	Size
Software Arch Doc.	YummySAD.doc	10 KB

Related topics

- Introducing IBM Rational Software Architect: Improved usability makes software development easier: This article is a basic introduction to the Rational Software Architect product.
- IBM Rational Software Architect product page: Find technical documentation, how-to articles, education, downloads, and product information about Rational Software Architect
- IBM Rational Unified Process product page: Find technical documentation, how-to articles, education, downloads, and product information about Rational Unified Process
- Architectural Blueprints—The "4+1" View Model of Software Architecture: A paper by Philippe Kruchten.
- UML and MDA: The official site for information on UML and MDA.
- J2EE tutorial, Sun Microsystems: A J2EE tutorial for J2EE developers.
- IBM Rational Software Architect: Download a trial version from developerWorks.