Define application architectures with Rational Software Architect: Part 1. Envisioning the architecture

Jean-Louis Marechaux

October 11, 2011

This series presents techniques for creating models to specify and communicate the architecture of software-intensive systems. It illustrates the elaboration of the Online Catering architecture for a fictional company, Yummy Inc. Using an iterative approach, it describes the key architectural activities that are necessary to specify a software-intensive system with IBM® Rational® Software Architect (RSA). In part 1 of the series, we focus on typical tasks to outline the architecture and to align the technical vision to development needs. Part 2 will describe how the architecture is iteratively refined using RSA. Both articles assume that readers are familiar with methodologies based on iterative development.

View more content in this series

Introduction

There are many different methodologies that provide a set of customizable best practices to help businesses reliably deliver quality software. Agile software development is now mainstream and one of its core principles is to adopt iterative and incremental methods. The focus of this article is on concrete activities from a software architect perspective, which assumes that you are already proficient in agile practices and iterative development (see the Related topics section for more information about IBM agility@scale, IBM Practices, OpenUP, and Rational Unified Process).

The main purpose of this article is to illustrate how Rational Software Architect 8 (hereinafter called RSA) can assist in architectural thinking and documentation. RSA is a collaborative design platform for delivering high-quality architecture. RSA helps you define models at various levels of abstraction, and it can be leveraged to support industry best practices in software engineering.

Architectural analysis: Envisioning and evolving the architecture

Architectural analysis is a set of activities to build and improve the software architecture of a system. When conducted iteratively, this architectural thinking helps uncover and address issues during software development without requiring significant up-front architectural effort. Architectural analysis activities are crucial for every software-intensive system. Despite what dogmatic agile development coaches might say, there is no successful software development without architectural analysis.
Architectural analysis occurs at two different places. First, at the very beginning of a project, which is often referred to as iteration 0, or sprint 0. During this initial iteration, the software architect wants to envision the architecture based on significant known requirements and on architectural constraints, decisions, and objectives. Architectural envisioning is not a long and cumbersome activity. Depending on the complexity of your system, it can take days or even hours within a time-boxed iteration.

As a software architect, you typically follow these steps to outline the target architecture of your software-intensive system:

1. **Identify significant requirements:** the key functional and non-functional requirements with a significant impact on the architecture
2. **Define the candidate architecture:** the high-level architecture of the system based on architectural constraints and goals
3. **Define the initial deployment model:** the topology representing the deployment nodes of the system
4. **Define the domain model:** the key business entities and their relationships.

Once you have completed the initial technical vision, you can build your system on sound architectural foundations. In each iteration, the development team uncovers new technical challenges and new opportunities for improvement. Those playing the role of the software architect have new architectural decisions to make. It is why continuous architectural improvement is the key to software-intensive system development. Architecture is not something you create up front and leave alone. Architectural thinking is part of software development from the beginning to the end of a project.

The following architectural activities are typically performed during development iterations:

5. **Refine architectural mechanisms:** Technical concepts to satisfy architecturally significant requirements identified in the first step.
6. **Refine design elements:** The architecturally significant design elements
7. **Refine the deployment architecture:** The deployment units and topologies

The iterative activities are described in the second part of this series.
Although we do not cover review activities in this article, any experienced practitioner knows that the architecture must be validated regularly to verify that it is consistent with the requirements and the needs of the team. So you need an effective mechanism to capture and communicate the architecture to different stakeholders.

Describe architectures by using views

A proven and widely adopted approach of describing architectures is the use of views and viewpoints. A view is a representation of a whole system from the perspective of a set of concerns or interests. From the viewpoint of a stakeholder, a view addresses all the critical or important aspects of the system. As a system typically has many stakeholders, several views are needed to cover all the stakeholders concerns.

In 1995, Philippe Kruchten proposed a model for describing the architecture of software-intensive systems: the 4+1 view model of software architecture (For more information, see Related topics). Most of the concepts in the "4+1" model have been included in development processes, such as the IBM Rational Unified Process (RUP) or OpenUP. More recently, the IEEE 1471 standardized the definition of a view to address the concerns of various stakeholders of a software architecture (See the Related topics section for more information about IEEE 1471-2000 / ISO 42010).

In the original "4+1" model, five views are used to provide a comprehensive description of a system, as shown in Figure 2.
In the 4+1 model, each view addresses the concerns of a specific set of stakeholders, allowing various stakeholders to find what they need in the software architecture. The model can be tailored. Depending on your project complexity, you might only need a subset of the proposed views. The model is also extensible, so you might want to add other views to describe your architecture from a different viewpoint. Note that additional views can usually be folded into existing views as sub categories.

In Table 1, each row represents a view with its corresponding audience, area, and model.

**Table 1. Architecture views**

<table>
<thead>
<tr>
<th>View</th>
<th>Audience</th>
<th>Area</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logical</td>
<td>Designers</td>
<td>Use-case realization</td>
<td>Analysis model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Design model</td>
</tr>
<tr>
<td>Process</td>
<td>Integrators</td>
<td>Performance</td>
<td>Deployment model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Scalability</td>
<td>Design model</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Concurrency</td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td>Programmers</td>
<td>Software components</td>
<td>Implementation model</td>
</tr>
<tr>
<td>Deployment</td>
<td>Deployment managers</td>
<td>Physical nodes</td>
<td>Deployment model</td>
</tr>
<tr>
<td>Scenarios</td>
<td>Everyone</td>
<td>Functional requirements</td>
<td>Use-case model</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>User stories</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Business processes</td>
</tr>
<tr>
<td>Data</td>
<td>Data specialists</td>
<td>Data persistence</td>
<td>Data model</td>
</tr>
<tr>
<td>(optional or part of logical view)</td>
<td>Data administrators</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Note:**
Traditionally, the process view is not that important for an application because 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.

**Preparing the workspace**

Before you start RSA, make sure that you have installed the minimum set of tools to support the work of the architect. Using the Installation Manager, verify that the "Architect – Standard" option is checked, as shown in figure 3. This pre-defined profile enables the UML and topology modeling capabilities that you need to perform the tasks presented in this article.
The next step is to create a new workspace. To do it, launch RSA and specify a storage directory (Figure 4).

Figure 3. Standard architect profile

Figure 4. The Workspcae Launcher window

RSA provides many different perspectives to customize the initial set and layout of views in the Workbench window. Verify that the *Modeling* perspective is open before going further (Figure 5).
RSA 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 6).

Each template is dedicated to a specific architectural purpose. Typically, to define each aspect of the system (remember the "4+1" view model), you need the Use-Case, Analysis and Design models. In our example, we also create a sketch for the architecture vision and a topology to specify the deployment target environment. So Yummy Inc. workspace contains the following models (Figure 7). Each one is based on a different RSA template.
Figure 7. Models that describe the architectural views for the Online Catering system

Note that we could have decided to use other types of models available in the RSA templates, such as the BPMN model (business process) or the Services model (SOA).

Now that your RSA workspace is ready, you need to perform several activities that we mentioned previously. Let's illustrate how RSA can assist you in your step-by-step architectural analysis.

Envision the architecture

As a software architect, you need to envision the initial architecture and outline the architectural decisions that guide development and testing. This effort relies on gathering experience gained in similar systems or problem domains to constrain and focus the architecture. Your conclusions should yield information to communicate the architecture to the team.

Identify significant requirements

One of the first tasks you will be doing during the architectural envisioning is to identify the requirements that are significant for the architecture. As an architect, your role is to ensure that the target architecture is appropriate to realize user needs. So you need to review the use-cases, the business processes, or the user stories to identify the ones that may have a significant impact on the architecture. Experienced architects work closely with project managers or product owners to educate them about the architectural impacts of items in the backlog. They influence how the backlog is prioritized to address technical uncertainties as soon as possible.

In RSA, the Use-case model contains the identified requirements for Yummy Inc. (Figure 8).
There is no universal rule to identify significant requirements. It depends on your environment, your technical framework, and your project team. Requirements with a huge impact on the architecture for a team can seem trivial for another team. A requirement to display a simple list of items will probably not influence your architecture too much. But if the requirement is to obtain the items for a business partner through an asynchronous service call, you must make sure that you have the plumbing in your architecture to support that need.

In our Yummy Inc. example, 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 the requirement (Figure 9). Of course, the same information could have been captured in a business process diagram or a user story.
Define the candidate architecture

After the significant requirements have been identified, the software architect creates an overview of the architecture. Nowadays, we rarely develop systems from scratch. We enrich existing applications, we modernize legacy systems, and we reuse and assemble assets. The architect leverages past experience with similar systems and makes sure that goals and constraints are considered. The candidate architecture takes into account the functional requirements (use-cases, stories, business processes) but also the non-functional requirements (availability, performance, scalability…) of the system.

In our example, we have chosen an N-tier architecture (Figure 10) where the application is web-based, but is also accessed through web services from different client types (see Related topics for more information about the multi-tiered applications)

Note that this technology diagram (Figure 10) created in RSA is quite simple at this stage, and can be completed in a short brainstorming session. The sketch shows the major components involved and the technology stack identified to develop the solution.
Define the initial deployment model

Using the overview of the architecture sketched before, a software architect can now draw the big picture of the deployment model. It depicts the major software and hardware components and how they interact at a high level. The deployment diagram takes into account constraints of the deployment environment, and is a great communication vehicle for the development and infrastructure teams to share information between groups.

The UML specification provides a set of elements to define deployment models. However, the deployment part of UML has proven to be limited in terms of capabilities. As a result, it has not been widely adopted by the industry, even among intensive UML users. RSA provides a rich set of tools to define deployment topologies (logical, physical, and deployment instances). RSA topologies are really powerful where the UML deployment models fall short: Simplicity, reuse, and traceability (see the Related topics for the further information about deployment topologies)

The deployment diagram (Figure 11) shows the different hardware components required to host the application. RSA comes with a bank of images to make your diagram more meaningful, and allows you to add your own images to represent specific model elements.

Figure 11. Deployment nodes for the Online Catering system
Define the domain model

For business applications, an initial domain model helps the development team understand the key business entities and their relationships. The software architect can obtain the information from a variety of sources like functional requirements, stories or business processes.

In RSA, a simple way to capture the business entities is by leveraging the analysis profile. This profile contains three stereotypes that you can apply to UML elements (Figure 12).

**Figure 12. UML stereotypes**

![Boundary, Control, Entity stereotypes](image)

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.

At this stage, you only care about the data so only the entity stereotype is used. It is important to capture the overall vocabulary of the system to implement. It is the starting point of the Analysis Model that you create in the next step.

The Analysis Model template provided by RSA is pre-structured and comes with a special package named **Perspective overviews** to collect information related to key concepts (Figure 13).

**Figure 13. Perspective overviews for the Online Catering system**

![Perspective overviews](image)

As a software architect, you can use a simple class diagram (containing the key business entities) to capture the first draft of domain model using analysis stereotypes (Figure 14).
Other options are available to define your domain model in RSA. First, you can choose to create a simple sketch representing the business entities (Figure 15).

**Figure 15. Domain model sketch for the Online Catering system**

In RSA, sketches elements can be converted to UML elements when you need a more formal notation, and you keep traceability between the generated UML and the initial sketch. Another option is to create a data diagram. In RSA, you can create tables, columns and keys in a diagram representing your data, or you can connect to your database to import an existing schema. Which option to choose to define you domain model really depends on your environment and on the way the project team works. There is no perfect solution but there is surely one that is best for you.
Summary

At this stage, you have defined the vision for the architecture of your software-intensive system. You know which views are useful to your stakeholders, you have identified the requirements with a significant impact on your technical solution, and you have outlined the target architecture accordingly. In a short time box (iteration 0), you have captured in RSA the key information related to technologies, deployment and domain models.

In part 2 of this series, you will see how to use RSA to refine the architecture of your software-intensive system.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yummy RSA models</td>
<td>Yummy2011.zip</td>
<td>84KB</td>
</tr>
</tbody>
</table>
Related topics

- Learn about modeling with Rational Software Architect in this [free self-paced training](http://ibm.com/developerWorks/).
- Learn about the [Design Management project](http://ibm.com/developerWorks/), a new solution to help team collaborate on architecture and design activities.
- Learn about [OpenUP](http://ibm.com/developerWorks/), the open source process that applies iterative and incremental approaches within a structured lifecycle.
- Watch a [short video](http://ibm.com/developerWorks/) to learn more about IBM agility@scale.
- Read this article to learn how to create deployment topologies with Rational Software Architect, [Deployment topologies with Rational Software Architect](http://ibm.com/developerWorks/).
- Read [Architectural Blueprints—The "4+1" View Model of Software Architecture](http://ibm.com/developerWorks/), a paper by Philippe Kruchten.
- Explore the [official site](http://ibm.com/developerWorks/) for information about UML.
- Download a trial version of IBM Rational Software Architect from developerWorks.
- [Evaluate IBM software](http://ibm.com/developerWorks/).

© Copyright IBM Corporation 2011

[Trademarks](http://ibm.com/developerworks/ibm/trademarks/)