The Rational Unified Process for Systems Engineering
PART II: Distinctive Features

by Murray Cantor
Principal Consultant
Rational Software Corporation

In Part I of this article, published in last month's issue of The Rational Edge, I introduced RUP SE, my technique for applying the Rational Unified Process® (RUP®) to address the specific needs of systems engineering projects. RUP SE is not a product but a deployment service provided by qualified Rational consultants. In Part I, I talked about the similarities and differences between RUP SE and the common RUP implementations, and began explaining in more detail some of RUP SE's innovative constructs. In this installment I'll continue that explanation, focusing first on requirements analysis in RUP SE. I'll also explain in general terms how RUP SE impacts the management of systems development projects.

Requirements Analysis in RUP SE

RUP SE follows the Unified Modeling Language (UML) and RUP in distinguishing two types of system requirements:

- **Use cases**, which describe services provided by the system to its actors. Use cases capture the system functional requirements, and may also include associated performance requirements.

- **Supplementary requirements**, which cover non-functional elements like reliability and capacity.

A critical goal for any successful systems engineering project is to specify a set of system use cases and supplementary requirements that, if met, would result in a system that accomplishes its business purpose.
The purpose of requirements analysis is to determine requirements for all the Analysis level architectural elements. The basic process for deriving requirements for these elements is well known, and common to several systems analysis methodologies:

- Determine the requirements for a given model, such as the business model.
- Decompose that model into elements, and assign roles and responsibilities to the elements.
- Study how the elements collaborate to carry out the model requirements.
- Synthesize the analysis of these collaborative interactions to determine requirements for the elements themselves.

In the case of the business model, for instance, the RUP SE method for deriving system requirements is to partition the business into the system and its actors. The next step is to study the means by which the system and its actors collaborate to meet the business requirements. This, in turn, enables you to extrapolate system requirements. The discussion below explains more about how this basic methodology is applied to systems.

**More About Model Levels**

For modeling purposes, any system architecture exercise is predicated on levels of specification. As the architecture is developed, it evolves from a generalized, low-detail specification to a more completely described, detailed specification. Like the RUP, RUP SE defines four architectural models:

- The Business Model expresses the business processes that the system supports.
- The Analysis Model reflects the initial partitioning of the system into its primary elements, based on what it needs to accomplish and how that effort should be distributed.
- The Design Model expresses the realization of the Analysis Model in terms of hardware, software, and people.
- The Implementation takes the Design Model to the level of specific configurations.

**Use-Case Flowdown**

In the RUP SE Analysis Model level, the system architecture elements we're concerned with are subsystems, localities, and processes. The RUP SE activity for deriving functional requirements (i.e., use cases) for these analysis elements is what I call *use-case flowdown* -- a key area of departure from the RUP. The outcomes of this activity are:

- A use-case survey for subsystems.
- A survey of hosted subsystem use cases for localities.
- A survey of realized subsystem use cases for processes.
I begin the use-case flowdown activity with the familiar RUP activity of choosing an architecturally significant set of use cases, and then describing the interactions between the system actors and the system for each chosen use case. In this context, the system's responses to the actions of the actors are "black box"; that is, the descriptions of what happens make no reference to the architectural elements.

Table 1 shows a sample flow of events for making a sale in a retail store. Note that each step has an associated performance requirement.

**Table 1: Sample "Black Box" Flow of Events**

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor Actions</th>
<th>Black Box Description</th>
<th>Black Box Budgeted Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This use case begins when the Clerk pushes the New Sale button.</td>
<td>The system brings up the New Sale, Clerk, and Customer screens, and enables the scanner.</td>
<td>Total response time is 0.5 second.</td>
</tr>
<tr>
<td>2</td>
<td>The Clerk scans the items and enters the quantity on the keyboard.</td>
<td>For each scanned item, the system displays the name and price.</td>
<td>Total response time is 0.5 second.</td>
</tr>
<tr>
<td>3</td>
<td>The Clerk pushes the Total button.</td>
<td>The system computes and displays on the screen the total of the item prices and the sales taxes.</td>
<td>Total response time is 0.5 second.</td>
</tr>
</tbody>
</table>
| 4    | The Clerk swipes the credit card. | This use case ends when the system validates the credit card, and, if the card is valid,  
|      |                             |   ● Prints out a receipt;  
|      |                             |   ● Updates the inventory;  
|      |                             |   ● Sends the transaction to accounting;  
|      |                             |   ● Clears the terminal.  
|      | If the credit card is not valid, then the system returns a rejected message. | Total response time is 0.5 second. |

In the next steps, you create the subsystem and process diagrams via standard Object Oriented Analysis and Design (OOAD) techniques. The Locality diagrams are found by similar techniques, except that the team specifies the major processing elements as discussed last month in Part I. You'll refine and re-factor the initial Analysis model throughout the flowdown process, as discussed below. For this reason, you can think of the initial model as a starting point, and you should not be overly concerned with its correctness.

**The White Box View**
Once the initial subsystem, locality, and process diagrams are in place, the subsequent steps depart from RUP activity. At this point, RUP SE revisits the interactions between the system and its actors by specifying how the analysis elements participate in carrying out the use case. As this version of the flow of events relates to specific architectural elements, I call it a "white box" view.

Table 2 shows a sample white box flow of events that parallels the steps in Table 1 above, adding a great deal of information:

**Table 2: Sample "White Box" Flow of Events**

<table>
<thead>
<tr>
<th>Step</th>
<th>Actor Actions</th>
<th>Black Box Description</th>
<th>Black Box Budgeted Requirements</th>
<th>Subsystem White Box Description</th>
<th>White Box Budgeted Requirements</th>
<th>Locality</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>This use case begins when the Clerk pushes the New Sale button.</td>
<td>The system brings up the New Sale Clerk and Customer screens, and enables the scanner.</td>
<td>Total response time is 0.5 second.</td>
<td>The Point-of-Sale Interface clears the transaction, brings up new sales screens, and requests that Order Processing start a sales list.</td>
<td>1/6 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Order Processing starts a sales list.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Point-of-Sale Interface captures the bar from the scanner. The Point-of-Sale Interface requests that Order Processing retrieve the name, price, and taxable status for the scanned data.</td>
<td>1/8 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
</tr>
<tr>
<td>2</td>
<td>The Clerk scans the items and enters the quantity on the keyboard.</td>
<td>For each scanned item, the system displays the name and price.</td>
<td>Total response time is 0.5 second.</td>
<td>Order Processing retrieves the name, price, and taxable status for the scanned data.</td>
<td>1/8 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
</tr>
<tr>
<td>Step</td>
<td>Description</td>
<td>Time</td>
<td>Component</td>
<td>Function</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
<td>------</td>
<td>-----------</td>
<td>----------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>The Clerk pushes the Total button.</td>
<td>1/8 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Point-of-Sale Interface displays the item name, price, quantity, and item total on the clerk and customer screens.</td>
<td>1/8 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>The system computes the total price of the items and sales taxes and displays the total on the screen.</td>
<td>Total response time is 0.5 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Point-of-Sale Interface requests that Order Processing sum the price and compute the taxes.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Order Processing sums the price and computes the taxes.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Point-of-Sale Interface displays the totals.</td>
<td>1/6 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>The system validates the card, prints two copies of the credit card receipt, and closes out the sale.</td>
<td>30 seconds</td>
<td>Point-of-Sale Terminal</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>The Point-of-Sale Interface reads the credit card data and requests that Credit Card Services validate the sale.</td>
<td>.5 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Credit Card Services requests validation through Credit Card Gateway for the given card number and amount.</td>
<td>28 seconds</td>
<td>Store Processor</td>
<td>Sales Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The Clerk swipes the customer credit card.

<table>
<thead>
<tr>
<th>Step</th>
<th>Time</th>
<th>Locality</th>
<th>Process</th>
</tr>
</thead>
<tbody>
<tr>
<td>If the sale is approved, then the Point-of-Sale Interface prints a receipt for signature.</td>
<td>1 second</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
</tr>
<tr>
<td>The Point-of-Sale Interface requests that Order Processing complete the sale.</td>
<td>1/6 sec</td>
<td>Point-of-Sale Terminal</td>
<td>Terminal</td>
</tr>
<tr>
<td>Order Processing requests that Inventory Control remove the items from inventory.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
</tr>
<tr>
<td>Inventory Control removes the items from inventory.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Store Accounting</td>
</tr>
<tr>
<td>Order Processing requests that Accounting Services post the transaction.</td>
<td>1/6 second</td>
<td>Store Processor</td>
<td>Sales Processing</td>
</tr>
<tr>
<td>Accounting Services updates the account.</td>
<td>1/6 second</td>
<td>Central Office Processor</td>
<td>Central Accounting</td>
</tr>
</tbody>
</table>

The purpose of the subsystem white box steps shown in Table 2 is to illustrate how the subsystems collaborate to carry out each black box step. The white box budgeted requirements map the budgeting of the black box performance requirements (see Table 1) to the white box steps. The Locality is the locality that hosts each white box step; the Process specifies which process executes the white box step.

If a white box step requires more than one hosting locality or executing process, then you can simply break the step into smaller steps, each with a unique locality and process.

When you assign white box steps to subsystems, localities, and processes, you make a series of design decisions, each of which helps flesh out the role that each analysis element plays in the overall system design. As your team makes these decisions, you may decide to re-factor the design, shifting responsibilities from one element to another within a given diagram.
The Subsystem Use Case Survey

Once you create a white box flow of events, the next step is to specify the subsystem use cases. You initiate this process by organizing the white box steps according to the subsystems they relate to. Then you sort the white box steps associated with each subsystem according to how they relate to one another. The result is a survey of use cases for each subsystem. (Recall that subsystem use cases specify what processing occurs at a given locality.)

Table 3 illustrates a subsystem use-case survey. Note that the survey includes both the locality that hosts each process, as well as the process that executes each white box step. This enables you to sort your subsystem use cases by locality or process, once you’ve completed the survey.

<table>
<thead>
<tr>
<th>Subsystem Use Case</th>
<th>Description</th>
<th>Locality</th>
<th>Process</th>
<th>System Use-Case Name</th>
<th>White Box Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiate Sales List</td>
<td>The subsystem initiates a list of items to be included in the sales transaction.</td>
<td>Store Processor e-commerce server</td>
<td>Sales processing</td>
<td>Enter a sale</td>
<td>Order Processing starts a sales list.</td>
</tr>
<tr>
<td>Add Product Data</td>
<td>The subsystem adds an item to a sales list when requested by the actor.</td>
<td>Store Processor e-commerce server</td>
<td>Sales processing</td>
<td>Enter a sale</td>
<td>The e-commerce interface requests Order Processing to instantiate an ordering list and add the item to the list.</td>
</tr>
<tr>
<td>Compute Total</td>
<td>...</td>
<td>Store Processor e-commerce server</td>
<td>Sales processing</td>
<td>Enter a sale</td>
<td>Order Processing sums the price and computes the taxes.</td>
</tr>
</tbody>
</table>
A survey that delineates the use cases hosted at each locality is valuable because it expresses what computing occurs at each locality, along with associated performance requirements. This information helps you specify the hardware components to be deployed at each locality. Likewise, a survey of use cases that each process executes helps you specify the software components you'll need at each locality.

**Creating Collaboration Diagrams**

The text descriptions of the white box flow of events (see Table 2) also have an important purpose: They form the basis for a series of collaboration diagrams. These diagrams visually convey the traffic between the analysis elements. The objects in each collaboration diagram are proxy diagram elements. The messages that connect the objects represent the subsystem use cases.

Figures 1 and 2 show the subsystem and locality collaboration diagrams for the flow of events represented in Table 2. Figure 2 provides insight into how the subsystems interact. This gives you a very useful way to evaluate your subsystem design. If there is considerable traffic between a pair of subsystems, for instance, then it might make sense to combine them.
Figure 1: A Sample Subsystem Collaboration Diagram

Figure 2, on the other hand, tells you what data must flow between the localities. From this viewpoint you can better determine how the localities should communicate in terms of protocols, throughput rates, etc., with one another.
Determining Supplementary Requirements

Fear not! I haven't forgotten about supplementary requirements. These are handled as part of the analysis process, in the context of an initial locality diagram that the system architects develop. This locality view provides a context in which you can begin looking at the system's non-functional considerations,
such as reliability, capacity, and permitted failure rates, as part of standard engineering practice.

This analytical effort results in a set of derived supplementary requirements for each locality element. These requirements form the basis for determining your locality characteristics.

When you complete the steps we outlined above, you will have done enough Analysis to begin designing the system. Remember that in an iterative development process, however, you may need to retrace these steps at a later time to adjust your locality requirements.

**Component Specification: Moving from Analysis to Design**

When you begin to specify the design of hardware and software components, you move from the Analysis level of the architecture to the Design level. You can now determine hardware components by analyzing the localities, their derived requirements, and their hosted subsystem use cases. Your goal is to produce a series of descriptor node diagrams (see Figure 1) that specify the components, servers, workstations, workers, etc., for each locality.

Note that this diagram does not specify the technologies that will be used to implement the components. You make those decisions by looking at cost/performance/capacity tradeoffs, which the descriptor diagrams help make more evident. Many systems will ultimately have more than one hardware configuration, each designed to balance these tradeoffs differently.

Figure 3 shows the descriptor view derived from a locality diagram shown in Part I of this series.
In RUP SE, you determine software components by specifying a set of object classes, and then compiling and assembling the code associated with those classes into executable files. A complete software component design must reflect a wide range of concerns, such as the locality where each component will run, and the hardware that will host each component. (Hence, you need to specify hardware components before you can fully specify software components.) The information required to specify software components is derived from several sources, including the survey of hosted subsystem use cases for localities and the surveys of executed use cases for processes.

The culmination of your Design efforts is a clear understanding of the hardware components, software components, and worker roles you'll need to implement the various system configurations. Now you're ready to move to the Implementation model level, where you begin to choose specific technologies to implement your design: What server platform? What database application? And so on. At the Implementation level, a single deployment diagram describes the hardware and software components of each system configuration.

**RUP SE and Project Management**

Project management is where the rubber of RUP SE meets the road of your organization. RUP SE impacts project organization and system development, integration, and testing in many of the same ways that RUP does, but with changes and additions that reflect the complexity of systems engineering projects.
On a typical RUP SE project, the organization is made up of several development teams, each with a project manager and technical lead:

- The **enterprise modeling** team analyzes the business case for the project and generates business models.
- The **system architecture team** works with the enterprise modeling team to create the system context and derive system requirements.
- The **project management team** looks after typical project issues such as reviews, resource planning, budget tracking, etc.
- The **integration and test team** receives each iteration's code and hardware components from the development team; builds the software components; installs the hardware and software components in a controlled setting; and conducts system tests.
- The **subsystem development teams** each design and implement one or more subsystems.
- The **hardware development and acquisition team** is responsible for the design, specification, and delivery of the physical systems, based on the localities.
- The **deployment operations and maintenance team** handles operational issues and liaises with users.

**Concurrent Design and Implementation**

Because it allows you to break systems down into subsystems and localities and their derived requirements -- any of which can be the focus of concurrent design and development -- RUP SE can scale to handle even the largest projects. Subsystems can be assigned to separate development teams, for instance; localities can, in parallel, be assigned to hardware development teams. Each team works from the appropriate use-case survey to develop their part of the design model and implementation models. In this way, the design and implementation of various design elements can proceed in parallel.

**Iterative Development, Integration, and Testing**

The iterative project lifecycle driven by RUP SE and RUP differs significantly from the serialized (a.k.a. *waterfall*) process typical of many organizations. With the iterative approach, the system is integrated and tested at each iteration; and each iteration adds functionality. The final iteration yields a fully tested system ready for transition to the operational setting.

When teams use a waterfall, or serial activities approach, workers are typically assigned to a project until their artifacts are complete. Engineering staff, for example, might complete the specifications, hand them off to the developers, and then move on to another project.

By contrast, no such handoff occurs in RUP-based projects. Instead, the specifications (and other artifacts) continue to evolve throughout the project lifecycle. Therefore, staff responsible for artifacts such as the requirements database and UML architecture will be assigned to the Development phase for its duration.
The content of an iteration, as captured in the RUP SE system iteration plan, is determined by the use cases and supplementary requirements for the components slated for development in the iteration. Each iteration is tested by an appropriate subset of system test cases. Subsystems and localities have derived use cases that form the basis for derived iteration plans.

In an iterative lifecycle, the role of the testing organization differs from its role in a traditional, serialized lifecycle. Rather than spending most of their available time planning for an overall system integration effort at the end of the lifecycle, the testing team spends time on integrating, testing, and reporting defects for each iteration.

More Specific Best Practices

RUP SE embodies all the most important fundamental RUP parameters: the four-phase lifecycle; the process disciplines; the iterative development approach; and the use of UML for visual modeling. This enables RUP SE to deliver all the advantages of RUP best practices, while providing a sound methodology for addressing system engineering issues. The most compelling benefits of RUP SE for system development teams include:

- A common modeling language and a unifying process support infrastructure for ongoing, evolving collaboration among business analysts, system architects, engineers, software developers, hardware developers, and testers.
- Comprehensive, visual perspectives, in the form of models, views, and diagrams, which make it possible to continuously verify and address system quality issues in the context of a component-centric process.
- The ability to visually model systems, thanks to the inclusion in RUP SE of UML artifacts for systems architecture and specification.
- Scalability from small systems to the largest systems engineering projects.
- Support for concurrent design and iterative development of hardware and software components.

I believe that RUP SE, like RUP, will help teams get a better handle on complex, evolving requirements; discover and mitigate development risks earlier; reuse more components; improve and ensure system quality; reduce project costs; and compress project timeframes (if only through better communication!). In short, it enables systems engineering projects -- and the organizations that depend on them -- to succeed.

Notes

1 An actor is any external entity that interacts with the system, such as a user or another system.
For more information on the products or services discussed in this article, please click here and follow the instructions provided. Thank you!