

Rational Unified Process[®] for Systems Engineering

RUP[®] SE1.0

A Rational Software White Paper
TP 165, 8/01

Table of Contents

INTRODUCTION	1
BUSINESS MODELING	3
SYSTEM ARCHITECTURE	4
SYSTEM ARCHITECTURE DIAGRAMS.....	5
LOCALITY	8
RELATION TO 4+1 ARCHITECTURE MODEL	11
REQUIREMENTS ANALYSIS	11
DERIVED REQUIREMENTS.....	11
USE-CASE FLOWDOWN	12
SUPPLEMENTARY REQUIREMENTS FLOWDOWN.....	20
COMPONENT SPECIFICATION.....	20
SYSTEM DEVELOPMENT	22
PROJECT ORGANIZATION	22
CONCURRENT DESIGN AND IMPLEMENTATION	24
ITERATIVE DEVELOPMENT, INTEGRATION, AND TEST.....	24
CONCLUSION.....	25

Introduction

A system is an assemblage of components used by an enterprise to carry out a business purpose¹. System components consist of hardware, software, and workers. Systems are specified by the components, their attributes, and their relationships. The problem of systems engineering is to design and implement a system that meets a broad set of requirements. Here is a partial list:

- **Function** – support the system provides the users and other systems to enable them to carry out their role in meeting the business need. Functional requirements should include the behavior the system exhibits as it provides the functionality
- **Usability** – ease of access to system function
- **Maintainability** – ease of discovery, isolation, and removal of defects
- **Extendability** – ease of adding function
- **Scalability** – ability to support number of users, data items
- **Reliability** – probability of a correct system response, possibly including safety concerns
- **Performance** – expected response time of the system to a step in a use case under capacity loads
- **Capacity** – expected number of users, data items
- **Supportability** – ease of service in the field, including acceptable down time
- **Manufacture, deployment cost**
- **Operational cost**

Depending on circumstances, there might be other system requirements such as logistics support, security, and remote training needs.

Some of these requirements are familiar to software development. Some cannot be addressed without hardware, software, and worker considerations. Systems design requires that all three types of components be specified concurrently.

A systems developer may want to maintain a number of system configurations. These systems configurations would have common architectures but different hardware or software deployments that meet different requirements tradeoffs such as cost/performance.

The system problem then differs from the software-only problem in that systems engineering addresses a broader set of requirements than are normally addressed in software efforts. Even so, it is important to note that almost all software development efforts contain some elements of the system problem. Examples of software developments that have system concerns include web-based applications, business applications, information technology integrations, and embedded software, as well as defense and intelligence systems.

This paper introduces a derivative of the Rational Unified Process, or RUP,² that addresses the problem of system specification, analysis, design, and development.

As a derivative of RUP, RUP SE consists of new artifacts as well as modifications of RUP disciplines and roles to support the creation of those artifacts.

¹ Blanchard and Fabrycky, *Systems Engineering and Analysis* (Third Edition), Prentice Hall, 1998.

² Kruchten, Philippe, *The Rational Unified Process, An Introduction* (Second Edition), Addison Wesley, 2000.

This paper provides an overview of:

- RUP principles that are maintained in RUP SE
- RUP SE requirements models
- The UML-based artifacts for system architecture modeling
- The workflows for creating the artifacts

RUP SE is delivered as a deployment package providing assistance to customers wishing to deploy RUP in Systems Engineering projects. Contact the local Rational account team for more information.

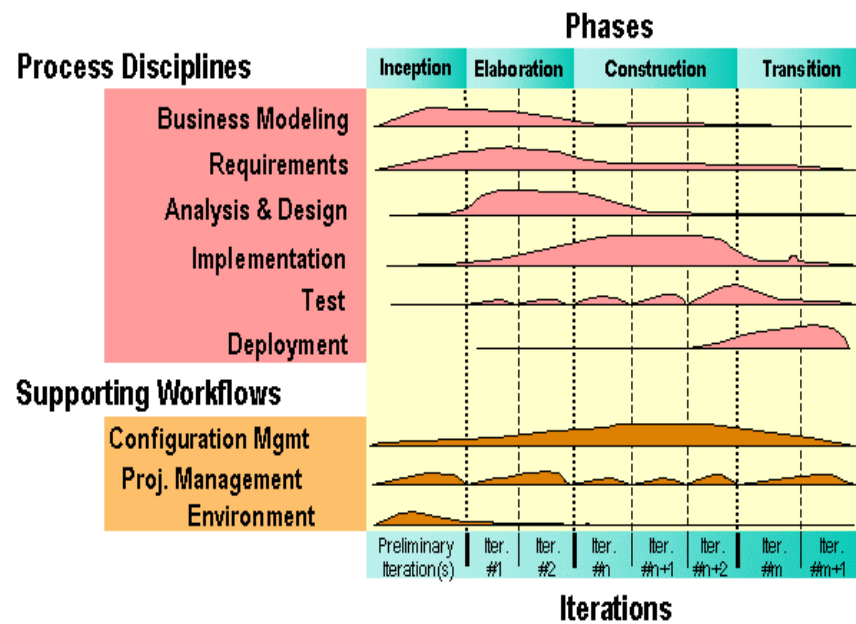


Figure 1: The Rational Unified Process

RUP is shown in Figure 1. RUP SE follows RUP in these ways:

- Lifecycle – the four phases based on the team’s evolving understanding and development of the project details
- Disciplines – the main focus areas of effort carried out by the team in developing the system. While the project team has systems engineers as members, there is no separate systems engineering discipline. Rather, the systems engineers participate in RUP disciplines.
- Iterations – RUP SE uses a series of system builds based on risk identification and mitigation. A key feature that RUP SE inherits from RUP is a rejection of waterfall development and the use of iterated development.
- Use of UML for visual modeling – RUP SE includes a set of UML artifacts suitable for system architecture and specification.

One key feature of RUP and RUP SE is that the development team consists of workers such as architects, developers, testers, and others who concurrently evolve their particular artifacts. These workers do not hand off work to each other using a serial approach. They work together throughout the effort, evolving levels of detail to address their areas of concern. In RUP SE, this idea is carried forward, adding systems engineers to the mix. Their area of concern is the design and specification of the hardware and system deployment to ensure that the overall system requirements are addressed.

In addition to adequacy of the software architecture to meet functional requirements, software architects are generally concerned with:

- Usability – ease of accessing the system functionality
- Maintainability – ease of isolating and removing defects without introducing others
- Extendibility – ease of adding new functionality to an existing software product
- Besides functionality, systems engineers or designers usually address the following types of concerns:
- Availability/reliability – the likelihood that the system will be available and respond correctly to input
- Performance – responsiveness of the system to some input
- Capacity – the number of items such as users or data records that the system can handle
- Scalability – the ease of increasing capacity
- Supportability – the ease of providing support in the field. Supportability can include installing the system and applying patches.

Other domain-specific systems engineering concerns include security, ease of training, and logistics support. RUP SE provides the artifacts for addressing these concerns and the workflows for evolving their detailed specification.

Business Modeling

Following Blanchard and Fabrycki’s definition, it is important when architecting a system to understand the business purpose it serves. It is not surprising that understanding and modeling the business that will use the system is crucial to RUP SE or any other systems engineering process. The system requirements rely on a solid understanding of the business activities.

RUP SE does not include changes to the business modeling discipline. However, for the business model to provide adequate information to support the determination of system requirements, it should include business use cases¹ with the associated identification of business actors and flow of events. These flows of events can be swimlane activity diagrams that show how the entities of the business collaborate to carry out the use case.

System Architecture

There are two dimensions to system architecture:

- Viewpoint – the context for addressing a limited set of quality concerns
- Model level – UML models that capture various levels of design specificity

The different viewpoints allow for separation of concerns. Table 1 outlines viewpoints and associated concerns. The viewpoints align with those found in ISO standard ISO/IEC 10746-1: Reference Model – Open Distributed Processing (RM-ODP)². The framework provides a set of viewpoints as expressed in Table 1.

<i>Viewpoint</i>	<i>Expresses</i>	<i>Concern</i>
Enterprise	Business activities supported by the system	Business context of system is adequately understood to support system specification.
Computation	Logical decomposition of the system as a set of class objects/subsystems	System functionality is adequate to realize use cases. System is extendible and maintainable. Internal reuse Good cohesion and connectivity
Engineering	Mechanisms to support distribution of functionality	System physical characteristics are adequate to host functionality and meet supplementary requirements.
Information	Data managed by the system	System has sufficient capacity to store data. System has sufficient throughput to provide timely access to the data.
Process	Threads of control, which carry out the computation elements	System has sufficient partitioning of processing to support concurrency and reliability needs.

Table 1: Common System Architecture Viewpoints

The viewpoints in Table 1 are some of the most common for software-intensive systems. Many system architectures also require additional, domain-specific viewpoints. Examples include safety, security, and mechanical viewpoints. Viewpoints represent different areas of concern that must be addressed in the system architecture and design. If there are system stakeholders or experts whose concerns are important to the overall architecture, there is likely to be a need for a set of viewpoint artifacts to capture their design decisions.

¹ Several good texts provide more information on business use cases. See *Writing Effective Use Cases* by Alistair Cockburn (Addison Wesley, 2001) or *Enterprise Modeling with UML* by Chris Marshall (Addison Wesley, 2000).

² Putman, Janis, *Architecting with RM-ODP*, Prentice Hall, 2001.

It is important to build a system architecture team with staff who are competent to look after the various viewpoints. The team might consist of business analysts and users who take primary responsibility for the enterprise viewpoint, software architects who attend to the computation viewpoint, and engineers who concern themselves with the engineering viewpoint, as well as experts on domain-specific viewpoints.

In addition to viewpoints, a system architecture exercise requires levels of specification. As the architecture is developed, it evolves from a general, low-detail specification to a more specific, detailed specification. Following the Rational Unified Process, there are three architectural levels, which are described in Table 2.

Model Level	Expresses
Business	<ul style="list-style-type: none"> • The work of the business that the system supports • The partitioning of the enterprise into the system and its context
Analysis	Initial partitioning of the system to establish the conceptual approach
Design	Realization of the analysis model to hardware, software, and people
Implementation	Realization of the design model into specific configurations

Table 2: Architectural Levels

Through these levels, the design goes from the abstract to the physical. The business provides the context of the system and the efforts it supports. At the analysis level, the partitions do not reflect choices of hardware, software, and people. Instead, they reflect design approaches for dividing up what the system needs to do and how the effort should be distributed. At the design level, the decisions are made as to the sorts of hardware and software components and worker roles that are needed. At the implementation level, specific choices of hardware and software technology are made to implement the design. For example, at the design level, a data server may be specified. At the implementation level, the decision is made to use a specific platform running a specific database application.

System Architecture Diagrams

The system architecture then is captured in a set of diagrams that express the architecture from various viewpoints and levels. As shown in Table 3, there is not a diagram for every viewpoint-level combination. At the implementation level, a single diagram captures the realization of hardware and software components for each system configuration.

Models	Viewpoints				
	Enterprise	Computation	Information	Engineering	Process
Business	<ul style="list-style-type: none"> UML organization model System context diagram	<ul style="list-style-type: none"> Enterprise object model 	<ul style="list-style-type: none"> Enterprise data model 		
Analysis		<ul style="list-style-type: none"> Subsystem diagram 	<ul style="list-style-type: none"> System data model 	<ul style="list-style-type: none"> Locality diagram 	<ul style="list-style-type: none"> Process diagram
Design	<ul style="list-style-type: none"> Business Worker Survey 	<ul style="list-style-type: none"> Subsystem class model Software component model 	<ul style="list-style-type: none"> System data schema 	<ul style="list-style-type: none"> Descriptor node diagram 	<ul style="list-style-type: none"> Detailed process
Implementation	<ul style="list-style-type: none"> Worker Instructions 	<ul style="list-style-type: none"> Configurations: deployment diagram with software system components 			

Table 3: Static System Architecture Views

Almost all the artifacts specified in Table 3 are standard UML diagrams. For example, in the analysis level of the computational viewpoint, the system is decomposed in UML as subsystems that collaborate to meet user requirements. In RUP SE, subsystems are defined as in The Unified Modeling Language Reference Manual¹. These subsystems, in turn, are decomposed into either subsystems or classes. The design level of the computational view is the detailed class model. Figure 2 is a subsystem diagram for a click-and-mortar retail system.

The Business Worker Survey is a current RUP artifact. Note that the worker instructions can be derived using the flow-down technique discussed below

¹ Rumbaugh, James, Grady Booch and Ivar Jacobson, *The Unified Modeling Language Reference Manual*, Addison Wesley, 1999, page 458.

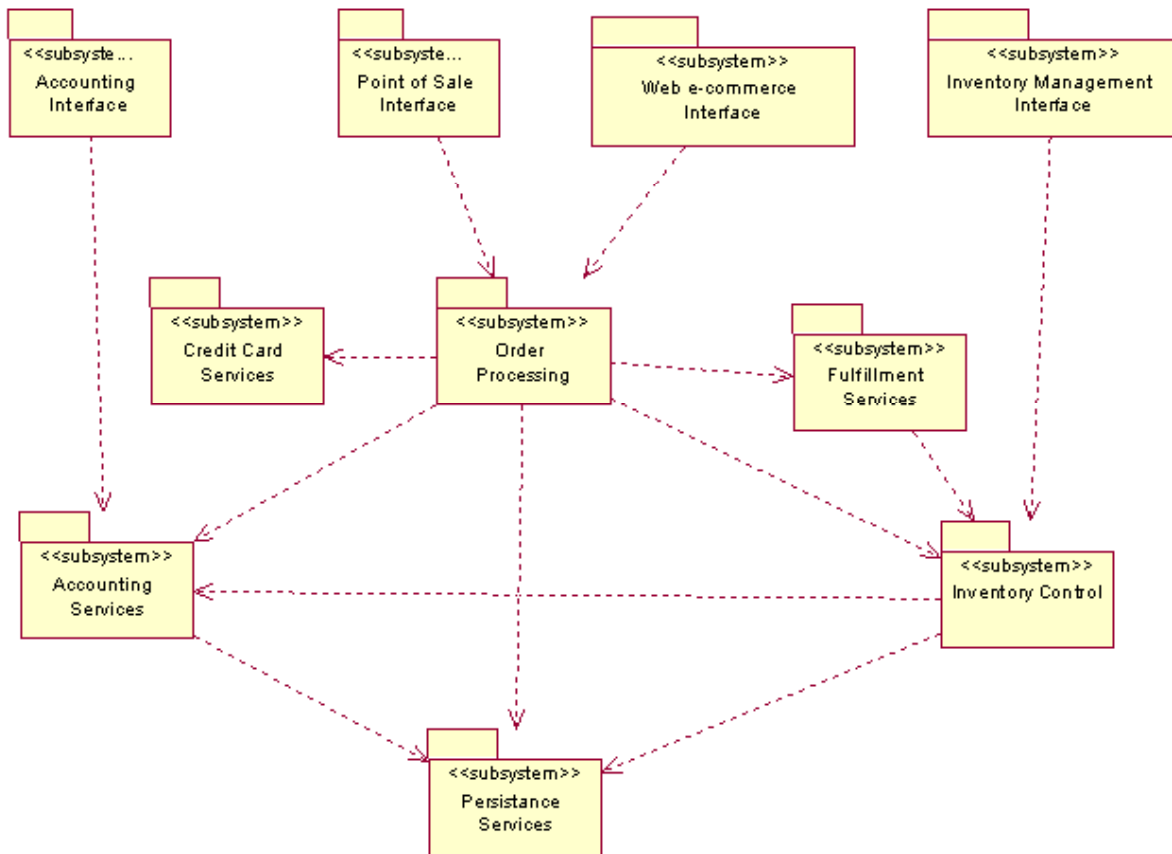


Figure 2: Example Subsystem Model

The process model is also standard UML¹. Figure 3 shows an example.

The domain-specific viewpoints should also have artifacts in place for one or more of the levels. The set of project artifacts, within this framework, should be a part of the project development case.

¹ Booch, Grady, James Rumbaugh and Ivar Jacobson, *The Unified Modeling Language User Guide*, Addison Wesley, 1999, page 455.

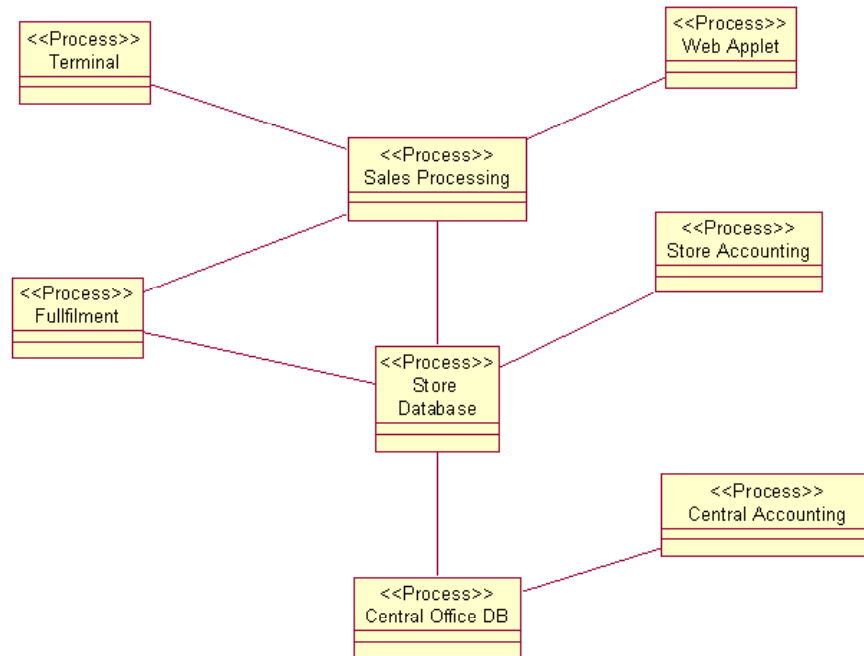


Figure 3: Sample Process Model

Locality

UML support for the engineering viewpoint (Table 1) is more problematic. UML does provide design level artifacts to capture engineering decisions in the descriptor version of the deployment diagram. The deployment diagrams are meant to capture configurations, actual choices of hardware and software, and to provide a basis for system analysis and design, serving as an implementation level in the technology viewpoint. The UML Reference Manual describes a deployment diagram as “a diagram that shows the configuration of run-time processing nodes and component instances and objects that live in them.”¹

As shown in Table 3, RUP SE uses an analysis level, engineering viewpoint diagram called *Locality*. In the engineering viewpoint, the system is decomposed into elements by which host the processing. Locality diagrams are the most abstract expression of this decomposition. They express notionally where processing occurs without tying the processing locality to a specific geographic location or even the realization of the processing capability as kinds of hardware. That level of detail is captured in the design model. For example, a locality view might show that the system enables processing on a space satellite and a ground station. The processing hosted at each locality is an important design consideration. Figures 4 and 5 provide other examples.

The locality diagrams show the initial partitioning, how the system processing elements are distributed, and how they are connected. Locality of computing is an issue when considering primarily non-functional requirements. For many systems engineers, this is “the architecture.” Sometimes the elements of this view are nodes.

¹ Rumbaugh *et al.*, *Op. cit.*, page 252ff.

Locality diagrams consist of two elements:

- Localities – a collection of computing and storage resources that can host processing
- Connections – information paths between the localities

The semantics of the locality diagrams are similar to those of deployment diagrams. Localities are stereotyped UML nodes. Recall that UML nodes are classifiers that have processing ability and memory¹. As such, they may be stereotyped and tagged values may be applied. Localities are stereotyped nodes. Their icon is a rounded cube (see Figures 2 and 3).

Localities may be realized as a hardware platform or a group of workers communicating via fax. They have *characteristics* specified by tagged values.

Localities have two sets of tags:

- Quality: reliability, availability, performance, capacity, and so on
- Management: cost, technical risk

These locality characteristics form a nominal set. Each development team should determine the best set of characteristics for their project. This determination could be a development case specification activity.

Locality characteristics are set to meet their derived requirements. There is a subtle difference between characteristics and requirements. For example, for good engineering reasons, you might specify a locality that exceeds requirements.

A locality is notionally where processing occurs. What processing occurs at the locality is specified by the subsystem use cases hosted on the localities, determined by the flowdown process discussed below. Each locality is a candidate for providing or hosting a set of logical subsystem use cases. Localities can participate in dialogs in much the same way as logical components.

Localities are related by connections, which are the mechanisms for information passing. Connections are stereotyped associations with tagged values, again capturing characteristics. Nominal connection tags are:

- **Throughput:** data rate, supported protocols
- **Management:** cost, technical risk

In the design model, localities may be realized as one or more processor nodes, or more than one locality may be realized as a single node.

Figures 4 and 5 are locality diagrams that document different engineering approaches to a click-and-mortar enterprise. The enterprise has a number of retail stores, central warehouses, and a web presence. In the first solution (Figure 4), there is processing capability in the stores. In the second solution (Figure 5), all the terminals are connected directly to a central office processor. In each case, characteristics can be set of the localities that are required to realize the design. These days, most people would agree that Figure 4 is a better design. However, the solution in Figure 5 may be superior in a few years.

¹ Rumbaugh *et al.*, *Op. cit.*, page 358.

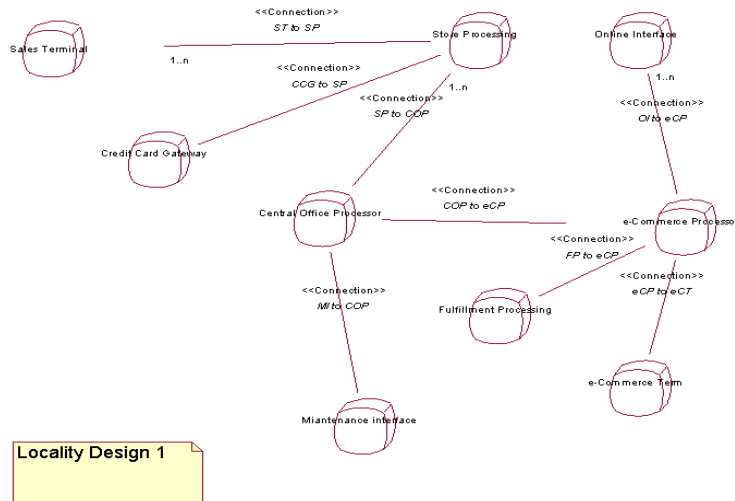


Figure 4: Locality Diagram, Example 1

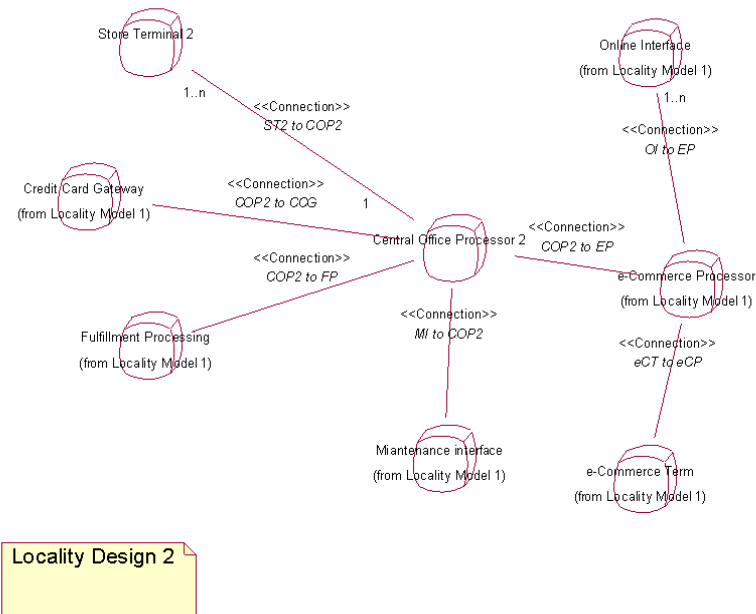


Figure 5 Locality Diagram, Example 2

Relation to 4+1 Architecture Model

The viewpoints and models, along with the use of derived requirements discussed in the following section, are consistent with the 4+1 architecture framework (Figure 6) and model levels currently documented in RUP¹. In particular, the engineering viewpoint is a generalization of the 4+1 deployment view, and the computation viewpoint is a generalization of the 4+1 view.

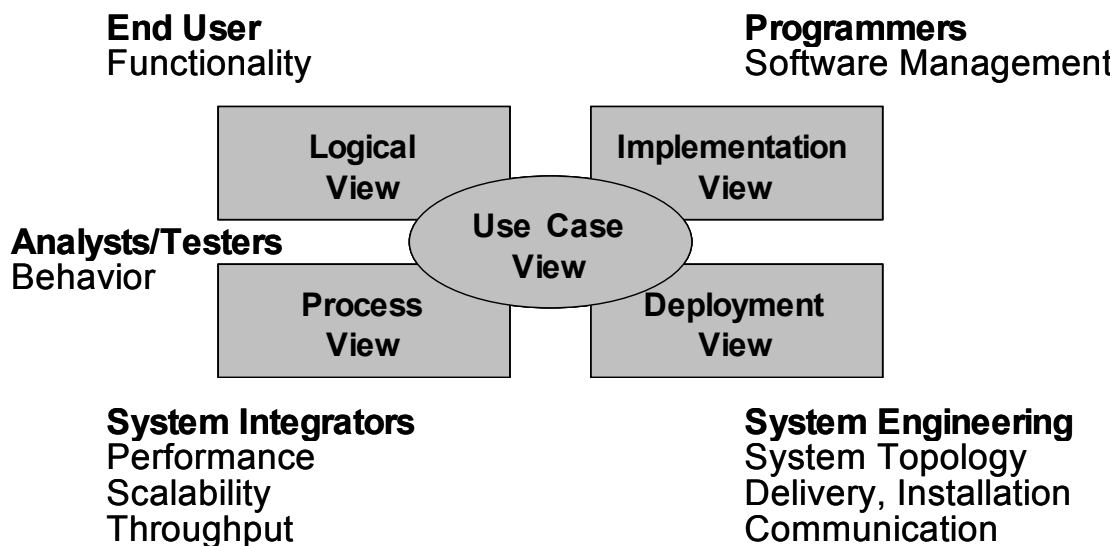


Figure 6. RUP 4+1 Architecture Framework

Requirements Analysis

Following UML and RUP, there are two types of system requirements in RUP SE:

- Use cases – services provided by the system to its actors. Use cases capture the system functional requirements and may have associated performance requirements. An *actor* is any external entity that interacts with the system. Typically, actors are users or other systems.
- Supplementary – nonfunctional requirements such as reliability and capacity

Derived Requirements

In RUP SE, a distinction is made between allocated and derived requirements. A requirement is *allocated* if a system requirement is assigned to an architectural element. A requirement is *derived* if it is determined by studying how the architectural element collaborates with others to meet a system requirement.

¹ Kruchten, *Op. cit.*

The use of derived requirements for subsystems collaborating to carry out use cases is called *logical decomposition*. Similarly, determining subsystem by allocation is *functional decomposition*. Generally, logical decomposition is essential for quality systems.

One aspect of the systems problem is to specify a set of system use cases and supplementary requirements that, if met, would provide for a system that meets its business purpose. It follows that the system requirements are derived from an understanding of the business model. The system architectural elements in the analysis model are subsystems, localities, and processes, as described earlier. In the requirements analysis discipline, requirements for each of these types of elements are determined.

There is a process pattern for deriving requirements for architectural elements:

- Determine the requirements for a given model.
- Decompose that model into elements, assigning roles and responsibilities to the elements
- Study how the elements collaborate to carry out the model requirements. This usually involves some form of collaboration diagram.
- Synthesize the analysis of the collaboration to determine the requirements for the elements.

This pattern is well known^{1 2}. It is particularly interesting that Friedenthal et al. in their Object Oriented System Engineering Method (OOSEM) also adopted the pattern³.

For example, with the business model in place, the RUP SE method for deriving system requirements is by partitioning the enterprise into the system and its actors. Then how the system and its actors collaborate to meet the business requirements is studied to determine the system requirements.

The following sections describe the application of this pattern for deriving requirements to the elements of the analysis model. The same method, with little modification, can be applied to determine system requirements from business requirements.

Use-Case Flowdown

Use-case flow down is the activity for deriving functional requirements for the analysis elements. The outcomes of the activity are:

- Use-case survey for subsystems
- Survey of hosted subsystem use cases for localities
- Survey of realized subsystem use cases for processes

The activity begins with the standard RUP activity of choosing an architecturally significant set of use cases. For each chosen use case, the flow of events is developed. This is the description of the interactions between the system actors and the system. The system responses are *black box*; the descriptions make no reference to the architectural elements. Table 4 shows an example flow of events for making a sale in a retail store. Black box steps have associated performance requirements.

¹ Cockburn, *Op. cit.*

² Putman, *Op. cit.*

³ Friedenthal, Sanford, *et al.*, "Adapting UML for an Object-Oriented Systems Engineering Method," Proceedings of the 2000 INCOSE Symposium.

Step	Actor Action	Black Box	Black Box Budgeted Requirements
1	This use case begins when the Clerk pushes the New Sale button.	The system brings up new sale clerk and customers screens and enables the scanner.	Total response time is 0.5 second.
2	The Clerk scans the items and enters the quantity on the keyboard.	For each scanned item, the system displays the name and price.	Total response time is 0.5 second.
3	The Clerk pushes the Total button.	The system computes and displays on the screen the total of the item prices and the sales taxes.	Total response time is 0.5 second.
4	The Clerk swipes the credit card.	This use case ends when the system validates the credit card, and, if it is valid, Prints out a receipt, Updates the inventory, Sends the transaction to accounting, And clears the terminal.	Total response time is 0.5 second.
		If the credit card is not valid, the system Returns a rejected message.	Total response time is 0.5 second.

Table 4 Example Black Box Flow of Events

The next steps are also standard RUP: Apply OOAD to determine the subsystem and process models. Table 4 is a flow for a use case for a click-and-mortar retail system. In this example, following RUP, the subsystem and process diagrams for the system are shown in Figures 2 and 3.

The next steps are a departure from the current RUP activity. With initial subsystem, locality, and process diagrams in place, the team revisits the flow of events by specifying how the analysis elements participate in carrying out the use case. Because this version of the flow of events refers to design elements, it is the white box view. Table 5 shows an example white box flow for the example system using locality model 1 (Figure 5).

- Subsystem white box steps – how the subsystems collaborate to carry out each black box step
- White box budgeted requirements – budgeting of the black box performance requirements to the white box steps
- Locality – which locality hosts each white box step
- Process – which process executes the white box step

The following information is added to each black step, as shown in Table 5:

Note if a white box step requires more than one hosting locality or executing process, the step should be broken into smaller steps so that each step can be associated uniquely with a locality and a process.

Step	Actor Action	Black Box	Black Box Budgeted Requirements	Subsystem White Box	White Box Budgeted Requirements	Locality	Process
1	This use case begins when the Clerk pushes the New Sale button	The system brings up the a new sale Clerk screen and Customer screen, and enables the scanner.	Total response time is 0.5 second.	The Point-of-Sale Interface clears the transaction, brings up new sales screens, and requests that Order Processing start a sales list.	1/6 second	Point-of-Sale Terminal	Terminal
				Order Processing starts a sales list.	1/6 second	Store Processor	Sales Processing
				Point-of-Sale Interface enables the scanner.	1/6 second	Point-of-Sale Terminal	Terminal
2	The Clerk scans the items and enters the quantity on the keyboard.	For each scanned item, the system displays the name and price.	Total response time is 0.5 second.	The 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.	1/8 second	Point-of-Sale Terminal	Terminal
				Order Processing retrieves the name, price, and taxable status for the scanned data.	1/8 second	Store Processor	Sales Processing

Step	Actor Action	Black Box	Black Box Budgeted Requirements	Subsystem White Box	White Box Budgeted Requirements	Locality	Process
				Order Processing adds the item to the sales list.	1/8 second	Store Processor	Sales Processing
				The Point-of-Sale Interface displays the item name, price, quantity, and item total on the clerk and customer screens.	1/8 second	Point-of-Sale Terminal	Terminal
3	The Clerk pushes the Total button.	The system computes the total price of the items and sales taxes and displays the total on the screen.	Total response time is 0.5 second.	The Point-of-Sale Interface requests that Order Processing sum the price and compute the taxes.	1/6 sec.	Point-of-Sale Terminal	Terminal
				Order Processing sums the price and computes the taxes.	1/6 sec.	Store Processor	Sales Processing
				The Point-of-Sale Interface displays the totals.	1/6 sec.	Point-of-Sale Terminal	Terminal

Step	Actor Action	Black Box	Black Box Budgeted Requirements	Subsystem White Box	White Box Budgeted Requirements	Locality	Process
4	The Clerk swipes the customer credit card	The system validates the card, prints two copies of the credit card receipt and closes out the sale	30 seconds	The Point-of-Sale Interface reads the credit card data and request that that Credit Card Services validate the sales	.5 sec	Point-of-Sale Terminal	Sales Processing
				Credit Card Services requests validation through Credit Card Gateway for the given card number and amount.	28 sec	Store Processor	Sales Processing
				If valid, the Point-of-Sale Interface prints a receipt for signature.	1 sec	Point-of-Sale Terminal	Terminal
				The Point-of-Sale Interface requests that Order Processing complete the sale.	1/6 sec	Point-of-Sale Terminal	Terminal
				Order Processing requests that Inventory Control remove the items from inventory.	1/6 sec	Store Processor	Sales Processing

Step	Actor Action	Black Box	Black Box Budgeted Requirements	Subsystem White Box	White Box Budgeted Requirements	Locality	Process
				Inventory Control removes the items from inventory.	1/6 sec	Store Processor	Store Accounting
				Order Processing requests that Accounting Services post the transaction.	1/6 sec	Store Processor	Sales Processing
				Accounting Services updates the account.	1/6 sec.	Central Office Processor	Central Accounting

Table 5: Example White Box Flow of Events

The assignment of white box steps to subsystems, localities, and processes involves a set of design decisions. Each decision adds detail to the role that each analysis element plays in the overall system design. In the process of making the assignments, the team may decide to refactor the design, shifting responsibilities from one element to another within a given diagram.

The next step is to determine the subsystem use cases. This is done by sorting the white box steps by subsystem. For each subsystem, the white box steps are sorted and aggregated by similarity. The result of this process is a survey of use cases for each subsystem. An example subsystem use case survey is shown in Table 6. It includes the hosting localities and executing process for each subsystem use case.

Subsystem Use Case	Description	Locality	Process	System Use Case Name	White Box Text
Initiate Sales List	The subsystem initiates a list of items to be included in the sales transaction.	Store Processor e-commerce server	Sales processing	Enter a sale	Order Processing starts a sales list.
				Enter online sale	The e-commerce interface requests Order Processing to instantiate an ordering list and add the item to the list.
Add Product Data	The subsystem adds an item to a sales list when requested by the actor.	Store Processor e-commerce server	Sales processing	Enter a sale	The scanner data is sent to Order Processing . Order Processing retrieves the name, price, and taxable status from Inventory and updates the list.
				Enter online sale	The E-Commerce Interface requests Order Processing to instantiate an ordering list and add the item to the list.

Compute Total	...	Store Processor e-commerce server	Sales processing	Enter a sale	Order Processing sums the price and computes the taxes.
Check Availability		e-commerce server	Sales processing	Enter online sale	Order Processing requests availability status of all items from Inventory Control.
Complete Sale		Store Processor	Sales processing	Enter a sale	When Order Processing receives a valid sale, it returns Valid status to the Point-of-Sale Interface. Order Processing sends a request to Inventory Control to remove the items from inventory. Order Processing sends the transaction to Accounting Services for posting.

Table 6 Example Subsystem Use Case Survey

Once the subsystem use-case surveys are created, the set of subsystem use cases may be sorted by locality or by process.

Sorting results:

The survey of hosted use cases for each locality expresses what computing occurs at the locality as well as the associated performance requirements. This information provides input to the specification of the physical components that will be deployed at the locality. Similarly, the survey of executed use cases for each process serves as input to the specification of software components. Specification of the components is described more fully in the next section.

For various reasons, it is important to maintain traceability between the system and subsystem use cases. This traceability, generally an m-to-n relationship, is best maintained in a requirements management tool such as RequisitePro.

The textual description in the white box flow of events can also be expressed as a set of sequence or collaboration diagrams.

These diagrams convey the traffic between analysis elements:

- For each locality, create a survey of hosted use cases
- For each process, create a survey of executed use case

Each diagram is a sequence diagram whose objects are proxy diagram elements. The messages are invocations of the subsystem use cases. Figures 7 and 8 show the subsystem and locality interaction diagrams for the flow of events in Table 5.

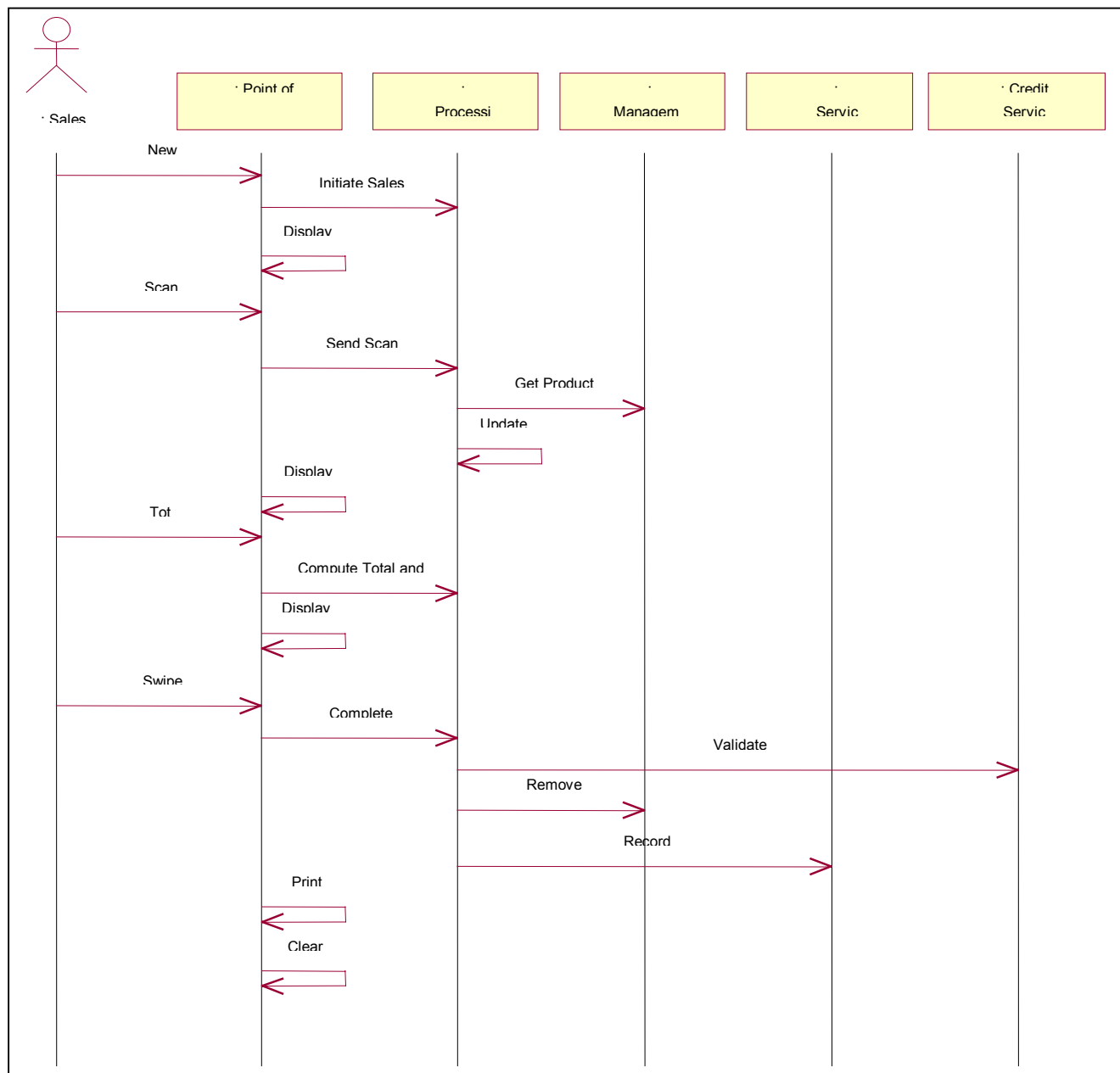


Figure7: Example Subsystem Interaction Diagram

Figure 7 provides insight into the coupling and cohesion of the subsystems. This insight may be used to refactor the subsystem design. For example, if there is a lot of traffic between a pair of subsystems, it may make sense to combine them.

Figure 8. Example Locality Interaction Diagram

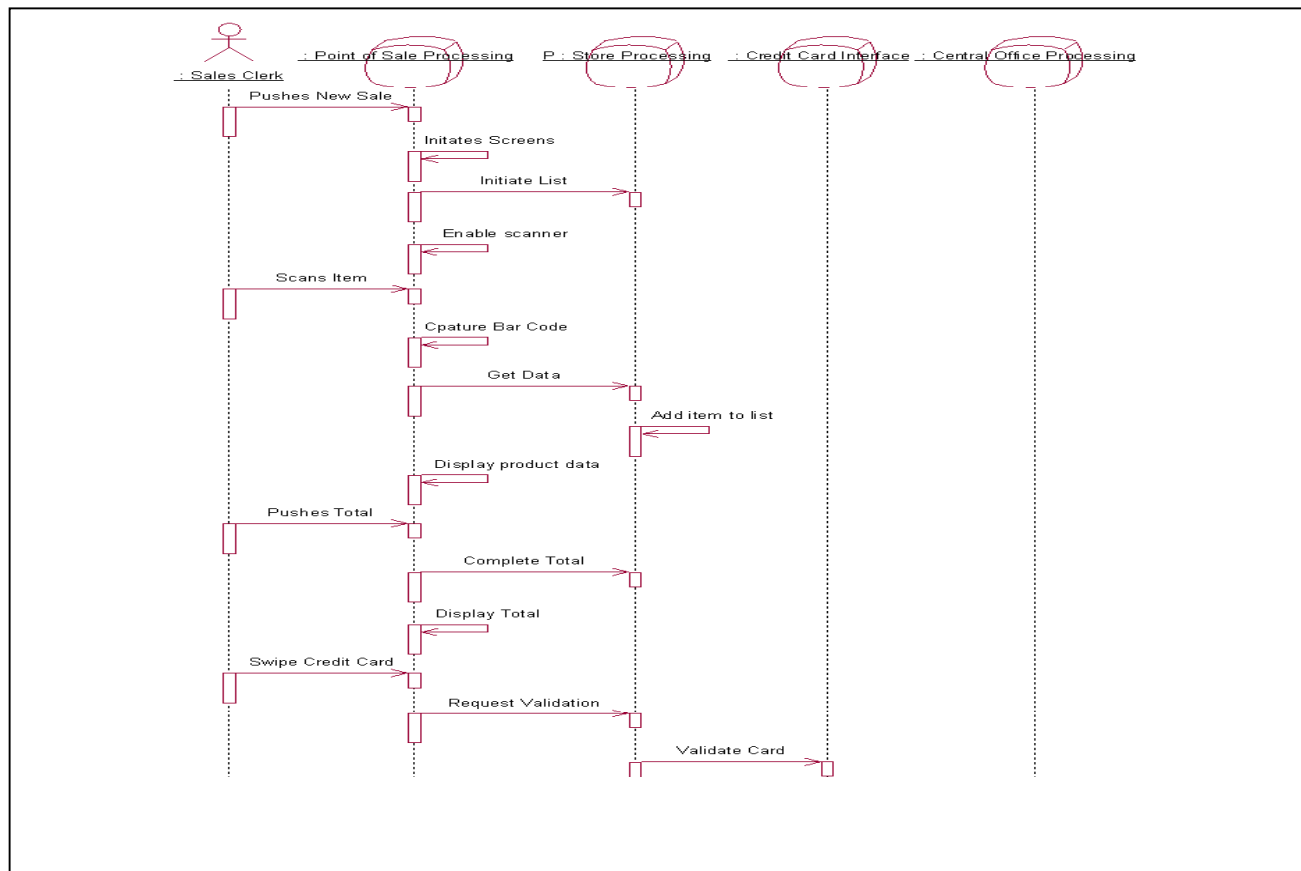


Figure 8: Example Locality Interaction Diagram

The traffic in Figure 8 shows what data must flow between the localities. This information is used to specify the associations between the localities.

Supplementary Requirements Flowdown

As a part of the analysis process, the system architects develop an initial locality diagram. The locality view is a synthesis of the non-functional considerations and provides a context for addressing how the non-functional requirements such as reliability and capacity will be addressed.

Standard engineering practice allows for the budgeting of capacity, permitted failure rates, and so forth. This effort results in a set of derived supplementary requirements for each locality element. The locality characteristics are determined from these requirements. The derived requirements and characteristics will be revisited after the hosting requirements are determined in the use-case flowdown activity described below.

Component Specification

Moving from the analysis to the design level of the architecture entails determination of the hardware and software component design. This design-level specification consists of the components, both hardware and software, to be deployed.

Hardware components are determined by analyzing the localities, their derived characteristics, and hosted subsystem use cases. With this information, descriptor-level realizations of the localities can be selected. Descriptor node diagrams specify the components, servers, workstations, workers, and so forth, without specific choices of technologies that implement those components. Figure 9 is an example descriptor node diagram that realizes the locality diagram shown in Figure 5. The fulfillment system in Figure 9 is realized as two components: a warehouse gateway and mailing/postage system, and two workers.

The descriptor nodes inherit characteristics from their localities through an allocation or budgeting process.

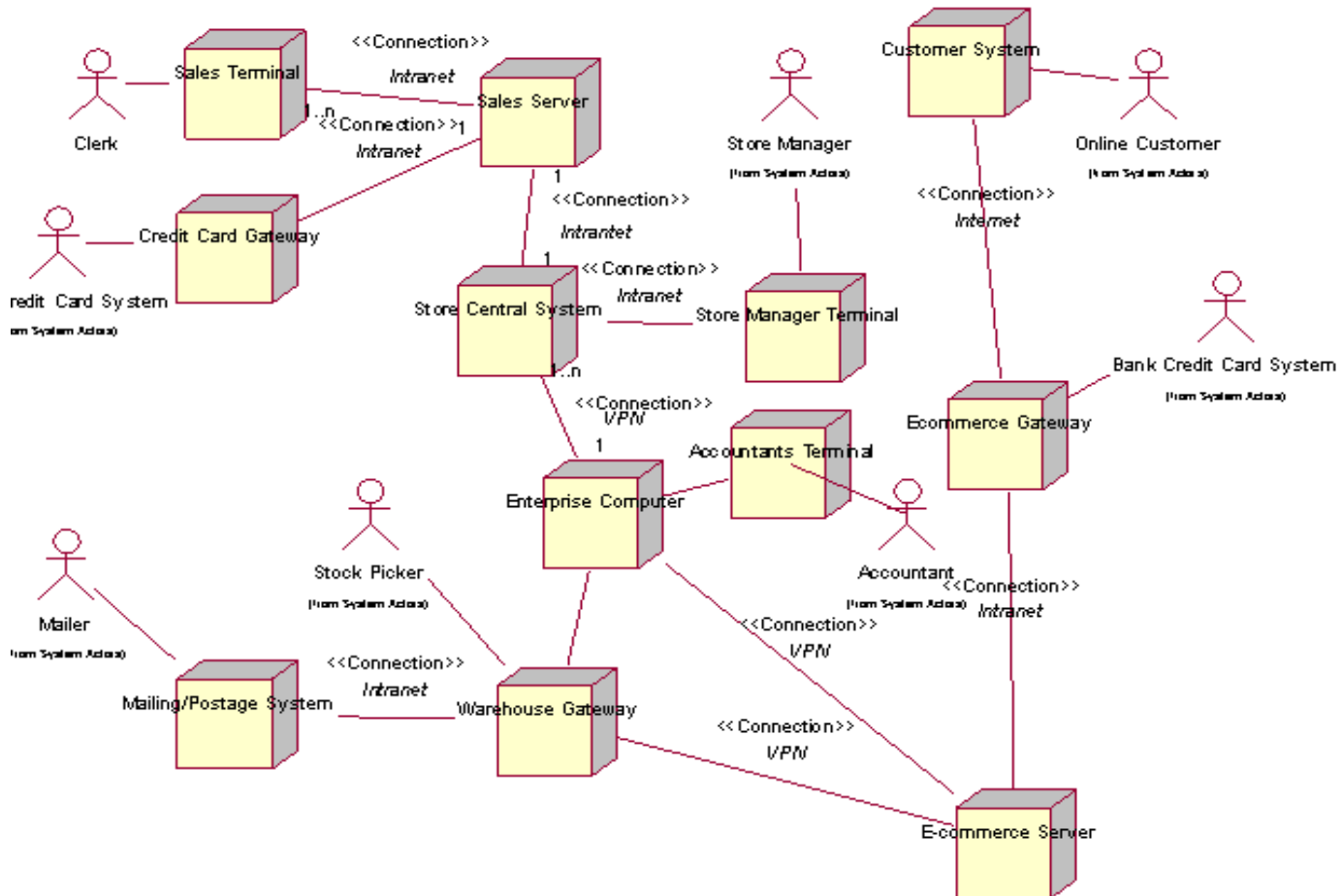


Figure 9: Example Descriptor Node Diagram

The implementation hardware components, the actual deployed set of hardware, are determined by making cost/performance/capacity trades from the descriptor view. In fact, a system may have more than one hardware configurations, each meeting different price/performance points.

Components are determined by specifying a set of object classes, and then compiling and assembling the code associated with those classes into executable files. A fully considered software component design must reflect a variety of concerns:

- Locality – where the components need to run
- Hosting – processor instruction set and memory restrictions for the executing code
- Concurrency – separation of processing into different hosts or memory spaces to address reliability and related concerns

It follows that the information needed to specify components includes the surveys of hosted subsystem use cases for localities and their realized hardware components, surveys of executed use cases for processes, along with the view of participating classes (VOPC) for the subsystem use cases.

An overview of the method is, for each hardware configuration to create a component from the class participating in all of subsystem use cases hosted on each node. If those use cases need to be executed in more than one process, divide the components further by assigning the participating classes of the subsystem use cases executed by each of the processes. Note that some subsystem use cases may be executed by more than one process and therefore their classes may be in more than one component. Complete the process by dividing the components further to account for memory constraints (such as .exe and .dll trade-offs), shipping media limitations, and so forth.

These activities result in a set of specific hardware and software components that make up the system.

System Development

RUP SE projects are managed much as any RUP project. However, because of the size and additional activities of most systems engineering efforts, there are some differences. These differences are discussed briefly in this section.

Project Organization

The movement from a serialized to an iterative process has profound implications in how a project must be organized. In a serialized process, staff is often assigned to a project until their artifacts are complete. For example, the engineering staff might complete the specifications, hand them off to the development staff, and move on to the next project. In any RUP-based project, no such handoff occurs. Rather the artifacts evolve throughout the development. It follows that the staff responsible for project artifacts, such as the requirements database and UML architecture, must be assigned to the development project throughout its duration.

Figure 10 shows the organization for a typical RUP SE project. The organization is collection of development teams, each with a project manager and a technical lead. There are also teams that deal with overall system architecture and project management.

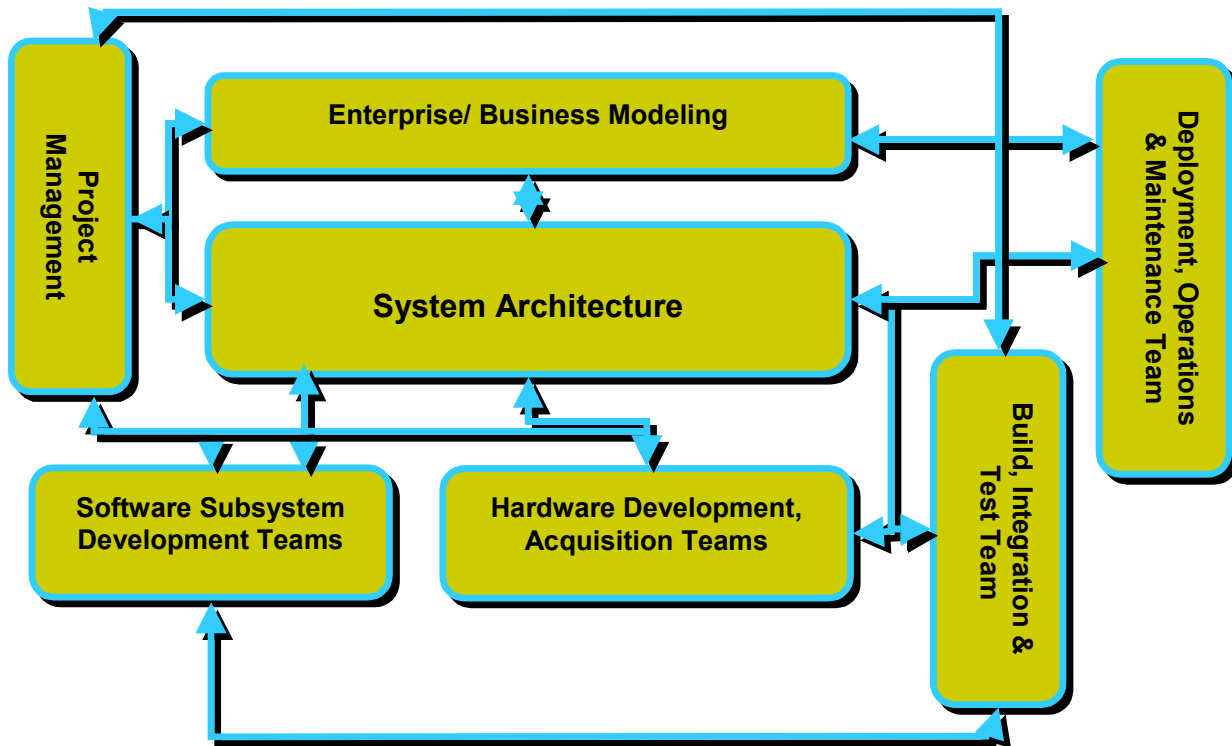


Figure 10 A RUP SE Organization Chart

- The **Enterprise Modeling** team analyzes the business need and generates business models and/or related artifacts such as Concept of Operations documents.
- The **System Architecture Team** works with the Enterprise Modeling Team to create the system context and derive system requirements. The team develops the subsystem and locality views as well as their derived requirements. Throughout the development, this team serves as a technical escalation point, resolving architectural and engineering issues. The System Architecture Team also works with the development teams to specify the software component architecture. Team members include the technical leads of the development teams.
- The **Project Management Team** looks after the standard project issues such as project reviews, resource planning, budget tracking, earned value and variances, and coordinated iteration planning
- For each iteration, the **Integration and Test Team** receives the code and hardware components from the development teams, builds the software components, and installs the hardware and software components in a laboratory setting. The team also plans, executes, and reports on the system tests for each iteration.

- The **Subsystem Development** teams are responsible for the design and implementation of one or more subsystems. The teams base their work on the derived use cases discovered during the flowdown activity. Depending on the size and complexity of the system, the subsystem use cases may be realized as class design and associated code modules or the subsystems may be further decomposed into subsystems. In the latter case, a subsystem team may be further decomposed into sub-subsystem teams and a subsystem architecture team may be created. This process enables scalability of the RUP SE approach.
- The **Hardware Development and Acquisition Team** is responsible for the design, specification, and delivery of the cases; this team might install and maintain the system in the field. In other cases, this team might handle user defect reporting and provide patches to the field.
- The **Deployment Operations and Maintenance Team** handles operational issues and serves as a liaison with the users. In some

Concurrent Design and Implementation

One feature of the RUP SE organization approach is that it scales to very large programs. This is accomplished by taking advantage of the decomposition of the system into subsystems and localities with their derived requirements. Each of these analysis model elements is suitable for concurrent design and development. As described in the previous section, UML subsystems may be assigned to separate develop teams, localities to hardware development or acquisition teams. Each team works off of its derived use case survey to develop their portion of the design model and implementation models. This way the design and implementation of the design elements can proceed in parallel.

For very large systems, a systems-of-systems approach can be adopted. In this case, each UML subsystem has its own locality model. This assignment permits there the application of the above organization structure at the subsystem level, providing even more scalability.

Iterative Development, Integration, and Test

One central feature of the RUP is that the system is developed in a series of iterations, each of which adds functionality. The system is integrated and tested at each iteration. The iteration testing is a subset of the system tests. Consequently, the final iteration results in a fully tested system ready for transition to the operational setting.

The timing and content iterations are captured in an Iteration Plan early in the project. However, like any RUP artifact, the Iteration Plan is updated continually to reflect the emerging understanding of the system as it comes together.

The content of an iteration, captured in a *system iteration plan*, is specified by what use cases and supplementary requirements are realized by the components developed in the iteration. Each iteration is tested by the subset of applicable system test cases.

Recall that subsystems and localities have derived use cases that trace from system use cases. This tracing provides a basis for derived iteration plans for the subsystems and localities. That is, the content of each system iteration determines by traceability the functionality that needs to be provided by the subsystems and localities to support the iteration. In practice, the development teams will negotiate the iteration content to reflect their development practicalities. For example, an early system iteration cannot require full functionality of a subsystem. Compromises must be made.

A good system iteration plan provides the opportunity to identify and resolve system technical risks early, before the typical panic of the waterfall-based integration and testing phase. The technical risks can involve both functional and nonfunctional requirements. For example, an early integration can shake out system bring up and fail-over issues that cannot be fully understood with detailed design and interface specifications. In practice, the early iterations should validate that the architecture is sufficient to meet the non-functional requirements.

Iterative system development may seem more expensive because it requires more testing, as well as scaffolded or simulated hardware environments to support the early iterations. Coordination of the iteration content across development teams also

takes more project management effort. However, these apparent costs are offset by the savings in early identification and mitigation of risks associated with the system architecture. It is a standard engineering principle that removing architectural defects late in a project is much more expensive than removing them early. Removing defects late also adds uncertainty and schedule risk late in a project.

The role of the testing organization is different than it is in an organization that adopts a serialized, waterfall approach. Rather than spending more of the development planning for an overall system integration at the end of the development, the organization spends its time integrating, testing, and reporting defects.

Conclusion

RUP SE is a derivative of the Rational Unified Process; RUP SE Deployment Service is a packaged service available from Rational Software. It is suitable for projects that have one or more of the following characteristics:

- Architecturally significant deployment issues
- Concurrent hardware and/or software development efforts

RUP SE provides the system development team with the advantages of RUP best practices while providing a setting for addressing overall system issues. Some of the benefits of RUP SE include:

- **System Team Support** – Provides for ongoing collaboration of business analysts, architects, system engineers, software developers, hardware developers, and testers.
- **System Quality** – Provides the views to support addressing system quality issues in an architecture driven process
- **System Visual Modeling** – Provides UML support for systems architecture
- **Scalability** – Scales from small to large systems
- **Component Development** – Provides the workflows for determining the hardware and software components
- **System Iterative Design and Development** – Supports concurrent design, iterative development of hardware and software components

Rational®

the software development company

Corporate Headquarters
18880 Homestead Road
Cupertino, CA 95014
Toll-free: 800-728-1212
Tel: 408-863-9900
Fax: 408-863-4120
E-mail: info@rational.com
Web: www.rational.com

For International Offices: www.rational.com/corpinfo/worldwide/location.jtmpl

Rational, the Rational logo, RUP and Rational Unified Process are registered trademarks of Rational Software Corporation in the United States and in other countries. All other names used for identification purposes only and are trademarks or registered trademarks of their respective companies. ALL RIGHTS RESERVED. Made in the U.S.A.

© Copyright 2001 Rational Software Corporation.

TP165 8/01. Subject to change without notice.