

## ► **Modeling for enterprise initiatives with the IBM Rational Unified Process**

### **Part I: RUP and the System of Interconnected Systems Pattern**

by [Peter Eeles](#)

Rational Software, UK  
IBM Software Group

and

[Maria Ericsson](#)

Rational Software, Sweden  
IBM Software Group

*Developing a system is rarely about developing a single software application. Addressing business problems typically requires a broader perspective that views the system as consisting of other systems.*

*Developing such a system may involve many interdependent projects. This article looks at how the Rational Unified Process,<sup>®</sup> or RUP,<sup>®</sup> can be applied in this broader context. Out of the box, RUP focuses on the execution of a single project to develop a single software application, but it is highly suitable for developing more complex systems as well. In particular, we consider RUP as a process framework for typical enterprise initiatives such as:*



Photo © 2003 Andrew Lampitt

- **Enterprise architecting:**  
*defining an architecture<sup>1</sup> that underpins a number of systems.*

- [subscribe](#)
- [contact us](#)
- [submit an article](#)
- [rational.com](#)
- [issue contents](#)
- [archives](#)
- [mission statement](#)
- [editorial staff](#)

- **Enterprise Application Integration (EAI):** developing a solution that includes the integration of a number of legacy systems.
- **Packaged application integration:** developing a solution that includes the configuration of a packaged application, such as an Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM) solution.
- **Strategic reuse:** developing reusable assets that are used within a number of systems.
- **Systems engineering:** developing a system that contains elements of hardware, software, workers, and data.
- **Outsourced development:** defining an architecture that lends itself to the outsourced development of its constituent parts, while ensuring the quality and integrity of these parts.

*Organizations often combine a number of these initiatives because of their business situation or technological factors, and each of these initiatives may represent a "system of systems." In other words, the inherent complexity of the overall system requires development organizations to decompose it into a number of "subsystems" that they implement within a number of projects. Maintaining a consistent relationship between the overall system and its associated subsystems requires careful consideration of a number of areas including:*

- *Architecting*
- *Project management*
- *Requirements management*
- *Change management*
- *Testing*

*This article focuses on what goes into architecting a system of systems using RUP and the System of Interconnected Systems Pattern (discussed below). We wrote it in response to a growing need we see among our customers to understand how all their development efforts tie together. Although we do not provide all the answers, this article is a first step toward understanding how to scale up the process framework provided in RUP for complex enterprise systems.*

*Note that we do not provide an introduction to RUP in this article; see the **References** section for suggested reading.*

## **Key Terms**

A key term we use in this article is *system*, and so before proceeding we should be more precise about its meaning, which we base on the following definitions:

A system is a top-level subsystem in a model. A subsystem is a

grouping of model elements that represents a behavioral unit in a physical system. A subsystem offers interfaces and has operations. In addition, the model elements of a subsystem can be partitioned into specification and realization elements. [OMG Unified Modeling Language (UML), Version 1.4]

A system is a collection of connected units that are organized to accomplish a specific purpose. A system can be described by one or more models, possibly from different viewpoints. [RUP Version 2002.05 ]

A system provides a set of services that are used by an enterprise to carry out a business purpose. System components typically consist of hardware, software, data, and workers.<sup>2</sup> [RUP-SE]

Although these definitions provide different levels of detail, they are remarkably consistent. In particular, we derived from them the following assumptions about the nature of a system:

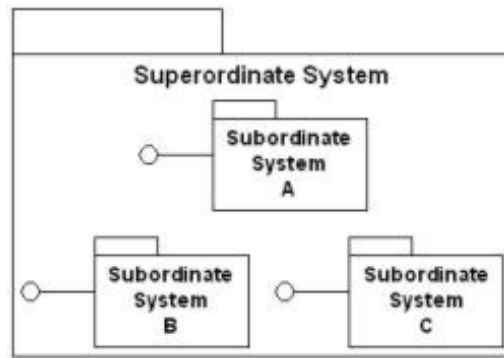
- A system exhibits behavior.
- A system can contain other elements.
- A system fulfills a specific purpose.
- A system can be composed of hardware, software, data, and workers.

Another key term in this article is *pattern*, which we use in a general sense to mean a "*common solution to a common problem in a known context.*" Christopher Alexander offers a more detailed definition that fits well with the System of Interconnected Systems Pattern we discuss specifically in this article:

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.<sup>3</sup>

## **The System of Interconnected Systems Pattern**

The System of Interconnected Systems Pattern is an architectural pattern that is used to help control the complexity inherent in a system of systems.<sup>4</sup> The pattern identifies a system that represents overall capability and refers to it as the *superordinate system*. The other systems that represent a part of this overall capability are each referred to as a *subordinate system*. This division is shown in Figure 1.



**Figure 1: The System of Interconnected Systems Pattern**

The System of Interconnected Systems Pattern is also **recursive**: *A subordinate system may have subsystems of its own and be superordinate in relation to those subsystems.* This characteristic is particularly important to initiatives such as enterprise architecting and systems engineering, as we will discuss later.

## The System of Interconnected Systems Pattern and RUP

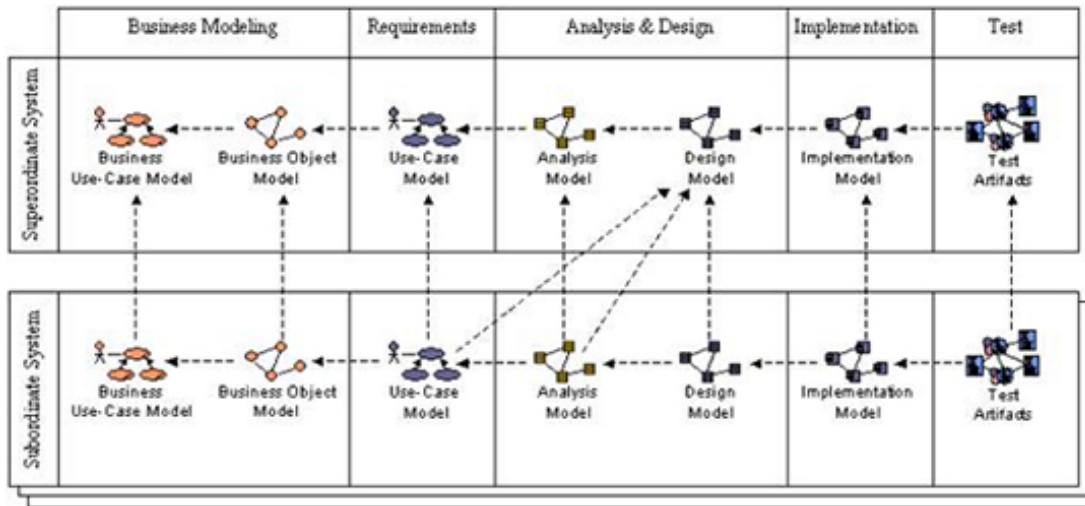
Before discussing the use of the system of interconnected systems pattern within the context of RUP, we should first be clear about the distinction between systems and projects. *The System of Interconnected Systems Pattern is primarily concerned with the architectural decomposition of a large and complex system into a number of subsystems, whereas RUP is primarily concerned with the execution of a project.* These two facets--architecture and projects--should not be confused.

Although it is often the case that a particular system (superordinate or subordinate) is best implemented as a single project, this article acknowledges that this is a simplification. A single project may implement a number of systems, and a single system may be implemented as a number of projects. On a related note, it is sometimes beneficial to think of a superordinate system as a "living entity" that, to a large extent, is never "finished." As a result, the superordinate system typically undergoes a series of sequential development cycles (i.e., sequential "passes" through the RUP phases of Inception, Elaboration, Construction, and Transition), each of which is executed within a separate project.

How can RUP support development for both levels described in the System of Interconnected Systems Pattern -- the superordinate and the subordinate? We have found that the best way to explain this is in terms of RUP artifacts, which we examine below. Later, we will touch upon related concerns, including a discussion in the **Appendix** on architectural views.<sup>5</sup>

Figure 2 shows the relationship between a superordinate system and a number of subordinate systems in terms of key RUP artifacts. It also provides a framework for thinking about specific artifacts of a superordinate project, specific artifacts of a subordinate project, and the relationships among them (we will consider additional relationships in Part

II of this series). The artifacts are aligned with the RUP disciplines in which they are produced. A description of each of these artifacts can be found in RUP itself as well as in Philippe Kruchten's book, *The Rational Unified Process: An Introduction* (see References). Although RUP is typically applied to software development projects, its concepts and best practices (such as requirements management) also apply to nonsoftware projects.



**Figure 2: Relationships among RUP artifacts for a superordinate system and subordinate system**  
([click to enlarge](#))

Figure 2 shows that, in general, a particular subordinate artifact is constrained along two dimensions:

- By its relationship with artifacts associated with the superordinate system. For example, a subordinate design model is constrained by the superordinate design model.
- By its relationship with artifacts associated with the same subordinate system. For example, a subordinate design model is constrained by the subordinate analysis model that it refines. We will describe each of the relationships shown in Figure 2 in Part II of this series.

It is also worth noting that the superordinate system is concerned primarily with a "broad brush" perspective, concentrating only on elements that are architecturally significant. However, it is the obligation of each subordinate system to provide required details -- each subordinate system effectively "populates" aspects of the superordinate system. Therefore, we can say that the *development of a system of systems is both top down* (the superordinate system provides a context for each subordinate system) *and also bottom up* (each subordinate system populates aspects of the superordinate system). RUP for Systems Engineering<sup>6</sup> provides a prescription for how to proceed -- in particular, with top-down modeling of use cases (use-case flowdown). The discussion in this article focuses on dependencies from subordinate artifacts to superordinate artifacts. However, in more general terms, the superordinate system is also dependent on each of the subordinate systems that implement it.

So what comes first -- the superordinate system or the subordinate system? It is actually a classic "chicken and egg" problem -- some might call it a "design paradox." *The whole cannot be defined without understanding the technicalities of the parts, and the parts cannot be defined in detail without understanding the whole.* This tells us that they are interdependent, and their development should go hand in hand. There is often a tendency to solve complex problems top down, just because it is easier to break a large problem down into a number of smaller and more manageable problems than to attack the whole beast in one go. However, *the risk is that we will forget to consider how the solutions for each of the smaller problems impact the overall solution* and get bogged down in building parts that in the end don't fit together. Another common problem is that when we have many "parts" (e.g., when integrating legacy systems) that we'd like to fit together, we must limit the whole, based on what the given parts are able to provide.

## **Applying the System of Interconnected Systems Pattern to enterprise initiatives**

Now that we understand how RUP relates to the System of Interconnected Systems Pattern, we can examine how to apply the Pattern to the initiatives we discussed earlier: enterprise architecting, EAI, packaged application development, strategic reuse, systems engineering, and outsourced development. Note that in each instance, the project teams should decide which of the RUP artifacts we described in the previous sections would be beneficial to the project (we will discuss this later), and then create those artifacts according to the relationships depicted in Figure 2.

### **Enterprise architecting**

Enterprise architecting is concerned with providing a "platform" for developing all systems that comprise an enterprise, and typically has concerns within a number of areas, such as data, functionality, geographic distribution, and people.

We can decompose an enterprise into its respective elements by expressing the enterprise itself as a superordinate system, as we have described it, and the elements of the enterprise (in this context) as subordinate systems. It is especially important to note that the System of Interconnected Systems Pattern is recursive, since enterprise architecting initiatives are often described at a number of levels.

### **Enterprise application integration**

EAI is concerned with developing a solution that includes the integration of a number of legacy systems. Such efforts are, to a large extent, driven from the "bottom up," since elements of the solution already exist. However, there is still some "top down" effort required to understand the context within which such legacy systems will "fit." Also, if the legacy system represents software, often techniques such as "wrapping"

interfaces to the legacy software applications are required, together with a good understanding of the available middleware technologies that can be used to interact with such applications.

We can describe the context within which a legacy system fits as a superordinate system, and represent the legacy system itself as a subordinate system.

## **Packaged application development**

Packaged applications are, in effect, customizable frameworks that allow you to build a "family" of applications that support a certain aspect of a business, such as CRM or HR (Human Resources). These frameworks can be considered at two levels.

First, such frameworks often implement a piece of a larger system. In this context, the packaged application (or a piece of the packaged application) represents a subordinate system. Second, such frameworks are often large and complex. Thinking of them as a system of systems in their own right can help us understand them, especially when it comes to understanding how the packaged application will be applied. What pieces will be used as-is? What pieces will be used after modification or configuration? What pieces will not be used at all?

## **Strategic reuse**

There are many dimensions to a strategic reuse initiative, including business concerns (such as return on investment decisions) as well as technical concerns. In this article, we are concerned primarily with the technical aspects of "architecting for reuse." Also, although reusable assets can take many forms (including documents and models), this article focuses primarily on elements of the final system, such as software and reusable hardware.

Reusable assets (whether a simple component or an entire system) do not, by definition, exist in isolation, because they are reused within a number of contexts. A fundamental premise of any strategic reuse initiative, therefore, is to define the services that each asset provides and any services it requires. These assets and their relationships can be described in terms of a system of systems. For example, in his book *Software Reuse*, Ivar Jacobson considers the concepts of an *application system* and a *component system*.<sup>7</sup> In simple terms, an application system represents an application that provides value to an end user, whereas a component system represents a set of components that are used by a number of application systems. The system of systems that shows the relationships between the application systems and component systems is described in terms of an overall layered architecture -- the product of *application family engineering*, which other texts refer to as *product line engineering*.

## **Systems engineering**

As we said earlier, systems engineering is concerned with hardware,

software, workers (people), and data. Two aspects of the process of systems engineering are identifying the elements that comprise the system and understanding the relationships among them. In identifying the elements, we particularly take into account nonfunctional requirements such as performance and cost. For example, we may choose to implement a system element in hardware rather than software for performance reasons, or in terms of workers (people) rather than software for usability reasons (an end user may have a better "user experience" of the system when interacting with a human being!).

The system as a whole can be expressed in terms of a superordinate system, which identifies the elements that comprise the system and the relationships among them. Each of these elements can then be expressed in terms of a subordinate system. Again, it is important to bear in mind that the System of Interconnected Systems Pattern is recursive, since most systems engineering efforts require more than two levels of decomposition. Note that various system boundaries (as well as subsystem boundaries, and potential subsystem boundaries) in system engineering actually enclose people, hardware (computational and noncomputational), and software.

See RUP for Systems Engineering<sup>8</sup> for more detail.

## **Outsourced development**

When an organization outsources development in order to supplement development capacity or reduce costs, it is vital to maintain architectural integrity between the overall system and its constituent parts. Constraints imposed on the development of the "parts" may vary. For example, you might describe an outsourced part in terms of the requirements it must fulfill, assuming complete flexibility in terms of the solution. Or you might describe an outsourced part in terms of a detailed set of services that it must provide, or even how it will be implemented.

Whatever the scope of the outsourcing, it is advantageous to describe the overall architecture in terms of a superordinate system and the constituent parts in terms of subordinate systems.

## **When to apply the System of Interconnected Systems Pattern**

This section discusses the circumstances -- either business-driven or technology-driven -- under which to apply the System of Interconnected Systems Pattern.

### **Merging organizations**

Often, organizations merge to (among other things) save costs and then struggle when they discover the complexity of this task. There are many difficult decisions to make: what future platform to choose, which systems to keep and which to replace, and how to tackle the business impact of changing systems, to name a few. A merger is a system of systems

problem involving people, hardware, and software, and may entail the following kinds of initiatives:

- **Enterprise architecting:** to get an overall understanding of the problem.
- **Enterprise application integration:** to integrate existing and new systems.
- **Strategic reuse and outsourced development:** to develop with efficiency.

## Modernizing legacy systems

Organizations often find it necessary to move to new technologies to modernize their systems. Over time, it becomes more and more costly (and time consuming) to add capability to legacy systems, so re-architecting and replacing these systems makes business sense. A typical approach is to gradually replace one or more legacy systems and integrate new ones on a schedule that ensures appropriate management of business or technical risks. Understanding the overall business impact of evolving legacy systems is a system of systems problem involving people, software, and hardware. This type of effort may involve the following initiatives, which can benefit from applying the System of Interconnected Systems Pattern:

- **Enterprise architecting:** to understand the impact of introducing modern technology.
- **Enterprise application integration:** to integrate legacy systems with new systems and also legacy systems with other legacy systems.
- **Packaged application development:** to integrate new technology such as an ERP solution.

## Building technically complex products

Building technically complex products, such as telecom network products or air-traffic control systems, has always been considered a system of systems undertaking that involves both hardware and software. As technologies are now changing more rapidly than ever before, there is a stronger need for a thorough treatment of such efforts. It is no longer sufficient to break these systems down into smaller pieces that you resolve independently while handling dependencies along the way. Instead, organizations need a well-defined approach that helps them to proactively handle dependencies and be flexible. Architectural decisions are challenging, and they should not depend merely on peoples' experience. More systematic and efficient methods are required to assess the potential impact (on cost, resources, architecture, and technology) of these decisions.

This type of effort may involve the following initiatives, which benefit from

applying the System of Interconnected Systems Pattern:

- **Systems engineering:** to integrate hardware, software, and process, and define the project overall.
- **Outsourced development and strategic reuse:** for more efficient development.

## Hardware development going soft

Many organizations that have traditionally considered themselves builders of hardware are becoming increasingly dependent on building quality software. Consumer products such as mobile phones, TVs, and cars contain more software than ever before, and designing the hardware and software to work optimally together is a system of systems problem. This type of effort may involve the following initiatives, which are appropriate for the System of Interconnected Systems Pattern:

- **Systems engineering:** to integrate hardware, software, and process, and define the project overall.
- **Strategic reuse:** for more efficient development.

## Deciding which RUP artifacts to create

The first step in deciding what RUP artifacts to create when you use the System of Interconnected Systems Pattern is to consider whether or not the Pattern should be applied at all. In many instances you may be able to consider a system in its entirety without considering any separate, subordinate systems. *The Pattern is most effective when the benefit of managing the system's complexity outweighs the overhead of defining subordinate systems* (and producing additional artifacts). For example, if we were to build an air-traffic control system, we might face several complex demands, including the need to 1) develop hardware as well as software; 2) clearly separate the responsibilities of (and boundaries between) system elements so that they can be outsourced; and 3) establish effective communications among geographically distributed teams. The different models of the pattern provide communication mechanisms (interfaces and abstraction) needed to manage team dependencies. In this case, the benefits of treating the system as a system of systems would outweigh the cost of defining a number of separate subordinate systems.

Once you decide to apply the Pattern, then you must *identify the artifacts that should be produced for both the superordinate system and subordinate systems (since you don't necessarily need to produce all the artifacts)*. We will discuss all of the key RUP artifacts (and the relationships among them), but keep in mind that there is a cost involved in creating and maintaining each of them. Therefore, project managers need to pragmatically think through the value that a particular artifact adds, and the cost of creating and maintaining it versus the risk of *not* creating and maintaining it. Below we list some examples of choices that organizations

often make with respect to certain artifacts.

- **Business Use-Case Model and Business Object Model.** Project teams often do business modeling for a superordinate system, but not always for a subordinate system, especially not if the business model for the superordinate system provides sufficient input for developing the subordinate systems. However, business modeling for a subordinate system may be useful if the superordinate system is very large, and complexity needs to be managed. In particular, project teams may want to do business modeling for a subordinate system if they need a more detailed understanding of a particular aspect of the organization.
- **Use-Case Model.** It is common for a project team to produce requirements artifacts for both a superordinate system and a subordinate system. The trick is to find the right level of detail at the superordinate level -- enough for those defining the subordinate level requirements, without doing their work for them. In other words, it is desirable to identify and briefly outline superordinate use cases (with an emphasis on defining architecturally significant elements), but not to detail specifications of the flow of events. Such detail will be provided at the subordinate level.
- **Analysis Model and Design Model.** The analysis and design artifacts are critical for defining architecture. However, just as with requirements artifacts, the emphasis is different for the superordinate system and the subordinate systems. Development of the superordinate system focuses on defining an architecture within which the subordinate systems will be constrained, and also on identifying the subordinate systems. However, this effort may extend only as far as identifying the architecturally significant responsibilities (functional characteristics) and quality attributes (nonfunctional characteristics) of each subordinate system. Then, developing the subordinate system involves adding the detail necessary to ensure that the subordinate system meets these responsibilities and quality attributes, in terms of elements that implement the subordinate system. In some cases, teams create an analysis model only for the superordinate system to assist in architectural decision making at that level, but they do not really need that kind of decision making or model at the subordinate system level.
- **Implementation Model.** Developing a superordinate system may not require any implementation at all (aside from implementations of the subordinate systems that comprise it). However, organizations may want to undertake some implementation to prove aspects of the superordinate system's architecture, or to ensure that some common components (for example) are available to each subordinate system. The development of subordinate systems typically includes an element of implementation for system delivery (unless the subordinate system is further decomposed).
- **Test artifacts.** Organizations may undertake some testing to validate aspects of the superordinate system, and they must always test the integration of each subordinate system. Testing during

development of a subordinate system is primarily to validate that system's implementation.

Deciding what information should be specified at what level (superordinate or subordinate) can be overwhelming. There is a tendency to treat the superordinate level very lightly and focus the majority of the work at the subordinate level. When this happens, the risk is that the overall architecture of the superordinate system will never become really coherent. Because many decisions are made at the subordinate level, the overall results may not be quite as consistent as they should be. Conversely, there may be a risk that the work at the superordinate level will go into too much detail, and that the detail will need to be redone at the subordinate level.

*The challenge is to find the right balance between providing sufficient detail to ensure consistency among subordinate systems, and allowing enough flexibility at the subordinate level (i.e., not imposing artificial constraints).* There are no simple rules for how to do this; decisions need to be based on the characteristics of the system of systems being developed. In the **Appendix** to this article, we discuss additional approaches to depicting a complex system's architecture.

## **A word about iterative development**

There is one particular characteristic of RUP that we do not want to overlook: iterative development. Although this article does not emphasize this particular characteristic, we do acknowledge that taking an iterative approach to applying the pattern is critical to ensuring success. This is true for all of the system elements we have discussed -- software, hardware, data, and workers.

The decision space associated with the initiatives we have discussed in this article is huge. The likelihood of defining a successful architecture up front is extremely small, so *the only effective means to converge on a suitable architecture is to design a little, implement a little, test a little, incorporate lessons learned, design a little, implement a little, test a little, and so on.* The **References** below include a number of sources that discuss iterative development in detail, including Philippe Kruchten's book and RUP itself. Particular consideration of iterative development with respect to a system of systems can be found in RUP-SE as well as Ivar Jacobson's book.

## **Summary**

In summary, we can apply the System of Interconnected Systems Pattern, the iterative approach embodied in RUP, and appropriate RUP artifacts, to a diverse set of enterprise initiatives, ranging from enterprise architecting to systems engineering. The System of Interconnected Systems Pattern provides a means of managing complexity within such initiatives and complements the best practices underpinning RUP.

In Part II of this series, we will describe in detail how to apply various RUP disciplines to the development of both a superordinate system and a

subordinate system, including a number of artifacts we discussed in Part 1.

## Acknowledgments

The authors would like to thank the following people for their help and guidance in writing this paper: Roger Bowser, Dave Brown, Murray Cantor, Kelli Houston, Ivar Jacobson, Paula Simmonds, John Smith, and Dave West, all of IBM Rational, Christina Cooper-Bland of BACS, Jon Pidgeon of Lloyds TSB, Alan Whitfield of UK Inland Revenue, and Gary Willcocks of EDS.

## References

The following references were used in preparing this paper.

Christopher Alexander *et al.*, *A Pattern Language*. Oxford University Press, 1977.

Paul Allen. *Realizing e-Business with Components*. Addison-Wesley, 2000.

Scott Ambler, *The Unified Process Elaboration Phase*. R&D Books, 2000.

Colin Atkinson *et al.*, *Component-based Product Line Engineering with UML*. Addison-Wesley, 2001.

Len Bass, Paul Clements, and Rick Kazman, *Software Architecture in Practice*. Addison-Wesley, 2003.

Jan Bosch, *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley, 2000.

Murray Cantor, "Rational Unified Process for Systems Engineering." IBM Rational whitepaper, available at <http://www.rational.com/products/whitepapers/wprupsedeployment.jsp>

John Cheesman and John Daniels. *UML Components--A Simple Process for Specifying Component-Based Software*. Addison-Wesley, 2000.

Paul Clements *et al.* *Documenting Software Architectures--Views and Beyond*. Addison-Wesley, 2002.

Maria Ericsson. "Developing Large-scale Systems with the Rational Unified Process." Rational whitepaper available at <http://www.rational.com/products/whitepapers/sis.jsp>.

Hans-Erik Eriksson and Magnus Penker. *Business Modeling with UML--Business Patterns at Work*. John Wiley & Sons, 2000.

Michel Ezran, Maurizio Morisio, and Colin Tully. *Practical Software Reuse*. Springer, 2002.

Peter Herzum and Oliver Sims. *Business Component Factory*. John Wiley & Sons, 1999.

Christine Hofmeister, Robert Nord, and Dilip Soni. *Applied Software Architecture*. Addison-Wesley, 1999.

IEEE-Std-1471-2000. *Recommended Practice for Architectural description of Software Intensive Systems*. Available at <http://standards.ieee.org/catalog/olis/se.html>.

Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse*. Addison-Wesley, 1997.

Ivar Jacobson, Maria Ericsson, and Agneta Jacobson. *The Object Advantage--Business Process Reengineering with Object Technology*. Addison-Wesley, 1994.

Philippe Kruchten. *The Rational Unified Process--An Introduction*. Addison-Wesley, 2000.

Janis Putman. *Architecting with RM-ODP*. Prentice Hall, 2000.

Rational Unified Process, version 2002.05. IBM Rational Software.

RUP for Systems Engineering. IBM Rational Software. Available from IBM Rational Developer Network (<http://www.rational.net>; authorization required).

Clemens Szyperski. *Component Software--Beyond Object-Oriented Programming*. Addison-Wesley, 2002.

Dave West, Kurt Bittner, and Eddie Glen. "Ingredients for Building Effective Enterprise Architectures." *The Rational Edge*, November 2002. Available at [http://www.therationaledge.com/content/nov\\_02/f\\_enterpriseArchitecture\\_dw.jsp](http://www.therationaledge.com/content/nov_02/f_enterpriseArchitecture_dw.jsp).

John Zachman. "A Framework for Information Systems Architecture." *IBM Systems Journal*, Vol.26, No.3, 1987.

## **Appendix: Architectural representation**

Although this article focuses on models for describing different aspects of a system, it is common practice to also define an architectural representation of a system that omits elements not deemed to be architecturally significant. This representation is often expressed in the form of "architectural views," with each view providing a particular perspective of a subset of one or more models.

*Architects can choose among a number of standard architectural representations, depending on the nature of the system they are describing.* Examples include the following:

The **4 + 1 Views of Software Architecture**, defined by Philippe Kruchten, is the architectural representation advocated in RUP.<sup>9</sup>

The **C4ISR** (Command, Control, Computers, Communication, Intelligence, Surveillance, and Reconnaissance) Architecture Framework, defined by the U.S. Department of Defense (DoD), is the standard used in military domains.<sup>10</sup>

The **IEEE Recommended Practice for Architectural Description of Software-Intensive Systems** (ANSI/IEEE-1471-2000) standard provides a conceptual framework for architectural description and defines what is meant by a 1471-compliant architectural description.

The **Reference Model for Open Distributed Processing** (RM-ODP) is an ISO standard.<sup>11</sup>

The **Zachman Framework**, defined by John Zachman, is most often associated with enterprise architecting.<sup>12</sup>

The **RUP-SE model framework** introduced in the Systems Engineering Plug-In to the IBM Rational Unified Process is available from IBM Rational Developer Network.<sup>13</sup>

---

## Notes

<sup>1</sup> In this article, the term "architecture" has a very broad meaning that encompasses software architecture, hardware architecture, organizational structures, and so on.

<sup>2</sup>Derived from *Systems Engineering and Analysis* (Third Edition), Blanchard and Fabrycky, Prentice Hall, 1998.

<sup>3</sup> Christopher Alexander *et al.* *A Pattern Language*. Oxford University Press, 1977.

<sup>4</sup> The system of interconnected systems pattern is documented in more detail in a number of publications, including the book by Ivar Jacobson *et al.*, *Software Reuse: Architecture, Process and Organization for Business Success* (Addison-Wesley, 1997) and an IBM Rational whitepaper by Maria Ericsson (<http://www.rational.com/products/whitepapers/sis.jsp>), "Developing Large-scale Systems with the Rational Unified Process." The latter presents principles consistent with the approach described in Murray Cantor's IBM Rational whitepaper, "Rational Unified Process for Systems Engineering," (<http://www.rational.com/products/whitepapers/wprupsedeployment.jsp>) and applied in the RUP-SE Extension to RUP, available through Rational Developer Network ([www.rational.net](http://www.rational.net); authorization required). Jacobson *et al.* use the pattern primarily for defining strategic reuse initiatives. Ericsson describes, at a high level, how to use the pattern within the context of RUP. This article describes this alignment of the pattern with RUP in more detail.

<sup>5</sup>For more detailed process discussion regarding specific initiatives, See Cantor, *Op.Cit.* on systems engineering and Jacobson *et al.*, *Op.Cit.* on strategic reuse.

<sup>6</sup>Available through Rational Developer Network: [www.rational.net](http://www.rational.net); authorization required.

<sup>7</sup>Ivar Jacobson, *Op.Cit.*

**8**RUP-SE Extension to RUP, *Op.Cit.*

**9**For more information, see Kruchten, *Op. Cit.* and RUP.

**10** For background and a rationale, see Dave West, Kurt Bittner, and Eddie Glen, "Ingredients for Building Effective Enterprise Architectures." *The Rational Edge*, November 2002. Available at [http://www.therationaledge.com/content/nov\\_02/f\\_enterpriseArchitecture\\_dw.jsp](http://www.therationaledge.com/content/nov_02/f_enterpriseArchitecture_dw.jsp).

**11**See Janis Putman. *Architecting with RM-ODP*. Prentice Hall. 2000.

**12** For more information, see John Zachman, "A Framework for Information Systems Architecture." *IBM Systems Journal*, Vol. 26, No.3.

**13** Rational Developer Network is at [www.rational.net](http://www.rational.net) (authorization required). Also see Murray Cantor, *Op.Cit.*



---

***For more information on the products or services discussed in this article, please click [here](#) and follow the instructions provided. Thank you!***