a|EA JOURNAL

Article

Applying Pattern Concepts to Systems (Enterprise) Architecture

Robert Cloutier and Dinesh Verma, Ph.D.

ABSTRACT

The existence of patterns is almost universal, and their use is evident in many domains. The human mind seems to perceive patterns without conscious thought - we notice an individual's personal habits because they form patterns. Patterns are also used in a number of engineering disciplines – software engineering, requirements engineering and mechanical engineering to name a few. Some of these disciplines have used patterns for over 20 years.

Today's enterprise systems have become extremely complex. It is difficult, if not impossible for a system architect to mentally juggle all of the details of a modern complex multi-functional and distributed system. Patterns may provide the enterprise architect an approach to managing this complexity. This article reviews some of the relevant research and application related to the use of patterns, reviews how other disciplines are using patterns, and discusses research that has been done on applying patterns to the practice of architecting complex system (enterprise) architectures. Examples of architecture patterns are presented and discussed, and a methodology and rationale for documenting architecture patterns is presented.

KEYWORDS

Patterns, enterprise architecture, systems architecture, architecture patterns, C2, command and control

INTRODUCTION

One goal of the enterprise architect is to develop and implement a complex system using a methodical and repeatable approach. This includes documenting the system architecture from a variety of views which take into consideration strategic business goals, business rules, existing and legacy systems interfacing with the system of interest, and the appropriate and applicable technologies. A number of frameworks exist to provide an organizing structure for the enterprise architect. One such framework is the Zachman Framework, providing 30 different artifact types to assist the enterprise architect in capturing, planning and documenting the new architecture (Zachman, 1987). Another is the EA^3 Cube (Bernard, 2004).

Systems architects within the US Department of Defense (DoD) have a similar tool, known as the DoD Architecture Framework, or DoDAF. This framework provides for 26 artifact types, representing different views and architectural information. The DoDAF is helpful in methodically defining, planning, and documenting the architecture of a complex system. And while there are differences, there are many parallels between the approaches suggested for enterprise architects and complex system architects. For instance, while the enterprise architect is concerned with business goals and business scope, the systems architect is concerned with the business goals of the customer (the entity paying for the complex system being architected and developed) and

the scope of the system. The two frameworks provide for artifacts that model the system and its constituent parts, the logical nodes in the system, and the infrastructure. These artifacts can be produced formally, using modeling tools and techniques such as IDEF0 or UML, or they can be created using more informal schemas. The important point is that they are created and maintained. The intent of this short discussion is not to educate the reader, but to illustrate the similarities between Enterprise Architects and Systems Architects within the Systems Engineering community – there are more similarities than differences.

Because of those similarities, there are techniques that could be useful to architects from both practices. The purpose of this paper is to describe the research being performed to investigate the application of patterns to the practice of systems architecting. Architectural patterns may have pragmatic utility for practicing enterprise architects as well. .

A SHORT HISTORY OF PATTERNS

The notion of patterns is almost universal. The human mind seems to perceive patterns without conscious thought. We notice an individual's personal habits because they form patterns. Music employs repeating patterns to make it easier to learn the tune. For example, three childhood songs have the same tune – Twinkle, Twinkle Little Star, Baa, Baa Black Sheep and The Alphabet Song - all derived from the same composition by Mozart.

In seventeenth century England, they made use of pattern books which contained rules of thumb to assist the general public in transacting business. These pattern books represented the documented experience accumulated over the first half of the 1600s in buying, selling, and leasing buildings. Figure 1 represents one example from Henry Phillips's "The Purchaser's Pattern" (Baer, 2003).

For the purposes of the discussion in this paper, a working definition of a pattern is offered as follows: A pattern is a model or facsimile of an actual thing or action, which provides some degree of representation (an abstraction) to enable the recreation of that entity over and over again.


THE 184.
Purchasers Pattern,
Much Enlarged.
The First Part.
Shewing the true value of Land or Houses, by Leafe, or otherwife.
Whereunto is added many rules for the valuing of the ruined Foundation in the City of LONDON, and for the composing of Differences between Landlord and Tenant about re-building them: alſo Rules and Tables for the valuing of all Party-Walls.

Figure 1. Mid seventeenth century "patterns book".

In modern times, Christopher Alexander is credited as being the first to understand the value of patterns in the development of systems. Alexander studied architecture at Cambridge. Though formally trained in mathematics and physics, his domain of interest was the design and construction of homes, buildings, and communities. Throughout the 60's and 70's, he developed patterns for use by other architects with the objective of improving the art of urban design. He began to identify patterns for architecture, urban designing, and planning by looking at an architectural design and then abstracting that design into its basic parts which were common across other designs.

Thinking of this in the context of Zachman's Framework, Alexander was the planner. From his perspective, communities became systems to be decomposed into component parts. He described the component parts and their relationship to other component parts in terms of boundaries. Alexander's first publication on this notion was *Notes on the Synthesis of Form* (Alexander, 1964). He further developed these concepts in *A Pattern Language* (Alexander, 1977). Alexander believed one could reuse a pattern "tens of thousands of times" and not have any two designs look the same. Nor did he did see the patterns as remaining stagnant. He believed patterns could always be improved upon. "We may then gradually improve these patterns which we share, by testing them against experience…" (Alexander, 1979).

The Alexandrian form of documenting patterns contains four components: 1) Name, 2) Context, 3) Problem, and 4) Solution (Alexander, 1977). When using this form, the *Name* of the pattern should be descriptive and should represent the solution being proposed. Naming a pattern succinctly is critical for pattern reuse. If the pattern name is cryptic and without mnemonic

value it becomes meaningless to those looking for a pattern to solve their particular problem, significantly reducing the value of documenting a pattern. The *Context* addresses the setting for the problem. This might include environment, the problem domain, or any other aspect that will help understand where the pattern is being applied. The *Problem* describes the challenge or issue which the pattern will be used to address. Finally, the *Solution* is a description on the application of the pattern – how it is used to solve the problem, and how it may be modified or adapted to accomplish the task. To demonstrate the application of the pattern concept, Alexander outlined multiple patterns for a farmhouse, as follows (Alexander, 1977):

- North South Axis
- West Facing Entrance
- Bedrooms in Front
- Garden to the South
- Half-Hipped End
- Two Floors
- Hay Loft at the Back
- Pitched Roof
- Balcony Toward the Garden
- Carved Ornaments

As one reads through the patterns used to design this farmhouse, a visual picture begins to develop in the mind's eye, creating a living picture of the farmhouse and the site on which it will rest; from nothing more than a written or spoken list.

Other simple, yet powerful patterns documented by Alexander include "House for a couple" and "House for a small family" (Alexander, 1977). The context for the "House for a couple" (Figure 2) pattern was simply characterized as "In a small household shared by two, the most important problem which arises is the possibility that each may have too little opportunity for solitude or privacy".

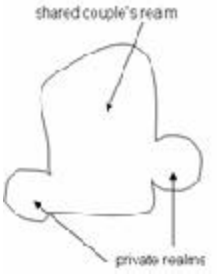


Figure 2. House for a couple.

The "House for a small family" pattern (Figure 3) context was: "In a house for a small family, it is the relationship between children and adults which is most critical". The pattern was documented to address issues regarding children and their parents: small children like to be around their parents, most parents do not have a place large enough to have a dedicated nursery, and parents do not have the heart to keep the kids out of special areas. If these issues are not taken into consideration in the design of a small house, the house soon has the character of a children's room – toys and clutter everywhere.

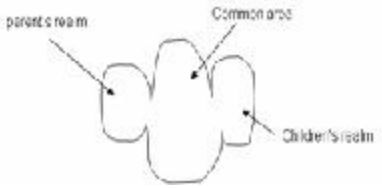


Figure 3. House for a small family.

Why did Alexander commit a great deal of his professional life identifying patterns, writing books about patterns, and extending the concept of patterns beyond civil architecture? He was attempting to lower the cognitive load of design by exploring large design spaces on behalf of the architect (Coplien, 1997)(Alexander, 1964). From his perspective,

> *"patterns helped him to express design in terms of the relationships between the parts of a house and the rules to transform those relationships"* (Coplien, 1997).

This is an important concept – in fact, if one were to take the previous sentence and replace the word "house" with "system," the same concept could apply to the notion of enterprise or system architecture patterns.

PATTERNS IN INFORMATION TECHNOLOGY

The information technology domain is beginning to embrace patterns. As an illustration, IBM is using patterns in the e-business domain. IBM has found that many customers have the same requirements, and these requirements have been abstracted into patterns. Their recipe steps

for creating an effective, run-time architecture are (Sachdeva, 2004):

Step 1: Develop a high-level business description
Step 2: Develop a Solution Overview Diagram
Step 3: Identify Business Patterns
Step 4: Identify Integration Patterns
Step 5: Identify Composite Patterns
Step 6: Identify Application Patterns
Step 7: Integrate a package into a solution
Step 8: Identify Run-time Patterns
Step 9: Identify Run-Time and Product mappings

PATTERNS IN SOFTWARE DEVELOPMENT

In the late eighties, software designers began to apply architectural concepts (patterns) to object oriented software development. The first book on the subject of software patterns was published in 1995 by Gamma, Helm, Johnson, and Vlissides (Gamma, 1995). It detailed 23 patterns, categorized as creational, structural, or behavioral patterns. All of these patterns are still in use today by programmers.

Learning to define and document a pattern is not an easy task. Martin Fowler discusses the difficulty in defining a pattern (Fowler, 1997):

> *"…we have had difficulty in defining the term pattern. We all think we can recognize a pattern when we see it, we think most of us would agree in most cases, but we cannot come up with a single definition."*

He goes on to provide his own definition of a pattern as "an idea that has been useful in one practical context and will probably be useful in others."

A number of parallels can be drawn between the use of patterns in the software engineering community and the development of ontologies by ontology engineers (Devedzic, 1999). Ontologies provide skeletal knowledge and an infrastructure for integrating knowledge bases at the knowledge level, independent of particular implementations. According to Devedzic, software patterns enable the communication of knowledge in order to solve problems effectively. Software patterns are effective in transferring knowledge by describing solutions to similar problems through common ways and techniques, regardless of the project problem domains or implementation tools and languages. And, using software patterns early in the design reduces the number of changes that have to be made later in the lifecycle. He considers some software patterns to be "micro-architectures" that contribute to the overall software system architecture. That assertion is made because software patterns are used to weave parts of the overall software system architecture into a whole. There are others in the software engineering community that agree with this belief – using multiple patterns, similar in nature to a pattern language, to create an entire software architecture. This concept was first presented in Patterns Generate Architecture (Beck, 1994) (Hanmer, 2004).

DOCUMENTING SOFTWARE PATTERNS

Software patterns have been documented in a variety of ways, and there is no consensus in this regard. Again, according to Fowler:

> *"When people write patterns, they typically write them in some standardized format—as befits a reference. However, there's no agreement as to what sections are needed because every author has his or her own ideas…* (Fowler, 2003)

A number of pattern documentation approaches have been developed by software engineers. Based on a literature search, Table 1 represents a survey of some of the most common pattern documenting conventions in use today, most of which come from the software domain.

DISCOVERING PATTERNS

An experienced architect begins to notice that it is not unusual, as one abstracts a systems behavior or capabilities; it begins to look like other systems. That is certainly the case when one looks at transaction based systems. The following example was originally developed to discuss requirement patterns and requirement management, but it can also be used to demonstrate the application of architecture patterns to a sales transaction based system (Kaffenberger, 2004). Let's look at an example of the application of a pattern to an information system.

Pattern Template (Rising, 2003)	Patterns for Effective Use Cases (Adolph, 2003)	Architecture Patterns (TOG0F, 2002)	U.S. Treasury Architecture Guidance (TOGAF, 2002)	A Pattern Language for Pattern Writing (Meszaros, 2004)
Pattern Name	Pattern Name	Name	Name	Pattern Name*
Aliases				*Aliases*
Problem	Problem Statement	Problem	Problem	Problem*
	Metaphoric Story		Implementation	
Context	Context	Context	Structure	Context*
Forces	Forces affecting the Problem	Forces	Interactions	Forces*
			Consequences	*Indications (symptoms)*
Solution	Solution	Solution		Solution*
Resulting Context		Resulting Context	Assumptions	*Resulting Context*
Rationale		Rationale	Rationale	Rationale*
Known Uses				
Related Patterns		Related Patterns		*Related Patterns*
Sketch	Picture			
Author(s)				*Acknowledgements*
Date				
Email				
References				
Keywords				
Example	Examples	Examples		Examples*
		Known Uses		*Code Samples*
* - required, *italics* – optional				

Table 1. A Survey of Pattern Documentation Schemas.

When developing a DFD (Data Flow Diagram) for the first level of decomposition for a small bookstore in England. The diagram in Figure 7 reflects the interactions with the customer, the shop processes, and the data stores. In fact, the individual that developed this example really created a hybrid use case/DFD. It is the first level of decomposition of "Customer wants to buy a book". The numbered circles represent the processes while the items that have horizontal bars above and below the titles represent some form of data storage – whether it is a customer database, a book inventory, or a collection of sales transactions. Finally, each arrow represents the flow of information. From this diagram, it is easy to trace the path from a customer ordering a book, checking the customer's credit, approving the order, checking the book availability, placing the order, and collecting the money for the book; as well as the paths for a number of other interactions.

As an architect looks closer, it becomes apparent that some of the actions can be made more generic (more abstract), and the initial example of buying a book can be abstracted to address the purchase of almost any product. Performing this abstraction, the architect may end up with a "customer purchase" pattern similar to the one shown in Figure 8.

The resulting pattern is generic enough to begin the design of a customer purchase application for almost any domain. The value is that the pattern has been proven over time, and therefore carries a reduced risk of

implementation for any architect willing to adapt it into their enterprise architecture.

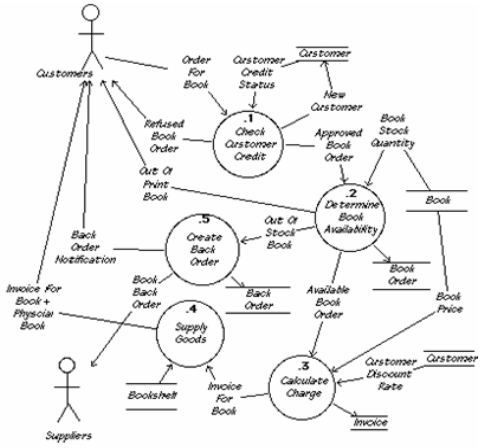


Figure 7. Book purchase diagram.

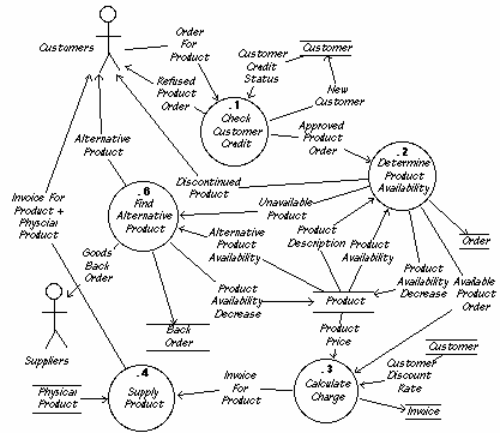


Figure 8. Customer purchase pattern.

To demonstrate the application of this pattern to a new domain, an architect could make changes to the nouns in the customer purchase pattern (Figure 8) to make it useful in a specific domain - whether it is the purchase of an automobile, a service, or even to register for college course. To demonstrate this, Table 2 articulates the minor modifications necessary for this pattern to suggest an initial high level architecture for a class registration service.

Change this noun	To this noun
Customer	Student
Product	Class
Order	Registration
Discount	Scholarship

Table 2 – Applying the Customer Purchase Pattern.

In Figure 8, everywhere the word "customer" appears, the architect would replace it with the noun "student". Where the word "product" appears, "class" would be put in its place, and so forth. Additionally, the architect may want to extend the pattern by adding the ability to keep track of students in each class to system. Finally, it may not make sense to back order classes, so the architect may remove the Back Order and Suppliers functionality from the pattern. The use of such patterns at this level may significantly enhance the R&D efficiency associated with architecting, while also enhancing characteristics such as commonality, testability, and system maintenance.

CAPTURING IMPLICIT KNOWLEDGE WITH PATTERNS

Within industry and government, a growing number of practicing architects have acquired considerable architecture expertise via formal or informal mentoring, and work experience in the work environment. This corporate systems knowledge is captured in explicit ways through mediums such as handbooks, lessons learned repositories, templates and tools, methods and practices, and metrics and measures. A significant component of this corporate knowledge, however, is implicit and undocumented, and largely represented through the technical leaders within an organization. This implicit knowledge is useful to others only if it is shared in a manner that allows its application. The holder of that implicit knowledge may become a bottleneck in applying systems experience on current or future projects (Hole, 2005). Patterns offer a formal method of documentation to capture aspects of such knowledge. If a pattern exists only in the form of implicit knowledge, it is not accessible by others and cannot be used by others without some form of repeated storytelling to convey the

pattern to others by the holder of this knowledge.

While a pattern is reusable by the person who first recognizes it, the real power and value of a pattern is derived only if it can be packaged for use by others. Though a pattern can be transferred through verbal communications, such as storytelling, it is more accurately and reliably transferred through more formal forms of documentation.

POTENTIAL BENEFITS OF ARCHITECTURE PATTERNS

Numerous benefits will result from this growing interest in architecture patterns. One significant derived benefit of patterns, based on precedence in other disciplines, should be improved communications between the various stakeholders. Improved team communications between members of the architecture team and the design teams was the result of using patterns while developing open source software (Hashler, 2004). Another benefit identified from the same study was that patterns facilitated application of sound architectural concepts and implementations. As the discipline of architecting assumes the challenge of developing increasingly complex systems, there is a need for a common lexicon between systems architects. Describing architectures in the context of known and understood patterns should foster better and more consistent understanding across the many stakeholder communities. Systems architecture patterns may also enable implementation of common design features across systems (reuse) leading to enhanced R&D efficiency, and lower ownership costs through reduced efforts with regard to system testing, integration, and maintenance.
In communities that have adopted the use of patterns, the patterns often become standardized through multiple implementations, presentations at research and professional conferences, and publication in research journals. This standardization fosters reuse of designs and even code that might be generated from the architecture patterns. Such reuse can improve development efficiency and productivity (Coplien, 1997). Based on Coplien's study, one could argue that documenting current patterns may reduce the documentation costs and complexity for any organization that elects to pursue systems engineering patterns. Finally, architectural patterns may help control the complexity of architectures by standardizing on well known and practiced patterns.

ARCHITECTURE PATTERN RESEARCH

At this point, we have established that there are many technical disciplines using patterns to manage complexity and reduce risk. Research into the applicability and use of patterns has been underway within the systems architecting and engineering community (Cloutier, 2005) (Cloutier, 2005b).

Once that research established there may be potential benefits supporting the use of patterns at this level, the next question to be asked was "does the systems architect need a different solution for documenting patterns?" This topic was discussed during a colloquia conducted at Stevens Institute of Technology in 2005 (Cloutier, 2005a). Two issues arose from that session that indicated that a unique systems architecture approach is necessary. The first issue is abstraction. The architecture of a system (whether it is an enterprise system within a business or a complex system to be marketed) requires a higher level of abstraction than that found in the software that may be a part of the system. Additionally, many systems include a combination of hardware, software and other resources which may result in pattern uniqueness. This abstraction may make it more difficult to use a simple approach to documenting patterns. The second issue that arose is related to the first – patterns need to address interfaces to non-software parts (if they exist) in the pattern description. This notion of interfaces has not been explicitly addressed in past software pattern discussions.

A survey was conducted in late 2005 to help identify requirements for a methodology for documenting architecture patterns. The survey was provided to over 70 individuals that either has experience using explicit patterns (e.g. software patterns), indicated they use implicit patterns in their work, or has done serious research and thought on the use of patterns for systems engineering/systems architecting. The researcher received 28 useful responses, and a couple of responses where there was a significant amount of narrative and input provided, but the provided data was not

consistent with the survey data, and therefore was not used in this analysis.

SURVEY RESPONSES

The demographics of the responses show a number of notable aspects. First, the responses came from around the world. Eight countries are represented in this data as shown Table 3.

Countries	Totals
Australia	1
Finland	2
Germany	1
Holland	5
Norway	1
Sweden	1
UK	1
USA	16
TOTAL	28

Table 3 - Survey Response Distribution.

The self-described roles of the respondents are shown in Table 4 Some responses characterized their roles with two descriptors – for instance, Director and Architect, which causes the data to look like there are more roles than responses.

Roles	Totals
Architect	13
Director	3
Educator	1
Research Fellow	1
SE	9
SWE	3
TOTAL	30

Table 4 - Respondent's Role.

The industries represented by the respondents are shown in Table 5 The largest industry represented is the aerospace industry. This is not surprising with aerospace representing large, complex systems engineering challenges. There are also ten other industries represented in this data.

Field	Totals
Aerospace/Defense	12
Automotive	1
Commercial IT	1
Consulting/Tool Vendor	2
Education	1
Government	2
Healthcare	1
High Tech (Semi/Materials/Nano)	3
Mechotronics	1
Software	1
Telcom	3
TOTAL	28

Table 5 - Industries Represented by Survey.

Those responding to the survey comprised a very experienced group. While the level of experience ran from 2 years to 48 years, the cumulative years experience represented by the group totals 569 years. Of the 28 responses received, 22 had more than 15 year professional experience, and 18 of the responses had more than 20 years of professional experience.

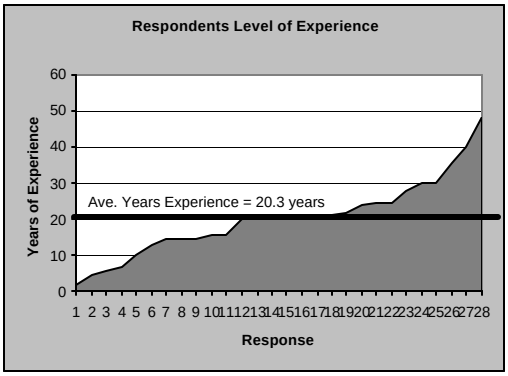


Table 6 - Respondents Experience Profile.

DOCUMENTING ARCHITECTURE PATTERNS

The data demonstrated that systems engineers and architects are most interested in the rationale for the pattern followed by an example of how to apply the pattern, and known uses of the pattern. The researcher broke the data into three distribution groupings to determine the highest rated sections. The distribution provided

for better visibility, facilitating a more clear understanding of which sections were deemed the most important, while also being able to identify which sections are deemed necessary to the majority of respondents though not necessarily having the same degree of importance.

There are several sections that seem to appear on all pattern documentation approaches. Since these sections were pervasive and logical, and they existed in all pattern templates found by the researcher, they were provided in the survey as a necessary given. Those sections are: pattern name, context of the problem, problem description, pattern solution.

Based on the analysis of the survey data, the results indicate that other sections necessary to document architecture patterns include:

- Aliases
- Forces
- Interfaces
- Resulting context
- Related patterns
- Date Documented
- Known uses
- Sketch
- Author(s)
- Email
- Rationale
- References
- Keywords
- Example

Another topic addressed by the researcher in the survey was to determine what form of graphic representation should be used in documenting patterns. Within the surveyed community, the two most common graphical notations are related with the two decomposition approaches (or methodologies) – functional decomposition and object decomposition. The following table represents the diagrams normally associated with the functional decomposition methodology, and the total number of survey responses that believed patterns should graphically represented with these types of diagrams (Table 7).

The second methodology originated in the software community and is referred to as object decomposition (sometimes referred to as logical decomposition). Object decomposition is documented using the Unified Modeling Language (UML). UML is actually methodology agnostic, but it almost always used with an object oriented methodology. The survey asked about what UML diagrams would be most useful in documenting patterns and the result is shown in Table 8.

Most Valuable Functional Decomposition Diagrams	Totals
FFBD	16
Block diagram	11
DFD	9
N2 diagram	6
IDEF0	5
E-R diagram	4
Other - FFDB w/ Data Flow	1
Other - Conops	1

Table 7 - Functional Diagrams for Patterns

Most valuable UML Diagrams	Totals
Use cases	15
Sequence diagram	13
Class diagrams	12
Activity diagram	12
Collaboration diagram	6
Composite Structure	5
Object diagram	4
Parametric diagram	3
Other - State diagram	3
Constraints diagram	2
Other - Component diagram	1
Other - Deployment diagram	1
Other Block diagram	1
Other - Domain Model	0

Table 8 - UML Diagrams for Patterns.

Based on this data, it appears the most useful graphical diagrams for use in documenting architecture patterns are FFBD, Block Diagram, DFD, N2 and IDEF0 for the functional decomposition. For the object decomposition approach, the most useful UML diagrams may be Use cases, Sequence diagrams, Class diagrams, Activity diagrams, Collaboration diagrams and Composite Structure diagrams.

EXAMPLE: THE C2 PATTERN

The following example takes the command and control (C2) process, sometimes referred to as plan, detects, control and act. Though this pattern is normally associated with designing systems for the military, it is easily applied to the design of a police or fire organization for large cities like Los Angeles or New York. It could also

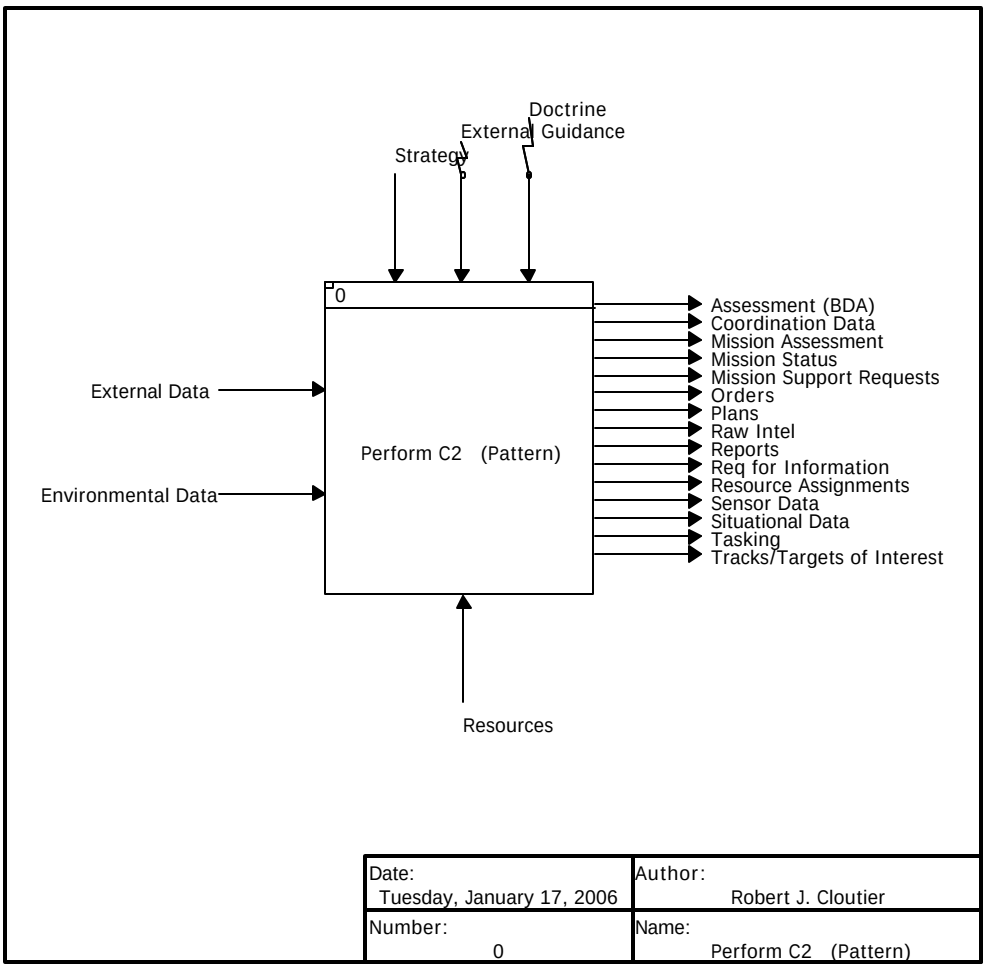
be applied to a command and control system for transportation systems like the railroad or a trucking fleet with a little creativity.

As in this case, the pattern documentation may include a model from a modeling tool such as Vitech Corporation's Core (used to generate these diagrams) or a UML modeling tool. The inclusion of a model file certainly will help "jump start" the new architecture.

Documenting an Architecture Pattern

Pattern Name:	Perform C2, Perform Command and Control
Aliases:	None known
Keywords:	Command and control, Plan, Detect, Control, Act, C2.
Problem Context:	Does not address "Prepare" precondition (though one might argue that prepare and plan go together) nor "Assess" post condition
Problem Description:	In command and control (C2), the situation is typically managed in identifiable phases. And, the situation may move back and forth between the stages. Those stages are Plan/Detect/Control/Act
Forces:	Terminology may vary from one domain to the next, and should be adapted in the application of the pattern
Pattern Solution:	This pattern provides the basis for developing the command and control (C2) interfaces and information that moves through the stages of C2. It provides the A0 Context and the first level of decomposition using IDEF0.
Model:	See next page
Interfaces:	Information flows between the stages of this pattern, as well as feedback loops. Some information is generated only in a particular stage and then output in the form of reports. Names of information can be modified as required by specific domain application.
Resulting Context:	Further work is required to define the tasks to be performed within each stage, and the allocation of tasks to systems, hardware, software or people.
Example:	This pattern may be used in the modeling of a C2 system for military or paramilitary operations system (such as police or homeland defense) where there would be a planning phase, a detection of an situation or "bad guy", an identification and controlling or managing of the information, and a required action to perform the mission. May even be extended to motor vehicle fleet operations.
Known Uses:	Command and Control applications
Related patterns:	OODA (Observe, Orient, Decide, Act)
References:	MCDP 6 Marine Corps Command and Control Handbook
Pattern Rationale:	This is a time tested doctrine used by the military, that may be applicable to other domains
Author(s):	Harry Johnson Ph.D., Ken Hartnett, Satya Moorthy, Robert Cloutier, 2006.

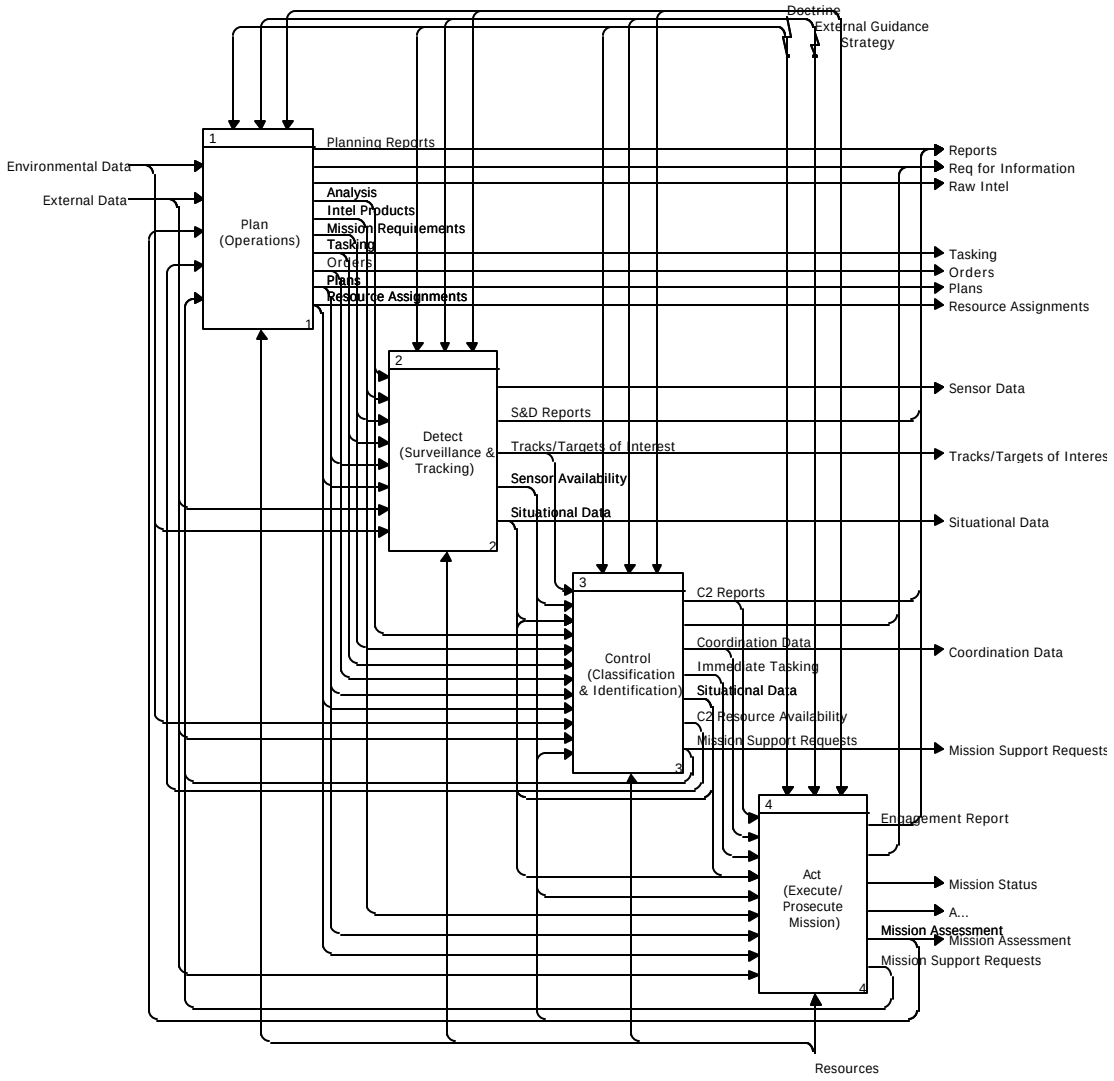
Pattern Description: Perform C2 **Page 1 of 3**



Pattern Description: Perform C2 **Page 2 of 3**

Note: At this level of the IDEF0 diagram, the architect documenting the pattern identifies the inputs and outputs (left to right). The resources are drawn from the bottom. In this case, it may be people or equipment. The controlling factors are drawn from the top. Examples of controls may be strategies to be employed, guidance from the commander or police chief.

Customizations when implementing this pattern may be to change the term "Tracks/Targets of Interest" to "suspect" for a police C2 system, or maybe "truck number" for a trucking fleet implementation. Sensor data may be humans in a stakeout, military satellite tracking (or quite frankly, satellite tracking in the form of a GPS device on a car or truck). Examples of external data may be intelligence, or a tip from an informer which will have some impact on the operation.



Pattern Description: Perform C2 **Page 3 of 3**

Note: There is a lot of data here. Each box represents a decomposition of the perform C2 top level function. Outputs are taken from one function and become inputs to another function.

Though an architect documenting a pattern of this magnitude could do a further decomposition of each of these functions into lower level functions, the authors experience has shown that the detail begins to become too detailed, and less able to be abstracted.

A PROPOSED PATTERN HIERARCHY

Patterns can be applied at different levels of an architecture or system based on the appropriateness of the pattern and the existing maturation of the system being developed. For instance, within the software community, there are system patterns (sometimes referred to as architecture patterns by the software folks), design patterns and idioms. Figure 9 represents a pattern hierarchy proposed by van Zyl and Walker for software systems (van Zyl, 2004).

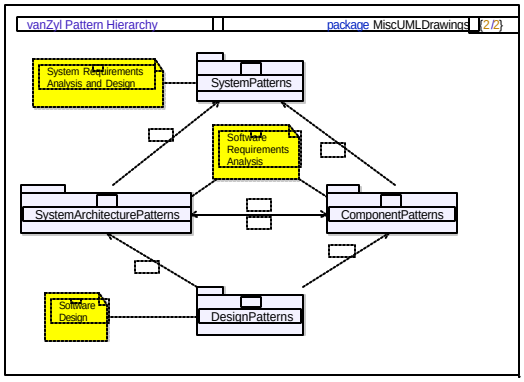


Figure 9. van Zyl Pattern Hierarchy.

It is important to recognize that in their work, when they refer to the system, they are referring to the software system. Their work may be extendable to the broader system, or system of systems architecture. System level patterns may be applicable when representing the highest levels of a system to represent an entire system or a part of a system. They may also include structure and system boundaries. Based on precedence from other disciplines, use of patterns in systems engineering and architecting may provide the foundation for a more common lexicon leading to improved communications between the various stakeholders, while also enhancing the R&D efficiency on complex development programs. For instance, the Publisher-Subscriber pattern, the Layers pattern and the Client-Server pattern were all first published as software patterns (Buschmann, 1996). Now, most enterprise architects understand those patterns, and use them to describe systems – a common lexicon.

The pattern hierarchy proposed by van Zyl (2001) can be extended for broader application and increased relevance to larger systems. Extending this hierarchy, Figure 10, shows the five types of system architecture patterns identified in this research (Cloutier, 2005). It also provides a necessary bridge to show the relationship between patterns that the Business process groups have been identifying for years and the software patterns we have already discussed. In this proposed taxonomy, system architecture patterns are broken into:

1. Structural patterns
2. System Requirements patterns
3. Systems Engineering Activities patterns
4. Systems Engineering Roles patterns
5. System Process patterns

Structural patterns provide a physical pattern to follow when designing a part of the architecture. System requirements patterns prescribe the format of a properly formed requirement, or a collection of requirements that can be reused to describe desired functionality. Activities patterns, also described as organizational process patterns, indicate how the process of architecting or systems engineering is performed. Finally, roles patterns help describe how the architecting role is performed.

WHEN TO NOT USE ARCHITECTURE PATTERNS

The discussion thus far has focused on the positive aspects of documenting and using patterns and the potential advantages of applying them within the system architecting discipline. For completeness, this discussion should also present the likely pitfalls associated with the application of patterns.

The first argument against the use of patterns relates to the notion of implicit structural constraints inherent in the use of patterns – particularly with highly independent and creative individuals, and the related impact on innovation. This concern is valid and must be balanced – the intent is to leverage and reuse existing solutions (albeit, abstract) when applicable, and to allow the necessary degrees of freedom to explore new and unique implementations when required and desired to ensure an atmosphere that encourages creativity.

The second argument may be described as "so what?" Sometimes, when an expert architect

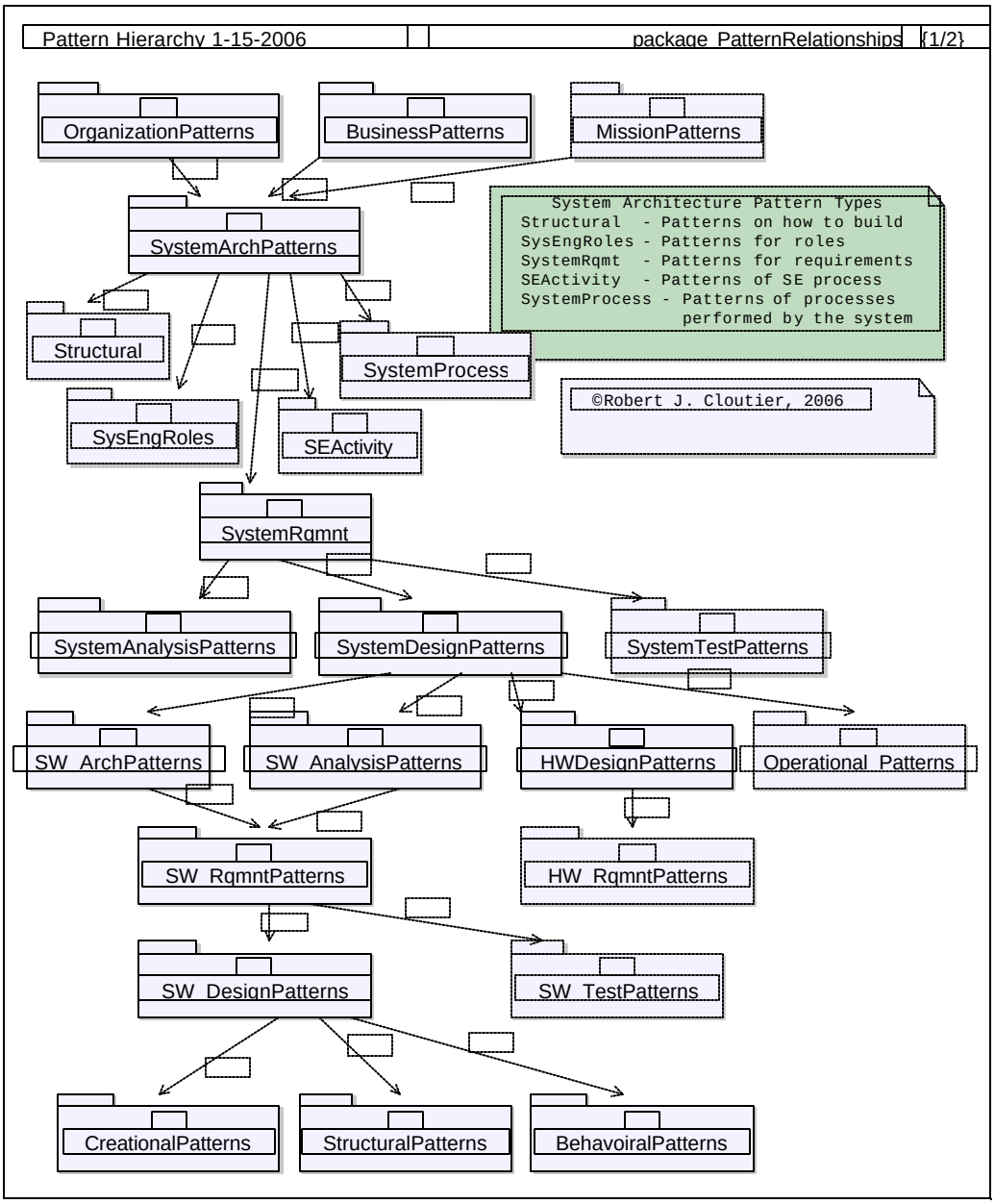


Figure 9. A proposed system pattern hierarchy.

looks at patterns, the reaction is – "so what, I knew that". The fact of the matter is that within their domain of expertise, patterns are of little use to experts. This attitude results in lack of inertia when it comes to adopting the use of patterns – particularly in regard to the documentation and validation of patterns by these experts for use by others. Organizational motivation usually becomes necessary. From the perspective of preserving, sustaining, and evolving corporate knowledge, patterns are a powerful medium for capturing aspects of implicit knowledge in a form that is pragmatic.

There are situations when it is inadvisable to use architecture patterns. These include:

- New or unique requirements (unprecedented systems) preclude the existence of a pattern,
- Unique solution (unprecedented concepts),
- When the pace of technological change does not warrant the use of patterns

For designs that are proven and effective, and addressing problems common across multiple systems and domains, there should be a strong motivation to leverage the benefits that can accrue from the application of patterns.

SUMMARY AND CONCLUSIONS

Patterns are models, or abstractions of reality. Today's systems have become extremely complex. It is difficult, if not impossible, for a systems engineer to mentally juggle all the details of a modern complex system. As already demonstrated, patterns can exist at multiple levels. During the course of systems architecture, design and implementation, a project team may use systems architecture patterns, process patterns, design patterns, implementation patterns (for software code), machine patterns (to cut metal for cabinets), test patterns, and validation patterns. At each level of the architecture, the pattern should contain the appropriate level of detail for the stage in which it is applied. However, patterns are not silver bullets. But they can be a powerful tool in the architect's toolbox. They help solve difficult problems by leveraging existing knowledge, if this existing knowledge is documented in a manner that facilitates this process.

In communities that use patterns, the patterns often become standardized through use in designs and technical discussion, presentations at research and professional conferences, and publication in research journals. This standardization has fostered reuse of architectures, designs and even code.

As was shown in the C2 pattern, sufficient detail can be captured in the pattern to enable an architect less familiar with the process to begin architecting a solution to a system with similar requirements. However, as the architect documenting the pattern "drills down" while documenting a complex pattern, a level of detail is reached whereupon the necessary detail begins to obfuscate the value of pattern abstraction. This is the point in which no further data should be added to the pattern An additional benefit to patterns at this level is the speed of knowledge transfer to train new systems engineers faster than working on multiple programs over several years. Patterns are one way to help minimize the possibility that details will not "fall through the cracks".

Continued research should be done on architecture patterns. It would be interesting to see if agent based modeling of the use of patterns may help in the quantification of the value of patterns in the architecting process.

AUTHOR BIOGRAPHIES

Rob Cloutier

Rob works for Lockheed-Martin Corporation in Moorestown, NJ where he is a Principle Systems Engineer in the System of Systems Architecture organization. He is responsible for developing architecture for complex systems. Rob is a Systems Engineering Doctoral Candidate and occasional guest lecturer at Stevens Institute of Technology in New Jersey. He is also developing an Object Oriented Architecture and Design course for Stevens Institute.

Rob holds a B.S. from the United States Naval Academy and an M.B.A. from Eastern University where he is an adjunct professor. He has over 20 years experience in systems engineering, software engineering, and project management in both commercial and defense industries.

Dinesh Verma

Dinesh received his Ph.D. and the M.S. in Industrial and Systems Engineering from Virginia Tech. He is currently serving as the Associate Dean for Outreach and Executive Education, and Professor in Systems Engineering at Stevens Institute of Technology. He concurrently serves as the Scientific Advisory to the Director of the Embedded Systems Institute in Eindhoven, Holland. Prior to this role, he served as Technical Director at Lockheed Martin Undersea Systems, in Manassas, Virginia, in the area of adapted systems and supportability engineering processes, methods and tools for complex system development and integration.

Before joining Lockheed Martin, Verma worked as a Research Scientist at Virginia Tech and managed the University's Systems Engineering Design Laboratory. While at Virginia Tech and afterwards, Verma continues to serve numerous companies in a consulting capacity, to include Eastman Kodak, Lockheed Martin Corporation, L3 Communications, United Defense, Raytheon, IBM Corporation, Sun Microsystems, SAIC, VOLVO Car Corporation (Sweden), NOKIA (Finland), RAMSE (Finland), TU Delft (Holland), Johnson Controls, Ericsson-SAAB Avionics (Sweden), and Motorola. He served as an Invited Lecturer from 1995 through 2000 at the University of Exeter, United Kingdom. His professional and research activities emphasize systems engineering and design with a focus on conceptual design evaluation, preliminary design and system architecture, design decision-making, life cycle costing, and supportability engineering. In addition to his publications, Verma has received one patent and has two pending in the areas of life-cycle costing and fuzzy logic techniques for evaluating design concepts. Dr. Verma has authored over 85 technical papers, book reviews, technical monographs, and co-authored two textbooks: Maintainability: A Key to Effective Serviceability and Maintenance Management (Wiley, 1995), and Economic Decision Analysis (Prentice Hall, 1998). He is a Fellow of the International Council on Systems Engineering (INCOSE), a senior member of SOLE, and was elected to Sigma Xi, the honorary research society of America.

REFERENCES

Adolph, S. Bramble, P. (2003). Patterns for Effective Use Cases. Boston: Addison Wesley.

Ambler, Scott, "The Process Patterns Resource Page". Retrieved October 28, 2004 from http://www.ambysoft.com/processPatternsPage.html

Alexander, Christopher. (1964). Notes on the Synthesis of Form. Cambridge: Harvard University Press.

Alexander, Christopher. (1977). A Pattern Language. New York: Oxford University Press.

Alexander, Christopher. (1979). A Timeless Way of Building. New Your: Oxford University Press.

Baer, William C. (2003). "How 'Pattern Books' Fueled England's First Speculative Real Estate Market". Harvard Business School Working Knowledge eZine, 2003. Retrieved May 7, 2004 from http://hbsworkingknowledge.hbs.edu/tools/print_item.jhtml?id=3313&t=finance

Beck, K., Johnson, R. (1994). "Patterns Generate Architectures". Proceedings Object-Oriented Programming 8th European Conference, ECOOP '94 (Bologna, Italy, 1994) Springer-Verlag, pp. 139-149.

Bernard, Scott. (2005) An Introduction to Enterprise Architecture (2nd Edition). AuthorHouse. Bloomington, IN.

Buschmann, F., R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal. (1996). Pattern-Oriented Software Architecture: A System of Patterns. West Sussex, England: John Wiley & Sons Ltd.

Cloutier, Robert J. (2005). Application of Patterns to Systems Architecting. Proceedings and presentation, 2005 Telelogic Americas User Group Conference, October 2005.

Cloutier, Robert J. (2005a). Survey on the Use of Patterns in Systems Engineering and Architecting Research Colloquium. CSER 2005. March 23, 2005.

Cloutier, Robert J.(2005b). "Towards the Application of Patterns to Systems Engineering". CSER 2005. March 23-25, 2005.

Coplien, James O. (1997) "Idioms and Patterns as Architectural Literature". IEEE Software Special Issue on Objects, Patterns, and Architectures. January 1997.

Devedzic, Vladan. (1999) "Ontologies: borrowing from software patterns". Intelligence, Vol. 10, number 3, pages 14-24. ACM Press, 1999. Available at: http://doi.acm.org/10.1145/318964.318968

Fowler, Martin. (2003) "Patterns". *IEEE Software*. March/April 2003.

Gamma, E., Helm, R., Johnson, J., Vlissides, J., (1995). Design Patterns: Elements of Reusable Object Oriented Software. MA: Addison-Wesley.

Hahsler, Michael. (2004). Free/Open Source Software Development. Edited by Koch Stefan. Idea Group Publishing. Pages 103-124.

Hanmer, Robert S. and Kocan, Kristin F., (2004) "Documenting Architectures with Patterns" Bell Labs Technical Journal 9(1), 143-163, Wiley Periodicals, Inc., 2004.

Hole, Eirik. (2005) "Architectures as a Framework for Effective and Efficient Product Development in a Dynamic Business Environment". Proceedings of the 2005 Conference on Systems Engineering Research, March 2005.

Kaffenberger, Ruediger. (2004). "The Difference – On the Use of Pattern-Based Requirements". 14th Annual International Symposium Proceedings, INCOSE 2004.

Meszaros, G. Doble, J. "A Pattern Language for Pattern Writing". Retrieved May 28, 2004 from (though available from numerous websites) http://webclass.cqu.edu.au/Patterns/Resources/writers/language/

Rising, Linda, PhD. (2003). Pattern Template. AG Communications Systems (A subsidiary of Lucent Technologies). Retrieved May 28, 2004 from http://www.agcs.com/supportv2/techpapers/patterns/template.htm

Sachdeva, N. and Godlszmidt, G. (2004). "On demand business process life cycle, Part 2: Patterns for e-business recipe". IBM Developerworks Website. 24 Nov 2004. Retrieved June 5, 2005 from http://www-106.ibm.com/developerworks/library/ws-odbp2/

TOGAF. (2002). "Architecture Patterns". The Open Group Architecture Framework (TOGAF) Website. Retrieved May 27, 2004 from http://www.opengroup.org/architecture/togaf8-doc/arch/p4/patterns/patterns.htm

van Zyl, Jay & Walker, A.J. (2004). "A Pattern Architecture: Using Patterns to Define Overall Systems Architecture". Retrieved October 22, 2004 from http://osprey.unisa.ac.za/saicsit2001/Electronic/paper37.pdf

Zachman, J. (1987). A Framework for Information Systems Architecture. *IBM Systems Journal*. Vol. 26, No. 3, pp. 276-290.