Applying the Concept of Patterns to Systems Architecture

Robert J. Cloutier* and Dinesh Verma†

Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ 07030

ABSTRACT

While much has been written about patterns in software engineering, little has been written about their application to systems architecting. This paper provides a discussion of patterns and their potential applicability to complex system architecting. A historical introduction to the concept of patterns is provided along with their evolution from the domain of civil architecture to other engineering disciplines and domains. The relevance and applicability of patterns to systems architecting is then examined. Research with regard to developing a pattern form for documenting patterns for systems architecting is presented, and this is demonstrated on a command and control pattern, using both IDEF0 and UML. The application of this pattern within a functional architecture is then explored. Finally, recommendations for the development and management of a systems architecting and architecting pattern repository are offered. © 2007 Wiley Periodicals, Inc. Syst Eng 10: 138–154, 2007

Key words: systems architecture; architecting; pattern; documenting patterns; pattern repository; knowledge

1. INTRODUCTION AND DEFINITIONS

Today’s complex systems make it difficult for systems architects to mentally juggle the system details and nuances. Models help in that arena, and patterns are examples of models, or abstractions of reality. As systems become more complex, the need for a method to capture and manage implicit knowledge within corporations and within the systems architecting discipline grows. This implicit knowledge should be socialized to such an extent that it can be made explicit. Furthermore,
this explicit knowledge needs to be properly documented such that it can be applied when relevant to solve similar problems. Patterns may be a partial solution to both these needs. This paper will present patterns as one tool for capturing implicit knowledge for use by systems engineers to develop system architectures. There are a number of concepts and practices already in the systems engineer’s toolbox today. These include documented processes such as CMMI® (Capability Maturity Model Integration® as defined by the Software Engineering Institute), heuristics, templates, and frameworks to name a few. While these tools may appear to be patterns in the broadest sense, they do not contain the richness of information found in modern day patterns. The remainder of this section will review some key pattern concepts as well as terms that are sometimes confused with what patterns have become, based on the continually maturing patterns community of other engineering disciplines.

For the context of this paper, that the authors have previously defined a systems architecture pattern as “a high-level structure, appropriate for the major components of a system. It expresses the relationship between the context, a problem, and a solution, documenting attributes and usage guidance. Patterns are time-proven in solving problems similar in nature to the problem under consideration” [Cloutier, 2006, p. 107, 2005c, 2005b].

1.1. Heuristics

A heuristic is sometimes referred to as a rule-of-thumb. Rechtin and Maier state that heuristics are very general, spanning domains and categories of guidance [Rechtin, 1991; Rechtin and Maier, 1997]. They go on to state that heuristics are imprecise and give the least guidance to the novice. Koen [2003, p. 28] defines a heuristic as “anything that provides a plausible aid or direction in the solution of a problem but is in the final analysis unjustified, incapable of justification, and potentially fallible.” A couple of examples presented by Koen may be helpful in understanding the concept of heuristics. “If we are unable to find a single example of a concept, we should suspect it may not exist.” (p. 146) Yet another example is “A properly designed bolt should have at least one and one-half turns in the thread.” (p. 167)

Heuristics represent the capture of implicit knowledge. For instance, one heuristic cited by Koen [2003, p. 35] is “at some point in the project, freeze the design.” Though this is good advice, it is very general in nature, nor is it expressed using Alexander’s basic form. That is not to say that it could not be put in the Alexandrian form—but that would be of little further value. Moreover, if the design is simple enough, or being worked by a single individual, there may be no need to freeze the design. Finally, it does not satisfy the definition in a number of ways, to include documenting attributes and usage guidance.

Rechtin and Maier [1997] described heuristics, patterns, styles and design methods as a progression, each becoming more prescriptive. Patterns were still gaining momentum when Rechtin and Maier addressed them in the context of systems architecting, and, though they cite the Alexandrian form (template), and state that since the form and definition of a pattern is quite general, they state that pattern concepts can be applied to other forms of architecture. They also believed that patterns are prescriptive heuristics, describing particular choices of form and their relationship to particular problems. Later in this paper, it will be shown that architecting patterns can be used well beyond the domain in which they were originally identified.

1.2. Templates

The relevant definition from the American Heritage Dictionary defines a template as “[a] document or file having a preset format, used as a starting point for a particular application so that the format does not have to be recreated each time it is used.” Said another way, a template is a document that contains topic headings, but no content. In the pattern community, the pattern form is the pattern template.

1.3. Processes

ISO 15288 defines a process as a set of interrelated or interacting activities, which transform inputs into outputs (referencing ISO 9000:2000). Most companies have documented processes covering everything from employee relations to how to engineer a system. The CMMI® provides a product suite developed specifically for those users who are system and product developers and want to improve their processes and products. Processes tend to be prescriptive in nature. Once adopted, one must follow a prescribed set of steps to change or tailor the process for use on a specific project or program. The desired changes or tailoring may also require approval by a governing body.

1.4. Frameworks

A framework is a logical, organizing structure used to classify information, concepts, data, etc. They may also provide mechanisms to transform information from one form to another. Gamma et al. [1995] defined a framework as a set of cooperating classes that make up a reusable design for a specific class of software. A frame-
work provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes the framework to a particular application by subclassing and composing instances of framework classes. Another popular framework was proposed by Zachman [1987]. It addresses the concept of information systems architecture in the form of a collection of architecture views and perspectives, and the products, depicted in a matrix. The Department of Defense Architecture Framework (DODAF) is a framework used by the U.S. Military. It captures complex systems architecture in a number of artifacts, and organizes those artifacts in three separate views—an operational view, a systems view and a technical view.

1.5. Pattern Languages

A pattern language is a collection of patterns that are complimentary to one another. In another reference, Alexander [1979] discussed a pattern language in terms of mathematics a set of elements (or symbols), and a set of rules for combining these symbols. Alexander [1977] also portrayed a pattern language as a network of larger patterns, comprised of smaller patterns. An explanation offered by Alexander [1979] for a pattern language is the collection of patterns that would be useful for designing a garden. The collection of patterns chosen to design a garden may include: Terraced Slope, Tree Places, Greenhouse, Garden Seat, Building Edge, and Sunny Place. By assembling these individual patterns from Alexander’s garden pattern language, any number of beautiful gardens could be assembled, containing elements of trees, sun, sitting, a building, etc. If one thinks of patterns as the elements, and the combination of patterns (or transitions from one pattern to another, representing interfaces, a pattern language emerges. This notion of a pattern language, constructed of smaller patterns will be revisited again in section 5.3.

2. PATTERNS—A HISTORICAL PERSPECTIVE

Most people in the pattern community attribute Christopher Alexander [Alexander, 1977] as the first to formalize and articulate the concept and value of patterns. Throughout the ‘60s and ‘70s, Alexander, a civil architect, strove to improve on the art of urban design by creating patterns that other architects could use. “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution…” [Alexander, 1979, p. 247]. That is to say, in Alexander’s view, one must capture all three elements to document a pattern. Alexander began to identify patterns for civil architecture, urban design, and community planning by looking at an architectural design and then abstracting that design into its basic parts that were common across other designs. Communities were viewed as systems that could be decomposed into component parts. He described the component parts and their relationship to other component parts in terms of boundaries. Alexander first published these notions in Notes on the Synthesis of Form [Alexander, 1964]. In this seminal work, he began what has resulted in several thousands of pages of published works in which Alexander investigates the process of architecture development. Examples of reusable patterns identified and tested by Alexander include patterns such as (1) 6-foot balcony and (2) light on two sides of every room. The first pattern captures the design elements, in the form of context, problem, and solution, to provide the minimum requirements for a useful balcony. Though the pattern seems self-explanatory from the title, Alexander [1977: 782–784] spent 3 pages documenting the pattern. Likewise, “light on two sides” captures the design approach that when architecting a room, one should provide architecture elements that allow natural light from two directions—be it windows, skylights, or doorways. Alexander did not invent these patterns, they came from observation and testing; and only then were they documented as patterns.

2.1. Alexandrian Pattern Examples

To demonstrate the application of the pattern concept, Alexander outlined a collection of patterns to govern the architecture of a farmhouse [Alexander, 1977]. The names of those patterns are:

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

As one reads through the listed patterns, a visual picture begins to develop in the mind’s eye, creating an image 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” (Fig. 1) pattern was characterized as follows: “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” [Alexander, 1977, p. 386]. The “house for a small family” pattern (Fig. 2) context was: “In a house
for a small family, it is the relationship between children and adults which is most critical” (while these examples do not contain the entire patterns, as documented by Alexander, they do convey the essence of the patterns) [Alexander, 1977, p. 382].

In developing the concept of patterns and researching their applicability, Alexander 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, and if one were to replace the word “house” with “system,” the same concept could apply to the notion of system architecture patterns.

2.2. Patterns Are Not Created

It should be noted that patterns “are not created from a blank page; they are mined” [Hanmer and Kocan, 2004]. Hanmer explains, when discussing design patterns, over time, similar designs are arrived at independently by different designers on various projects. As it becomes apparent that the same design elements exist in multiple designs, the design can be studied and documented to encourage reuse. This reuse prevents the reinvention of the same concept, and provides a vocabulary for the design concepts that a project can share [Sanz and Zalewski, 2003]. Consistent with the notion that patterns are not created, but instead mined, the rule of three was introduced by Ralph Johnson. He stated that a pattern is not a pattern unless there are at least three independent, observable applications utilizing the proposed pattern as part of the solution (though this was fully documented as a pattern, it may also be approached as a heuristic). Appleton [2000] expanded on this “rule of three,” stating that there are some in the software patterns community that believe it is inappropriate to identify a solution as a pattern until that proposed pattern is vetted, or scrutinized, by others. This scrutiny should give the user of the pattern some confidence in the completeness and appropriateness of the pattern, which may also reduce risk in a system employing the applied pattern. Once a pattern is identified and believed to be of use in the future, it should be documented using a pattern form.

2.3. Patterns Usage and Pattern Form

Based on the literature and usage of patterns in other engineering disciplines, patterns are not meant to produce “cookie cutter” architectures or designs. Instead of using patterns exactly as documented, they are meant to be a starting point from which the final product will be crafted. Alexander believed that in some settings, patterns might be reused tens of thousands of times, and not have any two designs look exactly the same. “We may then gradually improve these patterns which we share by testing them against experience…” [Alexander, 1979]. The benefit of this approach to patterns is that they should not remain stagnant, but instead be improved through continuous application.

The Alexandrian form of documenting patterns contained four topics to be addressed. They are: (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 that 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.

The intent of this section was to provide a historical context of where the modern notion of pattern use by engineering disciplines originated. Some of Alexander’s concepts are more than 30 years old. The discipline of systems engineering was just beginning to emerge then. Systems were much less complex then. Today’s modern automobile has more computing power than the Gemini spacecraft of that era, and the personal computer had not yet been invented. Like other engineering concepts, the concept of patterns has matured and evolved with time. In reading Alexander’s voluminous body of work, his notion of patterns changed too [Alexander, 2002]. So it is with the application of patterns in other engineering disciplines. Alexander’s early civil architecture pattern concepts provide a good reference foundation.

3. KNOWLEDGE AND PATTERNS

Today, there is already some carryover from the software patterns community into the systems architecting discipline. As an example of this, many systems engineers today understand what is meant by a client-server based architecture. The same is true for any number of patterns that have been abstracted to the systems level. The net effect of this discipline carryover is a common lexicon between systems and software engineers to exchange knowledge.

3.1. Implicit Knowledge

Though one can attain advanced degrees in systems engineering, there are many systems engineers today that do not have a formal systems engineering degree. Instead, they have acquired experience and knowledge to be called a systems engineer by working on a number of development projects, and by being able to think about how parts of the system interact, both positively and negatively, with one another. Corporate systems engineering departments attempt to capture knowledge explicitly through artifacts such as handbooks, lessons learned repositories, engineering project notebooks, templates and tools, methods and practices, and metrics and measures. A significant component of this corporate knowledge, however, is implicit and undocumented, and largely held by the technical leaders. This implicit knowledge is useful to others only if it is shared in a manner that allows its application. According to Hole [2005], the holder of that implicit knowledge may become a bottleneck in applying that systems experience on current or future projects.

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. The real power and value of a pattern is derived only if it can be packaged for use by others. If the pattern is not documented (written down), it may increase the possibility that the complete pattern will not be implemented the next time it is used—something may be forgotten or inadvertently left out. Formalizing patterns through documentation helps to ensure their usage by others, their management as assets, and their application in an appropriate manner, and that they are improved over time.

3.2. Explicit Knowledge

Though patterns and heuristics are cited as serving to improve communications between teams [Rechtin, 1991; Rechtin and Maier, 1997], Hashler [2004] may be the only documented, quantitative source in documenting improved communications between members of the architecture and design teams resulting from the use of patterns. He also found that they facilitated the application of sound architectural concepts and implementations. The sound architectural concepts identified by Hashler were the software patterns documented by Gamma et al. [1995]. As the discipline of systems 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.

4. USE OF PATTERNS IN ENGINEERING

We have discussed the notion that patterns represent implicit knowledge, which if documented, can be reused by others with the intent of managing the cognitive load while architecting complex systems. In this section, we will explore some examples of pattern usage in a number of engineering disciplines beyond software engineering. This section will briefly discuss some applications of patterns across a number of engineering disciplines.
4.1. Software Engineering

Patterns were adopted by the software engineering community in the late ’80s when software designers began to apply Alexander’s architectural concepts (patterns) to object oriented software development. In 1995, Gamma et al. published what is considered to be an authoritative work on software design patterns—Design Patterns: Elements of Reusable Object-Oriented Software [Gamma et al., 1995]. A recent search on Amazon.com for books on “software patterns” returned 1883 choices. Software modeling tools such as IBM’s Rational Software Architect includes a number of software patterns that can be applied, and the ability to add new patterns. The software community may represent the broadest application of patterns in the community of engineers today.

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. Using software patterns early in the design reduces the number of changes that have to be made later in the lifecycle [Devedzic, 1999]. Devedzic also considers some software patterns to be “micro-architectures” that contribute to the overall software system architecture. He makes that assertion based on software patterns being used to weave parts of the overall software system architecture into a whole.

4.2. Telecommunications

Bell Laboratories began mining their embedded systems for patterns, and then documented those patterns. What they found was that there were commonalities in fault tolerant and high availability systems. By identifying and documenting patterns that existed in the minds of their long-standing experts [implicit knowledge], they could document their core competencies [Coplien, 1997]. This is a strong argument for further exploration into the identification and use of architecture patterns—to enable the capture of good architectural concepts and implementations and to preserve them for future projects. In other words, patterns may represent a knowledge capture methodology that provides value to a business. Another value-added trait cited in support of patterns is that they provide a common lexicon between architects. A common understanding of the architecture is fostered by describing parts of the design and its implementations in the context of known and understood patterns.

4.3. Requirements Engineering

Large-scale system requirements difficulties have been identified as one of the three major factors that contribute to poor systems integration [Kim, 1999]. Kim goes on to explain that requirements are often “inconsistent, incomplete and emphasize performance without regard to cost” [Kim, 1999]. Existing requirements engineering methods, as applied to today’s complex systems, still pose the problem of missing, ambiguous, or misleading requirements [Kaffenberger, 2004]. According to Kaffenberger, patterns offer a better way to specify system requirements. He identified two types of requirements patterns: (1) patterns that describe the constituent elements of a single requirement statement and their relations and (2) patterns that describe a set of requirements statements and their relations between them. In his paper, Kaffenberger actually identified the first type of pattern as a requirements template, consistent with the definition presented in the introduction of this paper. The goal of his template was to improve the quality of the individual requirements statement.

The second type of requirements patterns were broken into two subcategories—requirements process patterns and requirements data patterns. According to Kaffenberger, by separating the system process essentials from the system products, patterns emerged in the handling of the products. Patterns were then identified for the processes and for the data structure of the information contained in the products. These patterns were reused in subsequent systems development projects.

In 2002, there was a special interest working group on requirements engineering of the German Informatics Society, four presentations representing four different projects from four different domains. However, all four programs were facing similar challenges:

- Stakeholders had to be convinced of the necessity and benefits of requirements engineering.
- Cultural differences of stakeholders had to be accommodated.
- Requirements engineering methods and tools had to be established.
- Skepticism regarding processes and transparency had to be overcome.

As a result of this working group, The Requirements Engineering Pattern Repository (RE Pare)1 was created in 2004. This online repository is a good model for a pattern repository.

---

1 See http://repare.desy.de.
4.4. Control Systems Engineering

Software now represents a significant portion of control systems. This has required the marriage of control systems engineers and software engineers to produce the necessary control system software. To facilitate this situation, Sanz and Zalewski [2003] have proposed using software “design patterns to document, transfer, and exploit [control system] design knowledge.” They are using software design patterns to capture both classic control system designs and promising new designs and see the use of patterns as a new way to manage and exploit existing design knowledge. Further, they have found that patterns enhance functional properties and especially nonfunctional properties of control systems designed using patterns. They presented a computer based feedback control system pattern, and according to Sanz and Zalewski [2003], the pattern mindset is not a search for the Holy Grail, but rather, a “continuous effort of thinking generic when designing, creating designs—patterns—that will last because they demonstrate wide applicability.”

5. SYSTEMS ARCHITECTURE PATTERNS

The purpose of this research is to provide a framework and methodology to document and manage some aspects of implicit knowledge in the form of system architecture patterns and to understand whether the positive benefits of enhanced productivity and effectiveness observed in other disciplines apply to systems architecting. Based on precedence from other disciplines (as well as parts of systems engineering), use of patterns in systems 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. Other disciplines that have adopted the use of patterns simply chose one of the pattern forms already in use by the software community, which originated from the Alexandrian form. However, systems architects may need different or additional information than that required by a software engineer to implement a pattern. For instance, the infrastructure of a system may include a mechanical system that has a rotating part that resonates at 60 Hz. That resonance may manifest itself in interference in an electrical system through inductance. Additionally, it seems reasonable that the more abstract the concept, the more thorough the documentation must be to enable reuse to solve similar problems.

Rechtin [1991] discussed the concept that a series of unprecedented systems establishes a precedent, and that the precedent establishes recognized patterns, sometimes called architectures. Each unprecedented system has an architecture, which represents the organizing structure of that specific unprecedented system. The difference between architectures and patterns is that while certain architectures are reusable, most architectures include details specific to the set of concerns for which the specific architecture was synthesized. A pattern abstracts (hides) specific details and concerns found in any number of unprecedented systems, keeping the common, reusable characteristics that occur across the unprecedented systems. These common characteristics which provide the organizing structure of the common aspect of the architecture is documented to address the generalized set of requirements in order to foster reuse or application to an entirely different problem with similar concerns.

5.1. Potential Benefits of Patterns to Systems Architecting

Numerous benefits may result from the growing interest in architecture patterns. For instance, Haskins [2005] has proposed creating a patterns language from patterns that may be found in the INCOSE Systems Engineers Handbook [2005]. One significant derived benefit of patterns, based on precedence in other disciplines, should be improved communications between various stakeholders. Improved communications between team members of the architecture and design teams was a measured and quantified 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 by reducing the effort required for system testing, integration, and maintenance. According to Coplien [1997], 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 cost of creating and documenting architectures, for any organization that elects to pursue systems engineering patterns. It may also follow that architectural patterns may help control the complexity of architectures by standardizing on accepted and practiced patterns.

Who would benefit from systems architecting patterns? Systems architecture is typically conducted by experienced systems engineers. Over the course of their careers, they have accumulated the knowledge that is carried forward from design to design. Recognizing patterns as they emerge over time requires the same type of experience. The same holds true in documenting patterns—it takes an experienced systems architect to know what can be abstracted, and how to abstract that information in a meaningful way so that it can be reused. Once documented, as the software community found, patterns provide a common communication medium between engineers and architects. However, there is another consumer of systems architecting patterns, which could yield higher return. If a bright systems engineer with some experience under the belt aspires to become a systems architect, patterns may be of use in their further education. A systems architect, mentoring the aspiring architect would be able to leverage the experience captured in the pattern, or pattern language, as a learning tool for the architect being mentored, providing a launching pad for the next architecture.

5.2. Application of the Pattern Concepts to Systems Architectures—A Survey

Since a literature search on the use of patterns in the engineering disciplines indicates that there may be potential benefits supporting the use of patterns at the system architecture and design level, one must then ask the next question: “Does the systems architect need a

<table>
<thead>
<tr>
<th>Form Heading</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pattern Name:</td>
<td>The name of the pattern should be descriptive to enable the pattern user to understand the usage.</td>
</tr>
<tr>
<td>Aliases:</td>
<td>Other names by which the pattern may be known</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Keywords which assist in locating appropriate patterns in a repository</td>
</tr>
<tr>
<td>Problem Context:</td>
<td>Brief discussion of the types of situations in which the problem may occur - it should be broad enough to allow for any number of situations in which the problem may arise</td>
</tr>
<tr>
<td>Problem Description:</td>
<td>What is the problem this pattern can be used to solve?</td>
</tr>
<tr>
<td>Forces:</td>
<td>What challenges exist in the problem being addressed by the pattern, and the problems in applying the pattern? May also include constraints the pattern may impose if used. May describe the pattern from multiple views</td>
</tr>
<tr>
<td>Pattern Solution:</td>
<td>Discussion on how the pattern solves the problem being addressed.</td>
</tr>
<tr>
<td>Diagrams:</td>
<td>This can be one or more diagrams necessary to represent the pattern. This can be any notational method desired.</td>
</tr>
<tr>
<td>Interfaces:</td>
<td>Discussion of the critical interfaces or information flows necessary in implementing the pattern - what parameters of the interface can change and which ones can not. What are the interface dependencies, if any?</td>
</tr>
<tr>
<td>Resulting Context:</td>
<td>What are the unaddressed issues remaining when the pattern is applied/used.</td>
</tr>
<tr>
<td>Example:</td>
<td>An example of how the pattern may be applied. Usually in the form of a diagram or model</td>
</tr>
<tr>
<td>Pattern Rationale:</td>
<td>Why the pattern works</td>
</tr>
<tr>
<td>Known Uses:</td>
<td>Where else is the pattern being used in other places or applications?</td>
</tr>
<tr>
<td>Related Patterns:</td>
<td>Other patterns that may work in conjunction or in association with this pattern</td>
</tr>
<tr>
<td>References:</td>
<td>Other information that may be useful in understanding or applying the pattern</td>
</tr>
<tr>
<td>Author(s):</td>
<td>Who documented the pattern? May add a date if desired.</td>
</tr>
</tbody>
</table>

Table I. Summary Data—Necessary Pattern Documentation Sections

<table>
<thead>
<tr>
<th>Sections Necessary to Document Systems Patterns</th>
<th>Rank 1-3</th>
<th>Rank 4-7</th>
<th>Rank &gt;7</th>
<th>Total Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rationale</td>
<td>17</td>
<td>3</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>Example</td>
<td>12</td>
<td>7</td>
<td>2</td>
<td>21</td>
</tr>
<tr>
<td>Known uses</td>
<td>12</td>
<td>11</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>Interfaces</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>14</td>
</tr>
<tr>
<td>Sketch</td>
<td>8</td>
<td>6</td>
<td>1</td>
<td>15</td>
</tr>
<tr>
<td>Resulting context</td>
<td>8</td>
<td>7</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>Related Patterns</td>
<td>7</td>
<td>10</td>
<td>3</td>
<td>20</td>
</tr>
<tr>
<td>Forces</td>
<td>6</td>
<td>8</td>
<td>2</td>
<td>16</td>
</tr>
<tr>
<td>References</td>
<td>4</td>
<td>12</td>
<td>7</td>
<td>23</td>
</tr>
<tr>
<td>Author(s)</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>11</td>
</tr>
<tr>
<td>Date Documented</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>12</td>
</tr>
<tr>
<td>Email</td>
<td>1</td>
<td>0</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Keywords</td>
<td>1</td>
<td>5</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Alias</td>
<td>1</td>
<td>8</td>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>
different solution for documenting patterns than that used by other engineers?” This topic was discussed during a colloquium conducted at Stevens Institute of Technology [Cloutier, 2005a]. Two issues with regard to patterns for systems architectures were highlighted at this colloquium. The first issue is abstraction. The architecture of a system (an enterprise system or a complex system) requires a higher level of abstraction than necessary for the software that may be a part of the system. Additionally, many systems include a combination of hardware, software, and other resources that 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 software and nonsoftware parts (if they exist) in the pattern description. This notion of interfaces has not been explicitly addressed in past software pattern discussions.

Survey data demonstrate 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 [Cloutier and Verma, 2006; Cloutier, 2006]. Summary data derived from the survey are presented in Table I. Based on the analysis of those data, the sections considered important in the documentation of architecture patterns are listed in Table II.

The Example heading is meant to provide a diagram or model of the pattern. The diagrammatic representation of patterns has matured over time. It is easy to imagine this is due to the increasing complexity and

---

Table III. Perform C2 Pattern (Page 1 of 3)

<table>
<thead>
<tr>
<th>Pattern Name:</th>
<th>Perform C2, Perform Command and Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases:</td>
<td>None known</td>
</tr>
<tr>
<td>Keywords:</td>
<td>Command and Control, Plan, Detect, Control, Act, C2</td>
</tr>
<tr>
<td>Problem Context:</td>
<td>Many systems are designed to perform detailed planning, management and execution of a situation. Command and control is the process of planning, managing and performing a complex set of tasks. This pattern does not address “Prepare” precondition (though one might argue that prepare and plan go together) nor “Assess” post condition.</td>
</tr>
<tr>
<td>Problem Description:</td>
<td>In command and control (C2), the situation is typically managed in identifiable phases. The situation may move back and forth between the stages. Those stages are Plan/Detect/Control/Act.</td>
</tr>
<tr>
<td>Forces:</td>
<td>Terminology may vary from one domain to the next, and should be adapted in the application of the pattern.</td>
</tr>
<tr>
<td>Pattern Solution:</td>
<td>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.</td>
</tr>
<tr>
<td>Model:</td>
<td>See next page.</td>
</tr>
<tr>
<td>Interfaces:</td>
<td>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. Adapting this pattern may mean that some flows are deleted because they are not required in a particular domain. However, some thought should be given before deletion – for instance, a sensor in the defense business may be radar, while in police work, it may be a person, or even a global positioning capability in a cell phone.</td>
</tr>
<tr>
<td>Resulting Context:</td>
<td>Further work will be required to define the tasks to be performed within each stage, and the allocation of tasks to systems, hardware, software or people.</td>
</tr>
<tr>
<td>Example:</td>
<td>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 a situation or “bad guy”, an identification and controlling or managing of the information, and a required action to perform the mission. The pattern may even be extended to motor vehicle fleet operations.</td>
</tr>
<tr>
<td>Known Uses:</td>
<td>Command and Control applications in various domains</td>
</tr>
<tr>
<td>Related patterns:</td>
<td>OODA (Observe, Orient, Decide, Act)</td>
</tr>
<tr>
<td></td>
<td>MAPE (Monitor, Assess, Plan, Execute).</td>
</tr>
<tr>
<td>References:</td>
<td>1. Command, Control, and Communications Systems Engineering, Walter R. Beem</td>
</tr>
<tr>
<td></td>
<td>2. Science of Command and Control: Part II, Coping with Complexity</td>
</tr>
<tr>
<td></td>
<td>3. MCDP 6 Marine Corps Command and Control Handbook</td>
</tr>
<tr>
<td>Pattern Rationale:</td>
<td>This is a time-tested doctrine used by the military that may be applicable to other domains.</td>
</tr>
<tr>
<td>Author(s):</td>
<td>Harry Johnson Ph.D., Ken Hartnett, Satya Moorthy, Robert Cloutier. 2006.</td>
</tr>
</tbody>
</table>
abstract nature of the information contained in a systems architecture pattern. Alexander’s diagrams were simply hand drawn sketches as seen in the patterns represented in Figures 1 and 2 (House for a Couple and House for a Small Family). They represent simple concepts, which did not require complicated diagrams. Gamma et al. [1995] expanded the sketch to class diagrams and interaction diagrams. This convention was carried forward by Buschmann et al. [1996], who used a combination of text, tables, class diagrams, and sequence diagrams to represent patterns such as the Blackboard pattern. As patterns are used to represent higher levels of complexity and abstraction, the “sketch” approach utilized by Alexander may become inadequate. For instance, the concept of command and control is considerably more complex and abstract than the notion of a gated garden. A simple sketch may be sufficient to convey the underlying architecture of a gated garden, while architecture of the underlying concepts of command and control may indeed require a more complex notation—e.g., IDEF or UML. Finally, patterns are methodology agnostic, so if the pattern is best represented using an SA diagram, or an OO diagram, use what best captures the pattern.

5.3. Example: The C2 Pattern

In this section, the Perform C2 pattern [Cloutier and Verma, 2006] will be discussed. The pattern documents the concept of operations for command and control (C2) using the Plan, Detect, Control, and Act paradigm. Though this pattern is normally associated with military systems, it has broader applications to other domains that involve emergency response. It could also be applied to a command and control system for transportation systems like the railroad or a trucking fleet with
little change. The Perform C2 pattern is documented in the following example (Table III and Figs. 3 and 4).

5.3.1. Alternate C2 Pattern Diagrams

Different graphical notations are usually associated with the engineering methodology that has been chosen. The most common methodologies are functional decomposition, represented with diagrams such as IDEF0, Functional Flow Block Diagrams, N2, etc. [Buede, 2005; Sage and Palmer, 1990]. When using an object-oriented approach like the Object Oriented Systems Engineering Methodology (OOSEM) [Friedenthal and Meilich, 2003], then the notation is more likely to be the Unified Modeling Language (UML). Based on the graphical notation, the diagrams presented in the C2 pattern may look like those shown in Figures 5 and 6. (In Fig. 6, it should be noted that the I/O can be represented more explicitly on the activity diagrams via object nodes/pins. Figure 6 shows the I/O simply as an annotation.)

5.4. The Perform C2 Pattern Language

As discussed in the Introduction, a pattern language is a network of patterns that are complementary, and may work together to form a larger pattern. The C2 pattern, as a whole, relies on the collection of smaller patterns—Plan, Detect, Control, and Act. Therefore, one might refer to the C2 pattern as a pattern language. Each of these four patterns could be used independently to architect the Concept of Operations (CONOPS) of another system. For instance, the Plan pattern shown in Figure 7 could be used by a marketing organization developing a new software application to manage new product launches.
Figure 5. Perform C2 pattern (alternate page 2 of 3).

Figure 6. Perform C2 pattern (alternate page 3 of 3).
5.5. Additional Thoughts on the C2 Pattern

There are a number of items that can be summarized from the documentation of the C2 pattern. First, the Plan/Detect/Control/Engage workflow for command and control (C2) can be documented as a pattern for reuse. Sufficient detail can be captured in the pattern to enable an architect less familiar with the process to begin architecting a solution to a new instance of the C2 problem. However, as one “drills down” in 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.

Anecdotal evidence from SME interviews that have applied this pattern in the past indicate that this type of pattern is extremely helpful in guiding stakeholder interviews, acting as a reminder to the systems engineer of questions that must be addressed. One of the creators of this pattern noted that last time the pattern was applied to a project; he arrived at the same design point of the previous implementation months earlier.

6. APPLYING AND MANAGING SYSTEMS ARCHITECTING PATTERNS

This section will address the actual application of patterns to systems architecting. A brief discussion on possible strategies for managing patterns is also included.

6.1. Applying Patterns in Systems Architecting

To demonstrate how a systems architect would use the Perform C2 pattern while architecting a system, a scenario will be used. The discussion will be generalized for simplicity, but the concepts will apply to most scenarios. The example should not be considered as a full treatment of what a systems architect would do to architect the system.

Consider the situation where a mid-sized town has embarked on an effort to upgrade the emergency response capability. They have decided to design an emergency response center (ERC). All communications associated with an emergency will flow in and out of the ERC. During an emergency, the town Mayor, Chief of Police, Fire Chief, and Emergency Response Director will report to the ERC. From the ERC, the team will be able to plan the response, monitor, and direct actions of the emergency response personnel. The system also has the requirement to integrate the police, fire, and EMT efforts, as well as assistance coming from neighboring towns, or even other government agencies. An engineering firm has been contracted to architect this system.

Upon interviewing the city officials and subject matter experts (SMEs), the systems architect believes the Perform C2 pattern in the firm’s pattern repository is a good starting point to model the concept of operations (CONOPS). Looking at the context diagram on page 1 of the pattern, many of the same terms are found in the

Figure 7. Part of a pattern language—a plan system pattern.
keywords section, and the problem description seems to be similar. However, many terms were never mentioned in the interviews—terms like track, sensor availability, etc.—not an exact fit, but a possible start. The consultant decides to use the pattern to help develop the CONOPS.

Customizations when implementing any pattern are normal, and in fact, expected. Patterns are not meant to be an exact fit. In the C2 pattern the term “Tracks/Targets of Interest” may represent assets—fire trucks, squad cars, and EMT units—all of which need to be tracked during an emergency response. Knowing the whereabouts of these assets allows the command center to redeploy as necessary, or when the situation changes. Sensor availability may be a citizen calling in a report before an official response team gets there. Sensor data could also be a latitude and longitude from a GPS receiver from a citizen to help exactly pinpoint a location if it is for a more rural location. Reports are critical items on this pattern. Looking at page 2 of the pattern, a number of items are still necessary without change: external data and environmental data are still necessary, as is a strategy, external guidance (county or state laws) and doctrine (standard responses already documented), and resources. The outputs need a bit of adjustment, however—BDA (battle damage assessment) may not be applicable, so the output is changed to be situation assessment. Mission assessment sounds like a duplicate to situation assessment, so it might be a candidate for deletion. Coordination data are still necessary. Orders, plans, reports, resource assignments, situational data, tasking, and request for information are all necessary. Other items may be renamed or deleted as applicable.

Once the context diagram (A-0) is correct, the diagram on page 3 of the pattern can be incorporated into the design by updating the terminology to be consistent with the context diagram, removing the items that were deleted, and adding new items that may not exist on the pattern. At this point, the pattern has been applied. The consultant has the beginnings of a functional architecture (A0 diagram) for the city’s ERC.

6.2. Managing Systems Architecture Patterns

One question that is asked frequently regarding patterns during the course of this research is the concern for being able to locate or select the correct pattern for the problem being addressed. Engineering patterns should not be thought of as a silver bullet. Although the pattern provides a basic structure of the solution to a particular problem, it does not provide a solution that can be used “as-is” [Buschmann et al., 1996]. Sound engineering is still required. The same is true when it comes to selecting a pattern—sound engineering judgment is required to make the correct pattern choice—or choose not to use a pattern in a particular case. However, like managing reusable software, there are things that must be done to foster reuse of patterns.

A review of the literature on software reuse libraries suggests the following lessons learned, equally applicable to managing and using system architecture patterns:

A. Build a “critical mass” of reusable components quickly. Then advertise the availability of the repository. If engineers go there to find something before a decent number of patterns are available, they will quit using this source.
B. Identify reusable components that should be documented for early inclusion in the library.
C. Reusable components should be easy to use, not promise too much, and be very reliable.
D. Provide a Reuse Tool from the start.
E. Assign a Reuse Librarian from the start.

The methodology described here provides the means to capture the content of the “critical mass” for a pattern reuse repository. It details an approach for documenting patterns, based on input from a body of experienced systems architects and systems engineers. The big challenge is for systems architects and engineers to recognize the value of patterns, to begin identifying patterns, and then to document them using this methodology. There is an opportunity for systems engineering academic institutions, and/or INCOSE—to collaborate on a systems engineering and architecting patterns repository similar to the REPARSE repository discussed earlier in this paper.

7. DIRECTIONS FOR FUTURE RESEARCH

The use of patterns in systems architecting intersects with a number of other key emerging technologies. A few of these technologies include the Object Management Group (OMG) model driven architecture (MDA) approach, product line management (PLM).

The advent of SysML should encourage the adoption of modeling the system architecture using formal modeling tools that support the SysML standard. These models may then be “executed” to begin early verification and validation. Part of the MDA approach includes developing a computational independent model (CIM) and platform independent model (PIM). These notions are described today in terms of software and software architecture. If they are described in terms of systems architecture, what use might systems architecture patterns serve in describing the CIM or PIM? Another
emerging concept in systems architecting is the application of product line management. How can one manage the product line of very complex system architecture on the magnitude of a military fighter or warship? In commercial terms, how might one apply product line management concepts to the global positioning system? Moreover, do patterns help in this architecture development and management? Intuitively, it seems there is a point of intersection with these technologies, and the author intends to continue research in these directions.

8. CONCLUSIONS

The research in this paper represents a methodology to address the specific needs of systems architects and systems engineers with regard to documenting and applying patterns. Looking at the pattern form documented in this paper, one could conclude that simply adopting a complete software pattern form may have been sufficient. However, that approach would have missed the importance of capturing information specifically related to interfaces. This may seem self-evident to experienced systems engineers. However, this paper presents research, based on a broad base of international experience, to confirm that observation.

The Perform C2 pattern demonstrates sufficient detail can be captured and documented to enable the reuse of the pattern in future projects. It does not provide all the answers, but it can provide a significant jump-start to architects and engineers that may not be as familiar with the domain captured in the pattern. This methodology enables the expert knowledge and advice of experienced systems engineers to be transferred to less experienced architects and systems engineers.

The documentation and application of patterns promises the potential to improve the efficiency and productivity of virtually all engineering disciplines using patterns. There is little evidence beyond anecdotal evidence that bears out this potential; however, there is sufficient evidence throughout engineering disciplines to suggest that systems engineering may benefit by using patterns. As an extension to this, the ability to document patterns in such a way that they can be applied at the system architecture/systems engineering level may also reduce the documentation costs and complexity. The major difference in documenting a pattern for use in systems architecture or design is the addition of interfaces within the pattern. What was not addressed is the parameterization of pattern parts—is there a way to parameterize the parts of an architecture pattern to help at the interfaces? Each of these unanswered questions provides future research opportunities.

Patterns offer the potential to manage and control the complexity of a system by standardizing designs on well-known and practiced patterns, where appropriate through the creation of a common lexicon to be used by systems engineers. Evidence that this lexicon is a reality is the understanding that today’s systems engineers have with a layers architecture, or a client server architecture. Both were first documented in the mid 1990s as software patterns. While this paper represents early into the application of patterns to systems architecture, there are many aspects that need further investigation. One such topic to be explored next year is the applicability of systems patterns to deeper, conceptual elements of systems architecture. Alexander’s [2003] most recent work, representing decades of pattern application to civil architecture, indicates that he still continues to struggle with what can and cannot be conveyed with patterns, and believes there is much to discover.

Finally, if patterns are to be useful to the systems engineering community, there will have to be an open repository for pattern submission and reuse. This repository would become a vehicle to build a maturing source of patterns that can be leveraged for similar realized benefits already gained in other engineering disciplines. Patterns are not cookie cutters. In some settings, patterns may be reused, and not have any two designs which are identical. Patterns should not remain stagnant. They should be improved upon. “We may then gradually improve these patterns which we share, by testing them against experience…” [Alexander, 1979]. Patterns and pattern languages are not created from a blank page; they are mined [Hammer and Kocan, 2004]. Sanz and Zalewski [2003] summarized the use of patterns by stating: “The pattern concept is not easy to understand, however, the best way for the reader to fully grasp its potential is to practice it. Select a pattern schema adequate for your field of work and try using patterns to document the simple design elements that proved useful in the past and that you are typically employing in your systems.”

ACKNOWLEDGMENTS

The authors acknowledge support from the Lockheed Martin Corporation, and specifically Sandy Friedenthal, Jeff Carter, and Kathy Tennon. Osmo Vikman, Sari Leppänen, Pasi Rinne-Rahkan, Markku Turunen, and Jukka Honkola from Nokia Technology Platforms and Nokia Research Center contributed significantly to this research. Finally, the support provided by Vitech
Corporation and Telelogic for the use of their modeling tools is appreciated.

REFERENCES

J. Coplien, Idioms and patterns as architectural literature, IEEE Software (Special Issue on Objects, Patterns, and Architectures) (January 1997).
E. Gamma, R. Helm, J. Johnson, and J. Vlissides, Design patterns: Elements of reusable object oriented software, Addison-Wesley, Reading, MA, 1995.

Robert Cloutier (Rob) received his Ph.D. in Systems Engineering from Stevens Institute of Technology. He has over 20 years experience in systems engineering, software engineering, and project management in both commercial and defense industries. He is a Research Associate Professor at Stevens, where his research interests include Systems Engineering Patterns and modeling complex systems with UML/SysML, Reference Architectures, and agent based technology applied to systems engineering. Rob also has an M.B.A. from Eastern College, and a B.S. from the United States Naval Academy. He is an Adjunct Professor for Eastern University and chairs the Rowan University Electrical and Computer Engineering Department Industry Advisory Board. Rob belongs to the International Council on Systems Engineering (INCOSE), and is a member of the Technical Leadership Team. He is also an active member of the Association of Enterprise Architects, and is a member of IEEE.
Dinesh Verma 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 Advisor to the Director of the Embedded Systems Institute in Eindhoven, The Netherlands. Prior to this role, he served as Technical Director at Lockheed Martin Undersea Systems, in Manassas, VA, 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, including 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.