Towards a Systems Engineering Pattern Language: Applying *i** to Model Requirements-Architecture Patterns

P. Pavan, N.A.M. Maiden & X. Zhu

Centre for HCI Design, City University
Northampton Square, London UK

Abstract

This paper reports the results of exploratory research to develop a pilot pattern language for systems engineers at BAE SYSTEMS. The pattern language was designed to encapsulate knowledge about possible trade-offs made by systems engineers about architecture designs that satisfied different system requirements for submarine manoeuvring systems. Our intention is that this knowledge can be reused in future systems engineering processes using our ART-SCENE environment. Knowledge about requirements, design alternatives and the complex trade-off space was elicited from systems engineers. To model this knowledge we applied the i formalism to represent the design space and design trade-offs, and to communicate the resulting patterns back to the engineers for validation and improvement. The research was a success, in that we produced a pattern language of 4 key patterns and their interactions for a submarine manoeuvring system, all using the i* formalism. The paper ends with a review of this research and how we plan to exploit the language to inform scenario-driven trade-offs between requirements satisfaction and architecture choice using the ART-SCENE environment.*

1. Patterns of Patterns – Linking Requirements and Architectures

There is increasing recognition of the need for systems engineers to link system requirements and architecture designs. Considerable research is being undertaken into a range of topics – from tracing architectural decisions to requirements to relating architectural patterns to requirements patterns (Jackson 1995) and formal foundations of the requirements-architecture relationship (Hall et al. 2002). However this research tends not to investigate the systems engineering processes that its results are intended to support. Rather we argue that requirements-architecture research must be based on sound process models of concurrent requirements-architecture engineering. Using one such process model, this paper presents a novel pattern-based approach for exploring requirements-architecture trade-offs, and introduces an innovative pattern language that underpins such trade-offs.

Our ART-SCENE (Analysing Requirements Trade-offs – Scenario Evaluations) approach advocates a process in which systems engineers and stakeholders concurrently:

1. Generate and walk through system-level scenarios using our established CREWS-SAVRE process and software tools (Sutcliffe et al. 1998);
2. Acquire stakeholder requirements using the ACRE framework (Maiden & Rugg 1996) and model them using the *i** formalism (Chung et al. 2000) with our REDEPEND tool (Maiden et al. 2002);
3. Model candidate system architectures using object-oriented modelling techniques supported by the AUTOFOCUS software tool (Huber et al. 1998);
4. Trade-off the satisfaction of different requirements by different architecture designs using scenarios to link requirements and architectures, then to simulate system and agent behaviours to compute their outcomes (Zhu et al. 2003).

A simple overview of the process is shown in Figure 1. Systematic scenario walkthroughs lead to the acquisition of more complete stakeholder requirements (Maiden et al. 2003). Candidate system architectures lead to the rejection of stakeholder requirements that are not viable. Simulations of models of candidate system architectures using the scenarios compute the emergent properties of the system that can then be tested for compliance with the measurable fit criteria of stakeholder requirements (Robertson & Robertson 1999). Based on these processes we are developing a suite of integrated software tools that, we hope, will provide systems engineers with an effective plug-and-play environment for exploring requirements-architecture trade-offs in the presence of system scenarios. The ART-SCENE software tools and techniques developed to implement these processes are described elsewhere (Zhu et al. 2003, Maiden et al. 2002).

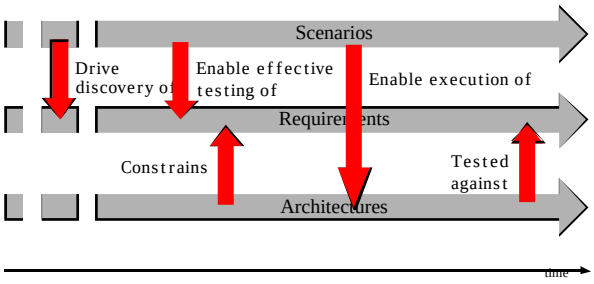

Figure 1. Simplified ART-SCENE process from slide

In this paper we report research that led to one of ART-SCENE's features to support this process – the use of systems engineering patterns to explore requirement-architecture trade-offs in the presence of scenarios. The paper describes how we overcame 3 challenges to produce a simple pattern language produced in collaboration with system engineers at BAE SYSTEMS as part of the UK EPSRC-funded SIMP project:

1. How to elicit a pattern language that could be applied to inform the essential task of making requirement-architecture trade-offs;
2. How to model the patterns using a formalism that was sufficiently expressive and computational;
3. How to implement the resulting pattern language in the ART-SCENE environment to inform requirement-architecture trade-offs.

The remainder of this paper is in 4 sections. Section 2 reports previous research in systems patterns undertaken by the authors which shapes the direction of the reported research. Section 3 describes how we elicited the patterns and applied the *i** formalism to model them. Section 4 present a pattern in detail and reviews the success of the elicitation exercise. The last section reports planned future use of such pattern languages in the ART-SCENE environment, as well as implications for future such pattern modeling in system and software engineering.

2. Researching Requirements Engineering Patterns: Lessons from the Trenches

There has been considerable recent interest in relating classes of problem domains (i.e. requirements) to classes of software solutions (i.e. architectures and designs). Jackson (1995), for example, advocates problem frames, a generalisation of a class of problem consisting of the principal problem parts and a solution task. Hall et al. (2002) extend these problem frames by linking them to simple system functions. Elsewhere Konrad & Cheng (2002) present system specification patterns using an object-oriented specification. Earlier work on requirements clichés (Reubenstein & Waters 1991), patterns (Coad et al. 1995, Buschmann et al. 1996) and generalised application frames (Constantopoulos 1991) are all examples of the research undertaken previously in this area. However, our experience in research area suggests major pitfalls await researchers who fail to learn from these past experiences.

The ESPRIT III 6353 'NATURE' basic research action, in which the authors were partners, undertook between 1992 and 1995 one of most comprehensive classifications of problem domains and software architecture styles to date (Sutcliffe & Maiden 1998). The NATURE approach argued that most requirements engineering problem domains are instances of a tractable set of object system models. Each model contains general features shared by all instances of that problem domain. For instance, one model contains general features of all resource hiring problem domains, examples of which are lending libraries, car rental and video hiring. Another contains general features of all object sensing problem domains. NATURE produced the first extensive categorisation of requirements engineering problem domains and derived a set of over 200 object system models (Sutcliffe & Maiden 1998) from domain analysis, case studies and textbooks.

Object system models were defined in a hierarchical class structure. The 13 highest-level object system models define the fundamental state transitions, sequences of transitions, states and objects in categories of problem domains, and indicate the breadth of the models specified. These models (with a prototypical example of each) are *resource returning* (e.g. car rental), resource supplying (e.g. order purchasing), *resource usage* (e.g. sales orders), *item composition* (e.g. goods manufacturing), *item decomposition* (e.g. unpacking deliveries), *resource allocation* (e.g. production planning), *logistics* (e.g. complex production scheduling), *object sensing* (e.g. aircraft detection), *object messaging* (e.g. electronic mail), *agent-object control* (e.g. air traffic control), *domain simulation* (e.g. cockpit simulation), *workpiece manipulation* (e.g. text processing) and *object reading* (e.g. a town's computerised information point).

Specialisation of these high-level object system models is achieved by adding different dimensions, in the form of fact types, at different levels in the hierarchies. For example, the specialisation of the object system model for object sensing generates a large number of lower-level, more detailed hierarchical object system models. The top-level model, which describes the sensing of an object by a sensor agent, is specialised at level-1 according to whether the sensor is sensing the physical location (e.g. position) or internal state (e.g. temperature) of the object, and whether there are one or numerous objects to sense. NATURE specialises each level-1 model further using:

- Different goal states (detect forbidden state, warn if forbidden state arises, forbid state to arise, etc);

- Different events and stative conditions on state transitions (e.g. a monitoring agent blocks an object from changing to a forbidden state);
- Different object and agent types (e.g. physical, conceptual and financial).

Such specialisation gave rise to over 30 level-4 models that are sub-classes of the original object sensing model alone.

We validated NATURE's object system models against natural mental categories elicited using card sorts with experienced software engineers. Results of this empirical validation led to some revision of the structure and contents of several models and how these models might be retrieved and used (Maiden & Hare 1998). Furthermore, to relate these classes of problem domain to software solutions, we developed an orthogonal classification of information system models that we linked to object system models to represent candidate information system solutions for different problem domain classes (Sutcliffe & Maiden 1998). To exploit the library of object system models we developed computational models of analogical reasoning (Maiden & Sutcliffe 1996a, 1996b) to retrieve object and information system models that matched a new application to enable reuse of knowledge about the problem domain and possible information system solutions to it. This reuse-driven approach to requirements engineering and high-level design was validated using several application case studies .

So, what conclusions did we draw from this extensive 5-year programme of research? Although the results of the basic research provides important insights into the nature of abstraction and classifications of problem domains, the more direct benefits to systems engineering were limited. Lessons learned included:

1. The object system models encapsulate problem domain knowledge that is often already known and accessible to systems engineers;
2. NATURE's classification of problem domains was too fine-grain for cost-effective reuse – most applications were an aggregation of instances of a large number of object system model classes, which made model retrieval and instantiation difficult;
3. Systems engineers gained little from directly reusing small fragments of knowledge from the object and information system models – indeed the emergence of large business reference models implemented in successful ERP solutions such as SAP R/3 (Curran & Ladd 1998) suggests that reuse of large, domain-specific models tend to be more effective;
4. Linking the object and information system models did not provide systems engineers with useful knowledge with which to make requirements-driven architectural decisions. The information system models, by their definition, described functions, classes and structures rather than the non-functional requirements and quality attributes essential to such decision-making (e.g. Franch & Carvallo 2003). The models did not capture the required richness and context or different design alternatives and their comparison.

These experiences directed us to a different approach to modeling patterns in systems engineering – one that captures the essence of architecture design alternatives in terms of how they satisfy a number of related system requirements. As an input to the ART-SCENE environment we sought to develop a pilot pattern language to do just that to inform requirements-architecture trade-offs. The remainder of this paper reports the results of research motivated to determine such patterns.

3. Eliciting Systems Engineering Patterns in ART-SCENE

Our research uses Alexander's (1979) original definition of a pattern as a solution to a problem in a context of use. Alexander defines a *pattern* as a three-part construct:

1. The **context** - conditions under which the pattern holds;
2. A **system of forces** - the 'problem' or 'goal' that the solution solves;
3. The **solution** – a configuration that balances the system of forces and solves the problem.

This definition contrasts markedly with the nature of most software and systems engineering patterns reported earlier. The patterns, from requirements cliches to problem frames, all use a weaker definition of the problem or goal without an explicit description of a system of forces that describes it. Likewise most patterns describe a single reusable solution without explaining how the configuration implemented in the solution satisfies the goal or solves the problem. One exception is reported in Gross & Yu (2001) who highlighted the need for a modeling approach that supports how business goals relate to the architectural decision-making process, and how changing business goals give rise to alternative architectural choices and solution structures. They also showed how the need to describe organisational stakeholders, their goals and how these are affected by alternative choices during the design process using agents and goals. Agents were used to describe architectural distribution of capabilities, while goals were used as a focal point for expressing where within architectural structures further design choices needed to be made.

So how can we apply Alexander's pattern definition to a modern systems engineering process? First of all we need to map Alexander's 3 parts of the pattern to systems engineering models and artefacts. We assert that:

- The **system of forces** represents a set of interconnected system requirements that a designed architecture configuration must satisfy:
- The **solution** represents the architecture design and to what degree that design alternative satisfies each system requirement;
- The **context** represents the conditions under which the pattern holds – that is the project or problem environment in which the solution applies. In ART-SCENE we equate the problem environment to one or more scenarios in which the architecture design must satisfy the system requirements.

Therefore the systems engineering patterns in ART-SCENE encapsulate knowledge about important design decisions that sought to balance the satisfaction of competing requirements in different scenarios for a previous but relevant system. To elicit and model these patterns we designed and applied a rigorous method described in the next section.

3.1. Pattern Elicitation and Modeling Method

We developed the pilot pattern language with BAE SYSTEMS, one of our partners in the EPSRC-funded SIMP project. Our objectives were to model patterns in a domain that was complex but could be understood by the academic researchers, did not require high-level security access, and could be scoped in order to provide results within the time frame of the exercise. The result was a decision to develop a pattern language for submarine manoeuvring systems – that is the systems that enable a naval submarine to steer when under water.

We elicited pattern knowledge from BAE SYSTEMS engineers in 3 phases:

1. Discover and elaborate key design decisions made on previous projects;
2. Model and validate the context, solution and system of forces for each pattern;
3. Elicit, model and validate the key relationships between the patterns established in the first 3 phases to produce the first-cut pattern language.

Each elicitation session took place with 2 systems engineers with shared engineering experience of the submarine manoeuvring system. Throughout each session we encouraged the systems engineers to converse with each other. This technique, known as constructive interaction (Miyake 1986), overcomes the unnatural aspects other elicitation techniques and provided supplementary data about the patterns at each phase.

In the first phase we combined brainstorming with semi-structured interviews to discover and prioritise previous design decisions made about manoeuvring systems. We used the interview structure to elicit data about different candidate architecture designs, why each was chosen or rejected, and conditions for its use. All data was recorded on flipchart sheets as informal sketches and written notes.

In the second phase we used the data to describe each pattern with the following attributes:

Name: A unique and meaningful pattern name;
Authors: The main contributors to the pattern;
Problem: The main trade-offs to be made and the different forces to be balanced to achieve an acceptable solution;
Principle: The principle behind the pattern;
Context: The pre-conditions under which the problem and its solution seem to recur, and for which the solution is desirable;
Forces: A definition of the relevant *forces*;
Solution: Descriptions of architecture solution that can be reused;
Rationale: A justification of the solution and the pattern as a whole in terms of how it resolves its forces to be in line with the desired outcome;
Known Uses: Known occurrences of the pattern and its application within existing systems;
Models: The *i** models that describe the pattern;
Further questions: Questions that need answering in order to further refine the pattern.

One innovation was to produce *i** models (Chung et al. 2000) for each pattern. Inspired by the earlier use of *i** to model requirement-architecture patterns (Gross & Yu 2001), we chose the *i** formalism to model our pattern language for 3 reasons:

1. *i** SD (Strategic Dependency) models allowed us to model each pattern as a network of dependency relationships among actors characteristic of large socio-technical systems – the types of system found in submarine design;
2. *i** SR (Strategic Rationale) models allowed us to model how different candidate **solutions**, represented as tasks in *i**, satisfied different actor goals and soft goals using *i** means-ends links – Alexander's **systems of forces**;
3. *i** contributes-to soft goal links in the SR models allowed us to represent complex trade-offs between requirements that occur when making choices about one solution over another – the **systems of forces** again.

Our aim was to produce one SD model and one SR model to represent each pattern. Other researchers have recogised the potential benefits of providing graphical representations of pattern solution spaces. Thomas (2001) claims that "providing people with a variety of potential representations and some process to encourage the exploration of alternatives… could probably improve performance significantly". The use of *i** models in our pattern language enabled us to explore whether such benefits accrue empirically.

In the third phase we combined semi-structured interviews with direct SD and SR modeling to model dependencies between actors identified in the 4 modeled patterns and contributes-to soft goal links

between important soft goals in these patterns. Such modeling provided the associations between the patterns to form a first-cut pattern language.

4. The Submarine Manoeuvring Pattern Language

The resulting submarine manoeuvring pattern language was elicited from 3 BAE SYSTEMS engineers working as pairs during 5 sessions over a two-and-a-half month period. Each session lasted approximately 2 hours and took place at BAE SYSTEMS premises.

The pattern language consisted of 4 principal patterns linked using additional *i** SD and SR models. The patterns were:

1. The Manoeuvring-Noise-Accuracy (MNA) pattern, describing trade-offs between accurate and quiet steering of the submarine;
2. The Manoeuvring-Weight-Distribution (MWD) pattern, describing, describing trade-offs between accurate manoeuvring and maintaining the stability of the submarine;
3. The Manoeuvring-Console-Manning (MCM) pattern, describing trade-offs about the number of operators who control the manoeuvring of the submarine;
4. The Manouevring-Hydroplane-Configuration (MHC) pattern, that describes trade-offs associated with possible configurations of the hydroplanes that steer the submarine.

The full pattern language is described in Maiden & Pavan (2001). In this paper we describe one of these patterns – the Manoeuvring-Noise-Accuracy (MNA) pattern – and the models that link the individual patterns to provide the pattern language. Each is described using it important attributes.

4.1. The Manoeuvring-Noise-Accuracy (MNA) Pattern

Problem: A submarine is required to be both quiet and accurate when manoeuvring. However to be more accurate it must activite the hydroplanes, which leads to more noise.

Context: When manoeuvring, especially in advanced underwater warfare, the submarine needs to avoid detection by alien systems and to navigate with a high level of accuracy. It avoids detection by controlling and regulating the radiated noise. However, there is a trade off between the quietness (low hydroplane activity) and accuracy (high hydroplane activity). This trade off holds true for accurate navigation as high hydroplane activity implies more accurate navigation and low hydroplane activity leads to less accurate navigation as the hydroplane activity is necessary to change the submarine's direction. Similarly, high hydroplane activity implies a high level of accuracy as well as a high level of noise.

Forces: Performance, quietness, accuracy, reliability, safety, technologies and cost.

Solution: Designers are given noise, manoeuvring and performance targets for the manoeuvring system to attain. They do this by exploring dependencies between the key agents involved in the noise-accuracy domain in order to achieve acceptable trade-offs. Historical data about the effectiveness of this design in different scenarios is available for reuse.

SD Model: We modeled 3 actors that influence manoeuvring – submarine, manoeuvring system and hydroplanes. Modeled actor dependencies were:

- The submarine depends on the manoeuvring system for the soft goal of *manoeuvre submarine quietly;*
- The submarine depends on the manoeuvring system for the soft goal of *manoeuvre submarine accurately;*
- The submarine depends on the manoeuvring system for the task of *manoeuvre submarine;*
- The submarine depends on the manoeuvring system for the goal of *navigate the submarine;*
- The manoeuvring system depends on the hydroplanes for the soft goal of *manoeuvre system is quiet;*
- The manoeuvring system depends on the hydroplanes for the soft goal of *manoeuvre system is accurate.*

The resulting SD model is shown in Figure 2.

Figure 2. The SD model for the MNA pattern

SR Model: The SR model models the goal structure of each actor in the SD model. The submarine has two high-level soft goals. The first is *avoid detection by alien systems*, which is achieved by attaining the goal of *control radiated noise* and undertaking the task *regulate radiated noise.* Successfully undertaking the task of *regulate radiated noise* depends on successfully achieving the soft goal *manoeuvre system is accurate* by the manoeuvring system actor. The SR model is shown in Figure 3.

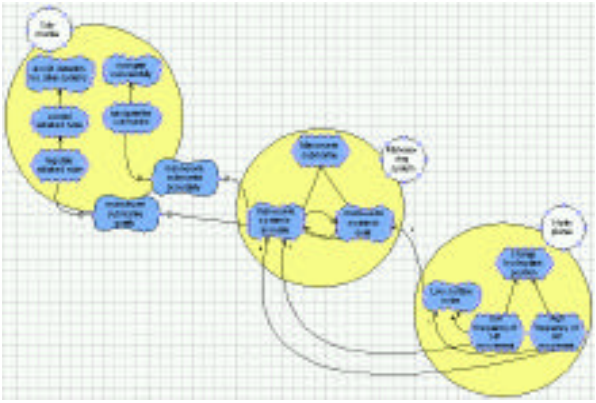

Figure 3. The SD model for the MNA pattern

Another soft goal of the submarine is to *navigate successfully*, achieved in part by achieving the goal *navigate submarine*, which in turn also depends on the soft goal *manoeuvre system is accurate* of the manoeuvring system actor. The *manoeuvring system* actor undertakes the task *manoeuvre submarine*. This task is decomposed into the two soft goals of *manoeuvre system is accurate* and *manoeuvre system is quiet*. Satisfying the soft goal *manoeuvring system is accurate* contributes negatively to the satisfaction of the soft goal of *manoeuvre system is quiet.* Likewise satisfying the soft goal *manoeuvre system quietly* contributes negatively to the soft goal *manoeuvre system is accurate*.

The third actor, *hydroplanes*, undertakes the task *change hydroplane position* which can be achieved by either *low frequency of hydroplane movement* or *high frequency of hydroplane movement*. *Low frequency of hydroplane movement* contributes positively to the soft goal of *low surface noise* while *high frequency of hydroplane movement* contributes negatively to the soft goal of *low surface noise*. In addition the soft goal of *low surface noise* has a positive contribution on the soft goal of *manoeuvre system is accurate* (in agent manoeuvring system). Finally, *low frequency of hydroplane movement* has a negative contribution on the soft goal of *manoeuvre system is accurate* and *high frequency of hydroplane movement* has a positive contribution on the soft goal of *manoeuvre system is accurate* of the manoeuvring system actor.

4.2. Additional Models forming the Language

To link the 4 patterns in the language we elicited 2 additional models from the systems engineers:

- An *agent* model, showing the logical associations between the principal actors in the manoeuvring domain;
- A model that shows the important *contributes-to soft goals links* that hold for all patterns in the language.

The models extend the *i** semantics and syntax. The agent model includes additional semantic associations between agents to describe their logical structure during manoeuvring. Each is described in turn.

The manoeuvring actor model is shown in Figure 4. It describes the aggregation of strategic dependencies between all of the actors from the 4 SD models of the 4 patterns. As such it summarises the dependencies in the pattern language, and adds to them through the definition of other semantic associations between agents stating which actors interact with each other to undertake tasks.

Figure 4. The pattern language actor model

One of the most striking features of the model is the existence of 2 key structures of manoeuvring. The first feature is a structure that states that:

- The manoeuvring system is part of the submarine;
- The auto pilot, hydroplanes and rudders, and trim and compensation tanks are part of the manoeuvring system;
- The console system controls the auto pilot, hydroplanes and trim and compensation tanks.

The second structure is the command-and-control structure within the submarine. The operator interacts with the console system. The operator reports to the supervisor who reports to the commander to manoeuvre the submarine. This structure reveals that

failure of one actor to achieve a goal or undertake a task can lead to serious difficulties to manoeuvre the submarine. The model calls into question the robustness of the design that this implies by this actor structure.

The 4 patterns also identified recurring structures of contributes-to soft goal links. A separate elicitation session was undertaken to determine these structures and to extract them from the specific patterns. The resulting model is shown in Figure 5.

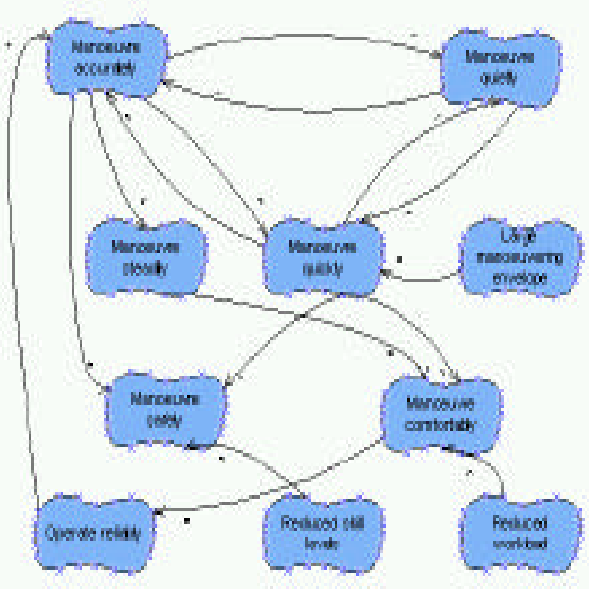

Figure 5. Recurring contributes-to soft goal links in the patterns

5. Conclusions and Future Work

This paper reports results from research to produce a pilot pattern language in systems engineering. We worked with BAE SYSTEMS engineers to elicit, model and validate a pattern language of issues to consider when designing the manoeuvring systems of naval submarines. We adopted a formal knowledge elicitation approach with the systems engineers. The resulting knowledge was modelled using the *i** formalism to show the allocation of capabilities in terms of goals, soft goals and tasks to different actors in the architecture, and the dependencies between the actors and the trade-offs to be made between the satisfaction of competing soft goals. The pattern language was accepted by the engineers as an accurate and useful representation of design alternatives for manoeuvring system. Although developed for a systems engineering problem, the definition of a pattern that we adopted for this research and the elicitation and modelling approach that was successfully applied has important consequences about linking software system requirements and architectures.

One important finding was the usefulness of the *i** formalism for representing patterns about complex requirements and design decisions. The use of *contribute-to soft goal links* in the SR model enabled us to go beyond existing pattern-approaches in software engineering and model not just one solution to a pattern but all of the design alternatives. In our language we chose to use the *i** concept of a task to model each candidate solution. Whilst an effective representation of solutions in business and information system applications, the use of tasks (e.g. steer with low hydroplane movement) is not an ideal representation for complex system architectures, and suggests the need to extend the *i** semantics and syntax in the future. Such an extension will need to take into account the differences in discrete and continuous solution spaces. Whereas some patterns, such as the Manoeuvring-Console-Manning (MCM) pattern had discrete solutions such as *manoeuvre with 1 operator* and *manoeuvre with 2 operators*, others such the reported Manoeuvring-Noise-Accuracy (MNA) pattern has a large space of possible solutions that are modelled mathematically in BAE SYSTEMS.

During the sessions with BAE SYSTEMS engineers we found that the patterns provided the common shared point of reference that was needed to address the issues relating to requirements trade-offs. The text attributes of each pattern provided the necessary background and contextual information while the *i** models provided the engineers with powerful visual depiction of the trade-off envelope. This representation also enabled the researchers and other BAE SYSTEMS participants not as familiar with manoeuvring systems still to participate in the sessions. We believe that this was in part because the *i** semantics provided homogeneity through its powerful, but easy-to-understand processes, semantics and syntax which participants used when referring to the different attributes of the system. This went some way towards creating a more egalitarian discussion arena among the diverse participants.

Furthermore, in the latter sessions, the BAE SYSTEMS engineers became sufficiently adept with the *i** formalism that they would arrive at the elicitation or validation session with SD and SR model sketches already produced, thus saving time and improving communication. This last result was a surprise to us as we had anticipated a large learning curve with the *i** approach. This finding supports other experiences with the *i** approach that it can be learned quickly and applied successfully with stakeholders with some engineering experience. Our decision to use *i** models to express the pattern language itself, that is the associations between individual patterns, also enabled us to explore the claim that pattern languages provide a lingua franca or common language that is

accessible to all the participants in a design process (Erickson, 2000). One possible role for this and other pattern languages in BAE SYSTEMS is to aid collaboration through communication between different stakeholders in order to provide a coherent but flexible framework for problem solving, rather than forcing systems engineers to implement the specified pattern solutions for future designs.

Another interesting side effect from developing the pattern language was that it provided the systems engineers with an opportunity to reflect on their designs and design practice often denied them due to project deadline pressures. Once reflection was recognised as a characteristic of the sessions, engineers were more motivated to participate and share their knowledge with others.

The next stage of this research is to integrate this and other pattern languages within the ART-SCENE environment described at the beginning of this paper. ART-SCENE is designed to trade-off satisfaction of different requirements by different architecture designs using scenarios to link requirements and architectures, then to simulate system and agent behaviours to compute their outcomes (Zhu et al. 2003). The correctness of scenario outcomes upon which we determine an architecture's compliance with system requirements depends upon the accuracy of the model and the simulation. We seek to make ART-SCENE's simulations more dependable by diversifying the sources of information, and in particular by reusing historical data about the performance of a design in previous similar contexts based on pattern languages. We will extend the pattern language to represent architectural designs using object-oriented constructs. We will then evolve NATURE's original computational analogical reasoning mechanisms (Maiden & Sutcliffe 1996) to match candidate patterns' requirements and architecture models to the models of the system under specification to retrieve historical data about that design's performance in previous scenarios, and input this data into scenario simulations within ART-SCENE. We look forward to reporting this next challenge in the next future.

6. Acknowledgements

This research is funded by the UK EPSRC's SII 'SIMP' project. Our gratitude to BAE SYSTEMS who provides the engineering and consent for the studies reported in the paper.

7. References

Alexander C., 1979, 'The Timeless Way of Building', NY: Oxford University Press.

Konrad S. & Cheng B., 2002, 'Requirements Patterns for Embedded Systems', Proceedings 10th International Joint Conference on Requirements Engineering', IEEE Computer Society Press, 127-136.

Chung L., Nixon B., Yu E. and Mylopoulos J., 2000, 'Non-Functional Requirements in Software Engineering', Kluwer Academic Publishers.

Curran T.A. & Ladd A., 1998, "SAP R/3 Business Blueprint", Prentice-Hall.

Franch X. & Carvallo J., 2003, 'Using Quality Models in Software Package Selection', IEEE Software Jan/Feb 2003, 34-41.

Gross D. & Yu E., 2001, 'Evolving System Architecture to Meet Changing Business Goals: An Agent and Goal-Oriented Approach', Proceedings STRAW'2001 Workshop.

Hall J., Jackson M., Laney R., Nuseibeh B. & Rapanotti L., 2002, 'Relating Software Requirements and Architectures using Problem Frames', Proceedings 10th International Joint Conference on Requirements Engineering', IEEE Computer Society Press, 137-144.

Franz Huber, Sascha Molterer, Andreas Rausch, Bernhard Schatz, Marc Sihling, Oscar Slotosch, "Tool Supported Specification and Simulation of Distributed systems", Proceedings International Symposium on Software Engineering for parallel and Distributed Systems 1998, pp. 155-164.

Jackson M., 1995, 'Software Requirements and Specifications', Addison-Wesley.

Mijake N., 1986, 'Constructive Interaction and the Iterative Process of Understanding', Cognitive Science 10, 151-177.

Maiden N.A.M. & Pavan P., 2001, 'Manoeuvring Pattern Language version 1.3', Technical Report, Centre HCI Design, City Uiversity London, October 2001.

Maiden N.A.M., Jones S. & Flynn M., 2003, 'Innovative Requirements Engineering Applied to ATM', submitted to conference.

Maiden N.A.M., Pavan P., Gizikis A., Clause O., Kim H. & Zhu X., 2002, 'Making Decisions with Requirements: Integrating i* Goal Modelling and the AHP', Proceedings REFSQ'2002 Workshop, 9-10 September 2002, Essen, Germany.

Maiden N.A.M. & Rugg G., 1996, 'ACRE: Selecting Methods For Requirements Acquisition, Software Engineering Journal 11(3), 183-192.

Maiden N.A.M. & Sutcliffe A.G., 1996a, 'Analogical Retrieval in Reuse-Oriented Requirements Engineering', Software Engineering Journal 11(5), 281-292.

Maiden N.A.M. & Sutcliffe A.G., 1996b, 'Computational Mechanisms for Parallel Problem Decomposition During Requirements Engineering', Proceedings 8th International Workshop on Software Specification and Design, IEEE Computer Society Press, 159-163.

Robertson S. & Robertson J., 1999, 'Mastering the Requirements Process', Addison-Wesley-Longman.

Sutcliffe A.G. & Maiden N.A.M., 1998, 'The Domain Theory for Requirements Engineering, IEEE Transactions on Software Engineering, 24(3), 174-196.

Sutcliffe A.G., Maiden N.A.M., Minocha S. & Manuel D., 1998, 'Supporting Scenario-Based Requirements Engineering', IEEE Transactions on Software Engineering, 24(12), 1072-1088.

Zhu X., Maiden N.A.M. & Pavan P., 2003, 'Scenarios: Bringing Requirements and Architectures Together', submitted for publication