Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Computer Science 16 (2013) 283 – 292

Procedia Computer Science

Conference on Systems Engineering Research (CSER'13)
Eds.: C.J.J. Paredis, C. Bishop, D. Bodner, Georgia Institute of Technology, Atlanta, GA, March 19-22, 2013.

Using architecture patterns to architect and analyze systems of systems

Roy S. Kalawsky[a], D. Joannou, Y. Tian, A. Fayoumi

Loughborough University, Loughborough, Leicestershire, LE11 3TU, UK

Abstract

The inherent nature of a Systems of Systems (SoS) makes it very difficult to model and analyze it through conventional means. One of the first challenges faced is how to represent the SoS in a form that lends itself to detailed analysis, especially when full details of the component systems may not be readily available. Therefore, an important consideration is whether use of model abstractions can be sufficient to deal with many of the analysis needs of the SoS. It is clear there is a need for a new paradigm, encompassing methodology, models, tools and flows that enable the future engineering of SoS in order that they can be operated effectively. This paper describes how we are using architecture patterns to architect and analyze SoS in order that we can compare different architecture solutions and provide guidelines for the development of a future architectures based on the analysis of existing architectures. Insights are given to show the benefits for SoS architecture analysis with exemplars taken from a test case dealing with emergency response for a major incident in the UK. Our findings show the significant increase in SoS characterization that patterns can afford the systems architect in all phases of SoS evolution in order to deliver improved SoS capability.

Keywords: Systems of Systems; SoS; Systems Architecture; Architecture patterns; Model Based System Engineering (MBSE); Model driven architecture (MDA)

1. Introduction

Over the past two decades there has been a steady growth in the number of 'inter-connected' systems known as systems of systems (SoS). The term SoS in its more elemental form describes a collection of evolutionary components that are in their own right systems designed to achieve a common goal, examples include water management systems, airports, transport systems and many others. The original concept of SoS dates back to the early 1970's within the US defense industry community where the concept emerged of a battlefield being populated by a number of intelligent devices (autonomous aerial and terrestrial vehicles) communicating wirelessly among themselves and with humans geographically distant from the battlefield itself. The management of these devices was such that they were coordinated to achieve a common goal and can appear on the scene and (willingly or unwillingly), leave it at any time and in any order. The Defense Acquisition Guide [1] defines SoS as "*A SoS is*

[a] Corresponding author. Tel.: +44-1509-635678.
E-mail address: r.s.kalawsky@lboro.ac.uk.

doi:10.1016/j.procs.2013.01.030

defined as a set or arrangement of systems that results from independent systems integrated into a larger system that delivers unique capabilities". More recently it has been reported [2] that "*exploratory analysis of the portfolio of SoS in DoD indicates that SoS are pervasive across the DoD*". This, and other examples demonstrate SoS are inevitably very complex and bring together a collection of new and existing systems within a loose framework of overarching technological capabilities to fulfill a larger set of requirements [3]. The literature provides numerous definitions of what constitutes a SoS [4], Maier [5] noted that the term system-of-systems did not have a clear and accepted definition. Furthermore, he stated "*Systems-of-systems should be distinguished from large, but monolithic systems, by the independence of their components, their evolutionary nature, emergent behaviors and a geographic extent that limits the interaction of their components to information exchange*" [5]. Based on Maier's definitions of the characteristics of a SoS (with the addition of a sixth characteristic) are shown in Fig 1.

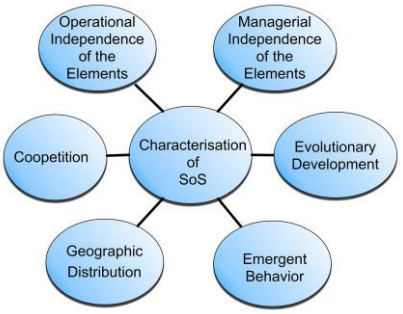

Fig. 1. Characteristics of a SoS

The above definitions and characterizations of a SoS help to illustrate the complex nature of these systems, and suggest they must be treated very carefully in order to understand the benefits. It is most interesting that many examples of SoS in operation today have resulted from 'spontaneous' evolution rather than by design, but happened unexpectedly over several years from a small 'seed' as typified by Gall's law: "*Complex systems that work evolve from simple systems that work*." [6]. The systems architect's role is to reconcile system form with the client's needs for function, cost, certification, and technical feasibility. However, in the case of a SoS, significant elements of the SoS may exist already and be outside the scope of the systems architect's control, in terms of being able to influence its operation or behavior. This is contrary to normal systems engineering where the systems architect would strive to optimize the whole system design against a number of key parameters across all components of the system. Supplementing this difficulty are differing classifications of a SoS; Directed, Acknowledged, Collaborative and Virtual, [1] magnifying the complexity issues which characterize a SoS. Consequently, today's complex systems and SoS make it extremely demanding for system architects to coherently pull together and manage the vast number of elements found within a system and the information exchange via copious interfaces both within the SoS, and with its external environment. As suggested by Brooks [7] the systems architect's responsibility goes far beyond the conceptual integrity of the system as seen by its users. It is the second of Maier's SoS characteristics that presents the biggest challenge for the SoS architect – 'evolutionary development'. The reason why this is so difficult in the context of a SoS is that the SoS exhibits operational independence of the elements which implies that constituent components of a SoS can evolve or change without due regard to other entities in the SoS, unlike a conventional system where full control is available over all the component parts. The inherent nature of a SoS makes it very difficult to model and analyze it through conventional means, especially how to represent the SoS in a form that lends itself to detailed analysis, especially when full details of the component systems may not be readily available. It is highly likely that the SoS will comprise a mix of legacy and new systems. During its evolution, while each of the constituent systems may be documented, it is not uncommon for the overall SoS documentation to be ignored. This is where new techniques are required which reduce the dependency on missing details of poorly specified component systems within the overall SoS architecture. Therefore, an important consideration is whether appropriate use of model abstractions can deal with many of the analysis needs of the SoS. It is clear there is a need for a new paradigm, encompassing methodology, models, tools and flows that enable the engineering of SoS from behavior to architecture, in order they can be operated effectively and their behavior analyzed. Also, during the lifespan of a SoS (which may never end) obsolescence will tend to drive the need to replace certain constituent system(s) during its lifespan to the point where the SoS becomes a complex heterogeneous mix of different systems each with different lifespans. It is not always possible, or even desirable to model all aspects of the component systems in order to model a SoS – the end result could be too huge to contemplate its execution on even the most

powerful computer. Our research is tackling this challenge by investigating architecting and analyzing SoS with modeling approaches through the use of patterns.

2. Engineering with patterns

2.1. Patterns as architectural blueprints

The use of patterns in certain areas of engineering is not new, design patterns have been used by software engineers during the design process and also when communicating designs to others [8, 9]. A classical and frequently quoted work describes the importance of patterns in constructing building and city architectures [10]. In essence, a pattern refers to recurring structures, objects and events, although they can also be used as designs, blueprints, models or templates in the construction of other structures, objects and events. In the latter case, newly created entities inherit the characteristics of the parent object (pattern). When used as creational elements, patterns can be used as the starting point to lay basic foundations, but the newly created entity can also evolve or be refined from the original design. Patterns can be descriptions (or templates) that capture practices that have proven successful in the past. It is important to note they are not prescriptive, but suggestive by including guidance on when their use is most appropriate and provides examples from existing systems. It has been stated [11] that "*A pattern is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts*". The notion of the form of a pattern is essentially defined through its representation as a set of interacting components and their relationships. Consequently a pattern has structural and dynamic properties whose form is realized through a finite number of visible and identifiable components. A component in this context can be technical or non-technical entities, services or even software. Expanding on this definition, we use the term architecture pattern as an expression of the architectural structure as opposed to the traditional design patterns encountered in the software engineering community. Two seminal works on patterns [10, 12] have proposed "Each pattern is a three part rule, which expresses a relation between a certain context, a problem, and a solution." This approach further explains that a pattern is a relationship between concepts that keep recurring in a specific context and a configuration. The pattern can be thought of as a recipe that describes how to create the particular entity and the context in which it can be used. The notion that a pattern [11] is a general repeatable solution to commonly occurring problems makes pattern reuse a big step in reducing system design risks. Although design patterns have been used extensively in the creation of elements of reusable object-orientated software [9], their role in supporting the architecting and analysis of ultra large-scale systems (known as SoS) is still very much in its infancy. The objective is to recognize which patterns can be found in a given SoS architecture description: if an element exhibits a pattern, this indicates important information for the systems analyst. When used correctly, patterns can provide an explicit way to articulate common concepts at the operational through to the detailed implementation levels. This in turn should ease the burden of characterizing the SoS for analysis studies but the approach should not be under-estimated.

An architectural pattern can be considered as a framework in the sense that it provides a template for the structure and behavior of an entire system within a domain [13]. The system's architecture model encapsulates decisions about the systems requirements, its logical elements and its physical elements. A system in this respect refers to an assembly of components or entities that can be interdependent or interacting (at a machine, services or human level). At the architectural level, patterns are very much concerned with the top most level blueprints (expressed in an architectural frameworkb such as Department of Defense Architectural Framework (DODAF), Ministry of Defense Architectural Framework (MODAF) or NATO Architectural Framework (NAF). Whichever approach is selected it is soon realized that each architectural framework offers a huge number of different viewpoints – not all of these are relevant and some are more useful than others in certain situations. Analysis of most system architectures will reveal many patterns. When applied correctly patterns play a huge part in architecting complex systems, they can be used to express common elements in a system design in a way that makes implementation easier. Also well-constructed patterns can be re-used in other places and make it easier to document and maintain existing systems through the use of a library or catalog of patterns. Numerous architectural frameworks exist (such as DODAF 2.02, UPDM, NAF etc.) that define a specific set of views that facilitate understanding of the overall system architecture through the different views. A view represents a behavioral, ontological, temporal or structural approach to describing the architecture in a pictorial manner. In the case of DODAF there are view families for; All Viewpoint (AV), Capability Viewpoint (CV), Data and Information

Viewpoint (DIV), Operational Viewpoint (OV), Project Viewpoint (PV), Services Viewpoint (SvcV), Standards Viewpoint (StdV) and Systems Viewpoint (SV). Each viewpoint can be further subdivided into more specific views, for example, for the SV there are fifteen specific views. It would be quite daunting to insist that all the views should be used when describing a given systems architecture. Instead, a limited set of the views in an architecture framework are generally used, the specific choice being down to the type of architecture and also the analysis that is being undertaken.

2.2. SoS hierarchy of patterns

Model-based design is becoming increasingly important in systems engineering [14] since model-based methodologies allow system designers to employ abstractions and model representations that match their design concerns rather than be constrained by specific limitations of a particular technological solution. This does not mean that very detailed implementation details can be ignored. However, working at higher abstract levels can help system designers see more clearly the top-level system-to-system interactions. This is particularly helpful in the case of SoS where the lower level implementation details may be hidden from the systems architect. In order to understand and reason about a SoS it is convenient to think in terms of a representational model based on a three-layered stack comprising operational, systems and component models respectively, refer to Fig. 2. At the highest level of abstraction is the operational model that defines the overall system architecture – the system architecture being completely independent of the way the underpinning systems and services are implemented. In essence this is the architectural framework on which the SoS exists. At the next layer down are the underpinning systems models – these are also implementation independent and enable a more specific model to be constructed comprising individual system models. Finally, at the lower level we have the component models that are implementation specific and encapsulate all the variables of a particular solution. Reference to Fig. 2 shows that there is some overlap between the boundaries of the three layers. This framework means that any particular SoS can be represented as an amalgamation of models across these three layers depending on the specific context of interest. An important approach to representing a SoS for analysis purposes is to consider expressing the SoS at systems architectural and systems design levels. The use of three pattern categories: Architectural, Interaction and Design (relating to the three-layered stack as shown in Fig 2) provides a clearer structure for any subsequent analysis of the SoS.

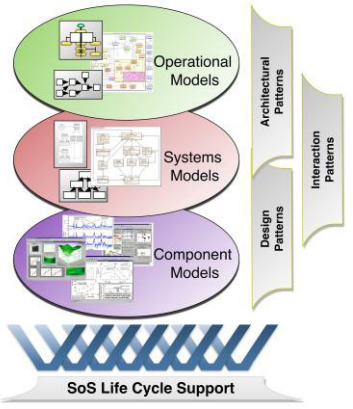

Fig. 2. SoS Hierarchy of Patterns

As system complexity increases it is not always possible to map a known solution onto a particular problem because the solution detail is too complex. In this situation, the nature of the problem to be solved is so complex that traditional approaches are inadequate to define the problem space. Unfortunately, conventional modeling and simulation techniques may not be suitable or even appropriate to provide tangible and verifiable results because the underlying system and data assumptions may be incomplete. This means there is a growing requirement to define system interactions at the operational level in abstract forms that permit a degree of analysis and understanding. Consequently, the ability to express a system in a manner that permits analysis at different levels and different viewpoints is fundamentally important. Without some form of framework in which to represent a SoS it is extremely difficult to see how the component system interactions can be understood, let alone optimized. This framework not

only needs to address the top-level operational concepts, it needs to be able to cascade down to the specific implementation level. When dealing with a very large system (such as a SoS the key challenge is how to partition the design into manageable entities or components for the systems designer. This is where higher order patterns called architecture patterns come into their own. The architecture pattern is drawn at a fairly abstract (system implementation independent) form and makes it easier to comprehend, implement and maintain.

Expressing system architectures through patterns provides system architects and designers with an opportunity to create libraries of reusable components based on prior experience or standard practices. Irrespective of whether we are dealing with a system architecture or system design it is possible to extract patterns (or templates) that relate specifically to architectural constraints or design specifics. Architectural patterns are not the same as design patterns because they deal with abstract and specifics respectively. Moreover, architectural patterns are conceived at the higher operational level as shown in Fig 2 whereas design patterns are applicable at the system and lower levels. Patterns are used in for many different areas in IT [15] such as design patterns, architectural patterns, interaction design patterns and security patterns. An understanding of patterns provides several benefits, particularly in that they provide a common language, which is independent of the underlying technology. During the course of design through to implementation a whole series of patterns at the different hierarchical level may be used. Consequently, the hierarchical set of patterns representing a SoS embodies and builds on the collective experience of a wide range of disciplines and promote good design practice.

2.3. Use of patterns to facilitate understanding of SoS evolution

We can consider a SoS almost as a continuum from the macroscopic SoS level through constituent systems all the way down to interactions at the molecular level. This might seem to be an absurd scale but we are dealing with interactions that can and do reveal themselves at different levels of the system representation framework. It has been mentioned that patterns can be considered at higher levels where instead of being confined to being within a single system boundary they can be applied across system boundaries so they help the focus of attention at the higher system-to-system level. The ability to do this is very important when in the context of a SoS it is necessary to consider a large scale enterprise in the form of a SoS where the SoS evolves in ways that were not originally conceived, or where parts of the SoS are actually services rather than system components. At the higher SoS level we are particularly interested in how the constituent systems are coupled together, where it's not just the nature of the physical connections that matter but the quality of service becomes a factor.

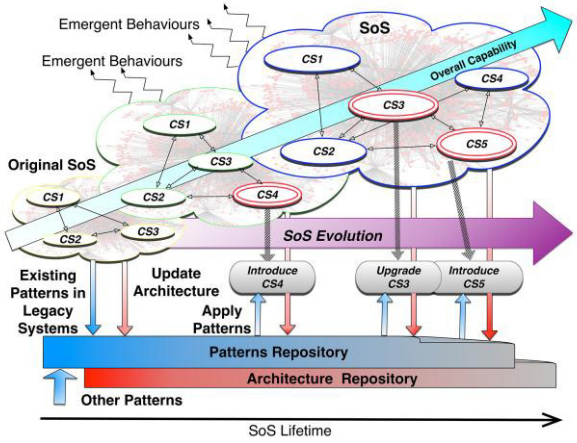

Fig.3. Illustration showing Evolution of SoS over time

The ever-changing, ever-adapting configuration of a SoS to meet its capability demands makes it distinguishably different from the nature of conventional systems. As already stated, a combination of a bottom-up and top-down (or a new method altogether) must be adopted for the management and development of these systems as the traditional top-down, once through approach will not suffice. The inherently dynamic nature of these multi-constituent, complex systems requires a constant monitoring of evolution as new constituents are added or interfaces are updated to meet existing protocols between the constituents. This evolution of the SoS over time may be managed or happen

in a spontaneous/uncontrolled manner. Additionally, new systems may join or leave the SoS without notice. Of particular interest to systems engineers is the notion of emergent behavior, which arises from systems coming together to exhibit new behavior that is not present in any of the individual systems alone. Ideally the new emergent behavior is positive but on occasions it is undesirable. If we assume that SoS generally evolve capability over time we need a way of representing current behavior as well as being able to represent future SoS states. Fig 3 illustrates a depiction of a SoS gradually transforming and increasing capability over time. It will be noted that over time more systems join, leave or interact with the original SoS. This is where architectural patterns help subsequent analysis of the evolving SoS. A pattern repository is required to store and make accessible to the architect, patterns that will help solve problems encountered when modifying the SoS. These patterns may be pre-existing patterns mined from the original ensemble of legacy systems making up the SoS, or from patterns mined in other domains but of which are relevant in the current application. By establishing a library or repository of architectural patterns for the SoS component systems it is possible to capture current state. More importantly, if additional data such as performance information is stored as part of the architectural pattern it makes it easier to model and simulate future states of the SoS with greater confidence. As well as building a pattern repository it is important to maintain an architecture repository (comprising a proven set of interacting patterns) that represents earlier versions of the SoS. This makes it easier to see how the SoS has evolved over a longer period of time.

3. Mining architectural patterns

Once a library of architectural patterns has been created it is possible to begin to explore the impact of alternative architectural solutions. Experienced systems engineering practitioners will be able to systematically extract the key information of interest or relevance and formulate patterns. When implemented correctly, patterns can incorporate performance information as well as structural abstractions that support a degree of analysis. In common with the field of enterprise architecture [16, 17] it may be necessary to consider architecture patterns within different architecture subdomains, and not for the domain of the SoS architecture as a whole. For instance it may be appropriate to consider workflow patterns as a subdomain where there is a control perspective, data perspective and resource perspective. As soon as a generic pattern is refined and adopted for use it can become difficult to recognize its origins unless this is included in the documentation. Consequently, the underlying pattern may be buried with the source code and no longer visible to future users with the result that certain important design information is lost. Consequently, trying to understand large complex system designs can be extremely difficult. Reverse engineering has been used but even this approach is fraught with problems – not to mention legal constraints in terms of attempts to reverse engineer someone else's design. Recent approaches to capture patterns have looked at pattern mining techniques [18-23] but these may introduce artifacts through the different approaches used to render patterns. The current best solution for mining architectural patterns is for an experienced system architect to extract patterns – they will more readily appreciate subtleties of the design that might be overlooked by a less experienced person. The key is to not only extract relevant patterns but also express these in forms that lend themselves to re-use at a later data. It is important to note that patterns should wherever possible be independent on specific implementations since this would render them less transferrable. Extraction of patterns requires a degree of intuition on the part of the system architect who will recognize certain patterns types but may need to create new pattern types. Care should be taken not to try and create huge patterns since these are less usable, but instead try and break the pattern down into smaller more recognizable elements. At the level of pattern architecture it is better to assume the set of patterns is recursive in the sense that they form a hierarchical structure comprising patterns and lower-level patterns. It can sometimes help to think in terms of levels of patterns – where Level 0 patterns exist at the highest level of abstraction and where successive lower-level patterns flesh out the detail.

3.1. Methodological approach to application of patterns for SoS

The use of patterns for evolving and analysis of SoS architectures is likely to become extremely important in the future because they facilitate abstraction at various levels of detail. In the case of integration with legacy system components the use of patterns may permit analysis of the SoS even through all system details may not be available. However, this is highly dependent on the quality of the pattern being used and how the pattern is to be deployed in the simulation or analysis. Before such patterns can be applied they need to be created or extracted from existing/legacy systems. As discussed previously patterns are typically descriptions (or templates) that capture practices that have proven successful in the past. They are not prescriptive, but suggestive by including guidance on

when their use is most appropriate and provides examples from existing systems. In essence, a pattern refers to recurring structures, objects and events although they can also be used as designs, blueprints, models or templates in the construction of other structures, objects and events. Also, it is important to note that patterns are hierarchical in the sense that high-level abstract patterns can be evolved into lower level patterns that can more specifically represent the implementation form of the components of a SoS. There are three key processes involved in the use of patterns for SoS. The first is clearly the creation of patterns, followed by pattern selection and then refinement of the pattern for subsequent use/deployment within the architecture of a SoS. There are plenty of well-documented examples of design patterns at the software implementation level [9, 24, 25], which have been used in the creation of modern software systems. However, patterns at the SoS architectural level are seriously lacking, meaning such patterns need to be created before they can be used. Since SoS are most likely to evolve from a collection or pre-existing systems it seems logical to mine patterns by examining the legacy system to see if it possible to create a representative pattern. In fact the pattern for a legacy system may be the only artifact that can be used to represent a given system on account of its inner operational structure being inaccessible. Also, in the case of a future evolving SoS the exact form the SoS takes may be unknown at the outset but through the use of appropriate architectural patterns it may be feasible to represent the SoS so that analysis can take place. In the first instance, experienced practitioners will need to extract specific patterns since they have the knowledge of what is important (and potentially re-usable) – the danger the less experienced person may fall into is expression of the pattern in too much detail that it becomes too implementation specific rather than more generally usable.

3.2. Example of using architectural patterns to understand emergency response SoS for a major incident

In order to illustrate how architectural patterns can aid the analysis of a SoS, an example is now described that is based on the SoS involved in the response of emergency services to a multi-agency major incident situation. Since the 9/11 incident, there has been an increasing interest in proposing improvements in the ability to respond to emergencies with the aim of mitigating the severity of the impact caused by an emergency [26]. The majority of these early efforts are focused on infrastructure improvements to mitigate the impacts of the disasters [26]. This has created a gap in the SoS principles that are directly related to understanding the interaction with interrelated systems and their impact on the infrastructure systems. The example outlined here is based on the major incident response SoS for London, known as 'The London Emergency Services Liaison Panel (LESLP)'. The SoS constituents include the Metropolitan Police Service, City of London Police, British Transport Police, the London Fire Brigade, the London Ambulance Service and local authorities. Additionally, the Port of London Authority (PLA), Marine Coastguard, RAF, Military and voluntary sector are also represented. The London Emergency Planning Procedure (LEPP) [27] has been used to as the basis for pattern mining which represent the interactions within this critical SoS. The interesting aspect is this SoS only exists when a major incident is declared and over the period which the various system components (emergency services) go about their usual separate business. The LESLP timeline for tackling a typical major incident comprises four phases: Initial response, consolidation phase, recovery phase and restoration of normality). The LEPP [27] is a textual document that describes the agreed procedures and arrangements for the effective co-ordination of their joint efforts. Unfortunately, in this form it is extremely difficult to understand all the complex interactions that need to take place between the different emergency services.

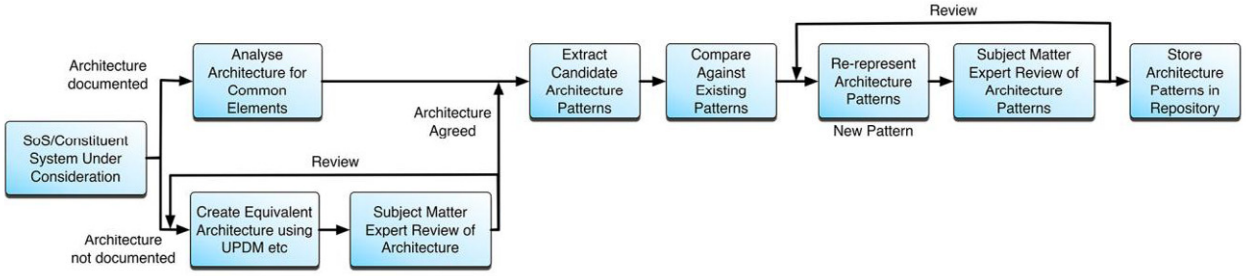

Fig. 4. Outline of the architecture pattern mining process

A comprehensive set of architectural patterns were mined from the LEPP using experienced systems architects who created a set of UPDM [28] view families that represents the Police, Fire and Ambulance Services. An outline of the architecture mining process is shown in Fig 4., which comprises an iterative process involving discussion with emergency service authorities confirmed, or otherwise until the UPDM models were correct. At this stage an experienced systems architect is key to the process since they are able to use their experience to abstract the key elements of the constituent systems of the SoS. At the heart of the LEPP operations is a Joint Emergency Services Control Centre (JESCC), which forms the focus from which the entire operations are managed. The extracted architecture pattern (Command Relationships View OV-4) for the JESCC is shown in Fig 5., and provides details of how a specific agency (emergency service) operates in such a scenario.

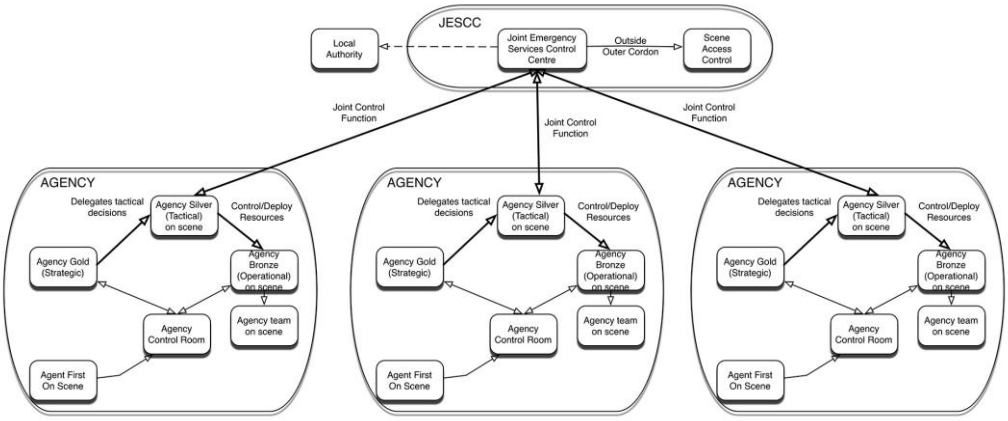

Fig. 5. Architecture pattern for the JESCC

In the diagram the term ‘agency’ refers to one or more of the emergency services. In the UK the ‘agency silver’ is responsible at the scene for all of that agency’s resources (tactical operations). It should be noted that the primary emergency service functions (fire, ambulance and police) are usually based at different geographical locations. The interesting question is whether any cost/performance benefits can be achieved by different arrangements such as centralizing the three emergency service functions in single locations across the country. Such an arrangement opens the door to different node connectivity patterns exist as shown in Fig.6., these offer different capabilities and levels of redundancy (please note it is the interconnections between the constituent systems that is important here).

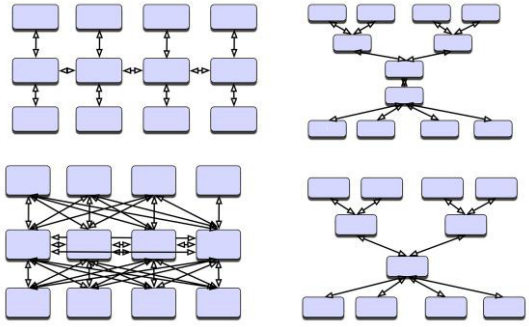

Fig. 6. Alternative node connectivity patterns

Architectural Patterns can also be identified within the individual agencies, for example the requesting of additional resources by a local commander is a common requirement between all agencies and an overview of this pattern can be seen in Fig. 7., as a sequence diagram. From such an architectural pattern (which is notably at a lower abstraction level) it may be desirable to form new configurations that increase the efficiency of requesting resources to the scene of the incident.

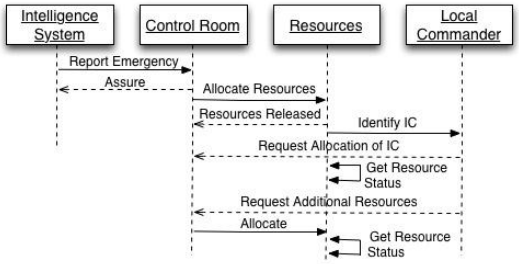

Fig. 7. Resource request pattern

3.3. Analysis of a SoS

It is clear that architectural patterns can be identified from the highest to the lowest levels of a SoS. Such architectural configurations make it possible to analyze possible architecture changes. SoS analysis can drive changes in the system to exploit opportunities or correct problems that were not originally anticipated. A key architectural tool in this respect is the use of predictive modeling and simulation to compare architectural alternatives. In any process for improving a SoS, alternative architectures would need to be very carefully considered and modeled to ensure the SoS is not compromised or undesirable emergent behavors result. Some of the elements that comprise a SoS may be outside the control of other elements in the SoS. This means the SoS may not be responsive to a single analysis, consequently, SoS analysis must be inherently incremental and the SoS should be available for testing almost on a continuous basis. Since SoS are potentially in a state of flux it is important to document specific changes to a pattern if it is modified during use. Whilst, a number of elements in a SoS may be verified in isolation, when they are integrated into a larger SoS then overall verification may not be possible. A number of options exist for analyzing the SoS but they depend on how comprehensive this needs to be. Architecture patterns can be readily specified in many of the architecture modeling frameworks and their associated tool environments. Tools such as IBM Rhapsody permit patterns to be stored in a repository and used in a what-if manner so that alternative architectural solutions can be evaluated through modeling and simulation approaches. However, care must be taken when creating or mining patterns to ensure they correctly represent the system of interest.

7. Conclusion

Whilst SoS consist of collections of constituent systems (possibly independent, pre-existing, geographically distributed and following their own goals) whose behaviors are coordinated to provide services and added value they can present difficult technical, management, and political challenges. At present, such large-scale systems are assembled haphazardly using common sense and already available components originally conceived for different purposes. Consequently, attempting to analyze a SoS to understand its behavior is a very complex undertaking. Whilst conventional modeling and simulation techniques go some way towards understanding the SoS they fall short of adequately being able to represent the entire SoS. This paper has discussed the use of patterns for creating a model of the SoS, representing the constituent systems. Representation of the overall SoS architecture by means of patterns is a significant step in understanding the operation of the SoS. Careful use of patterns as abstractions makes it feasible to create a reasonably good systems model. This provides the key to undertaking trade-offs between different solutions as one tries to optimize a particular aspect of the overall SoS.

Acknowledgements

This work was supported in part by European Commission for funding the Large-scale integrating project (IP) proposal under the ICT Call 7 (FP7-ICT-2011-7) ‘Designing for Adaptability and evolutioN in System of systems Engineering (DANSE)’.

References

1. Defense-Acquisition-University, *Defense Acquisition Guidebook*, 2011.
2. Dahmann, J. and K. Baldwin. *Implications of Systems of Systems on System Design and Engineering*. in *6th International Conference on System of Systems Engineering,*. 2011. Albuquerque, New Mexico, USA.
3. Corsello, M.A., *System-of-Systems Architectural Considerations for Complex Environments and Evolving Requirements.* IEEE Systems Journal, 2008. **2**(3).
4. Jamshidi, M., *System of Systems Engineering: Innovations for the 21st Century*. Wiley Series in Systems Engineering and Management, ed. A.P. Sage2008, Hoboken, New Jersey, USA: Wiley.
5. Maier, M.W., *Architecting principles for systems-of-systems.* Systems Engineering, 1999. **1**(4): p. 267–284.
6. Gall, J., *SYSTEMANTICS: The Underground Text of Systems Lore. How Systems Really Work and How They Fail*. Second ed1986: General Systemantics Press.
7. Brooks, F.P., *The Mythical Man-Month: Essays on Software Engineering - (anniversary ed.)*1995: Addison-Wesley Longman Publishing Co. 322.
8. Buschmann, F., et al., *Pattern-Oriented Software Architecture: A System of Patterns*. .1996: Wiley.
9. Gamma, E.R.J., R. Helm, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Orientated Software*2008, Massachusetts: Addison-Wesley.
10. Alexander, C., S. Ishikawa, and M. Silverstein, *A Pattern Language: Towns, Buildings, Construction*1977, New York: Oxford University Press.
11. Riehle, D. and H. Züllighoven, *Understanding and using patterns in software development* Theory and Practice of Object Systems - Special Issue on Patterns, 1996. **2**: p. 3-13.
12. Alexander, C., *The Timeless Way of Building*1979, New York: Oxford University Press.
13. Booch, G., J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*1999: Addison Wesley Longman, Inc.
14. Sangiovanni-Vincentelli, A., et al., *Metamodeling: An Emerging Representation Paradigm for System-Level Design.* IEEE Design & Test of Computers, 2009. **26**(3): p. 54-69.
15. Zachman, J.A., *A Framework for Information Systems Architecture.* IBM Systems Journal, 1999. **38**(2&3).
16. Aalst, W.M.P.V.D., et al. *Workflow Patterns Initiative: Workflow Patterns (WWW document)*. 2009 [cited 2012 12 July 2012]; Available from: http://www.workflowpatterns.com.
17. Aalst, W.M.P.V.D., et al., *Workflow Patterns.* Distributed and Parallel Databases, 2003. **14**: p. 5-51.
18. N. Tsantalis, et al., *Design Pattern Detection Using Similarity Scoring.* IEEE transaction on software engineering, 2006. **32**(11).
19. Wendehals, L. *Improving design pattern instance recognition by dynamic analysis*. in *Proceedings of the ICSE workshop on Dynamic Analysis*. 2003.
20. Zhang, Z., Q. Li, and K. Ben. *A new method for design pattern mining*. in *Proceedings of the 3rd International Conference on Machine Learning and Cybernetics*. 2004.
21. Heyuan Huang, et al., *A practical pattern recovery approach based on both structural and behavioral analysis.* Journal of Systems and Software, 2005. **75**(1-2): p. 69-87
22. Jing Dong, Yajing Zhao, and T. Peng. *Architecture and Design Pattern Discovery Techniques – A Review*. in *Proceedings of the 6th International Workshop on System/Software Architectures (IWSSA)*. 2007. USA.
23. Jing Dong, Yongtao Sun, and Y. Zhao. *Design Pattern Detection By Template Matching*. in *Proceedings of the 23rd Annual ACM Symposium on Applied Computing (SAC)*. 2008. Ceará, Brazil.
24. Gomaa, H., *Software modeling and design : UML, use cases, architecture, and patterns*2011, Cambridge ; New York: Cambridge University Press.
25. Welie, M.v. and H. Trætteberg. *Interactions patterns in user interfaces*. in *7th Pattern Languages of Programs Conference (PLoP)*. 2000.
26. Turoff, M., et al., *The Design of a Dynamic Emergency Response Management Information System (DERMIS).* Journal of Information Technology Theory and Application (JITTA), 2004. **5**(4): p. 1-35.
27. The-Stationery-Office, *Major Incident London Emergency Services Liaison Panel (LESLP) Manual*. 8th ed2012: The Stationery Office.
28. OMG, *Unified Profile for DoDAF and MODAF (UPDM), version 2.0*, 2012, OMG.