Can System of Systems Be Given Self-X Requirement Engineering Capabilities?

Alisdair MacDiarmid, Peter Lindsay

School of Information Technology & Electrical Engineering,

The University of Queensland, Brisbane (St. Lucia), Australia 4072

Abstract. System of Systems (SoS) are a relatively recent phenomenon and present a whole new set of challenges for systems engineers. The system elements of an SoS are often managed and operated in a predominantly independent manner, over widely distributed geographic locations and are subject to evolution with various rates of change. The goals of the SoS itself often change over time. One purpose of this paper is to survey the literature on requirements management issues that are brought to the fore as a result of these and other SoS characteristics. We then explore a vision of how the key artefacts of requirements engineering might need to evolve, together with their supporting tools and processes, in order to better support the development, operation and maintenance of SoS's. The vision is inspired by the autonomic computing paradigm, in which computing systems are equipped with self-x capabilities – such as self-configuration and self-healing – in order to manage themselves. Rather than presenting a solution our purpose is to better understand the new requirements engineering capabilities that will be required for SoS.

1. INTRODUCTION

Background. Recent advances in software and hardware computational and processing capabilities have led to systems that were previously stand-alone now being connected to each other and becoming increasingly dependent on each other. Furthermore, the systems are increasingly distributed, with system elements many hundreds, or even thousands, of kilometers remote from each other in many cases. This has led, in turn, to the creation of System of Systems (SoS's), whereby a number of these distributed systems grouped together constitute a larger, over-arching system. Examples include the US DoD Future Combat System (FCS), space exploration missions, medical and health management services, and air transport operations (DiMario 2006; Hata, Kobashi et al. 2009; Jamshidi 2009).

Given that SoS are a relatively recent innovation, there are many aspects that require further understanding and are the subject of significant research. This research is given added impetus due to the increasing number, scale and cost of SoS projects now being developed in defence, space and commercial aviation businesses. Furthermore, the potential offered by SoS solutions is opening up research into new applications in areas such as healthcare (Hata, Kamozaki et al. 2007; Wickramasinghe, Chalasani et al. 2007).

Requirements management is central to system engineering activities, and this applies equally as well to SoS applications. We survey the SoS literature and highlight some of the particular requirements management issues that are brought to the fore by SoS. SoS characteristics that present particular challenges to requirements engineering include emergent properties, independent management and/or operation of the systems that make up an SoS, and a lifecycle that is typically more evolutionary in nature than the traditional standalone-system lifecycle (Simpson and Dagli 2008). It is becoming clear that the traditional human-centric-process approach to requirement engineering cannot be a complete solution for SoS and that new tools and procedures will be required, taking better advantage of advances in Information and Communications Technology (Keating, Padilla et al. 2008; Lewis, Morris et al. 2009).

Self-X Capabilities. We explore a vision of how the key artefacts of requirements engineering (such as goals, requirements specifications and as-built/as-operated system specifications) might need to evolve, along with their supporting tools and processes, in order to best support the development, operation and maintenance of an SoS. The vision is inspired by the autonomic computing paradigm, in which computing systems (or requirements specifications, in our case) are equipped with self-x capabilities in order to "manage themselves given high-level objectives from administrators" (Kephart and Chess 2003).

Autonomic computing is a key approach to dealing with the increasing complexity of computer-based systems (Murch 2004). The four self-x capabilities that Murch believes are most important for achieving autonomous system behaviour are self-configuring, self-healing, self-optimising and self-protecting behaviours. Self-configuring abilities enable autonomous identification and management of system-element functional and physical characteristics. Self-healing capabilities involve self diagnosis and repair of detected problems within the system. Self-optimisation occurs where the system monitors its own performance and decides by itself how to improve execution. Finally, self-protection is about the system defending itself from malicious attacks from external entities. Other self-x values, including self-adaptation and self-organisation, are discussed in the literature (Markose 2005; Seebach, Ortmeier et al. 2007).

Figure 1 illustrates a model proposed by (Kephart and Chess 2003) which represents an implementation architecture of the elements in an autonomic computing system.

Each system element consists of a managed element and an autonomic manager. The managed element corresponds to the classical hardware or software configuration item, while the autonomic manager is the part that enables the system element to monitor the external environment and its own 'managed element', and execute plans based on their status. The autonomic manager performs the self-x actions through execution of policies in the monitor, analyse, plan and execute cycle. The policies and manner of their execution reside in the knowledge component of the autonomic manager.

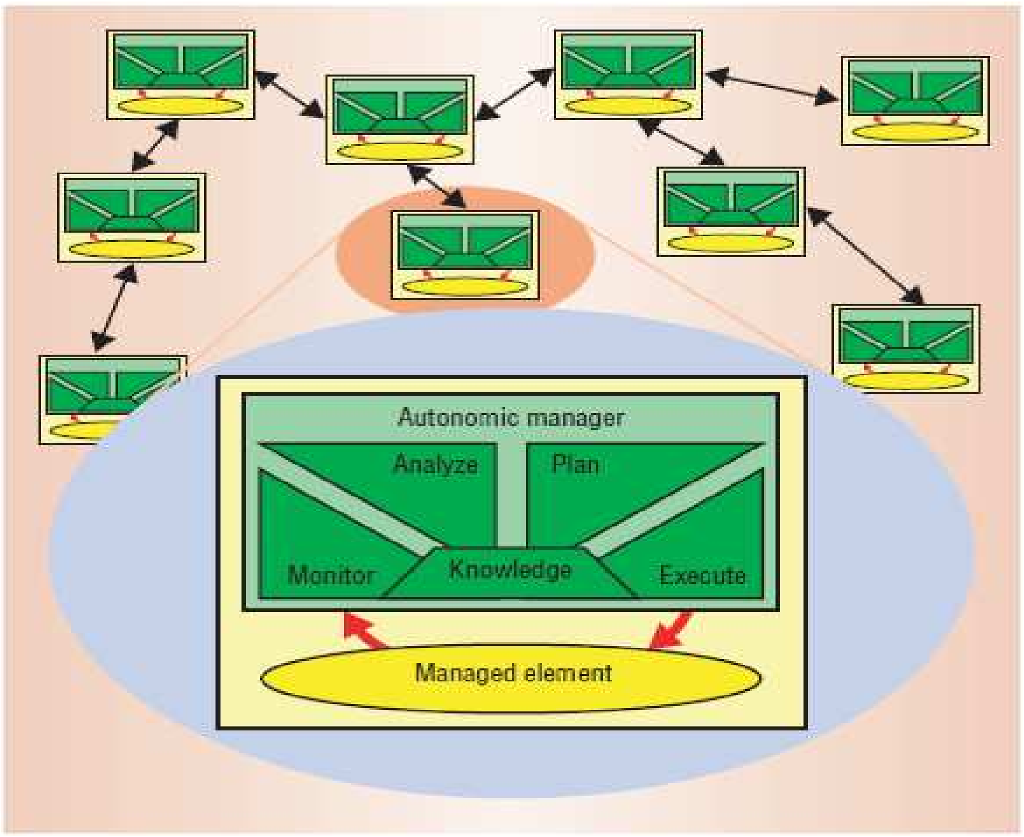

Figure 1. Autonomic Computing Architecture

Research Motivation. Whilst there is a very active research effort into understanding autonomic computing and autonomous systems, the target applications are predominantly computer and IT systems (Murch 2004; White, Hanson et al. 2004; Doyle, Kaminsky et al. 2005). Our motivation here is to explore the potential for carrying across ideas from autonomic computing to requirements engineering for SoS applications. Specifically, we consider how SoS elements and their associated requirement engineering artefacts might be equipped with self-x capabilities such as self-configuration and self-optimisation, in order to address key SoS requirements management issues. Rather than setting out a solution however, this paper is concerned with understanding requirements for SoS requirements management, and exploring the capabilities that will be required.

Present requirements models in the literature still rely primarily upon human-centric processes in specification of requirements or goals (Kephart and Chess 2003). We would agree that there is still a role for human involvement in specifying goals in SoS but we suggest that this task also needs to become part of the autonomous behaviour of the SoS. In fact, the human role will more so be the provision of global goals for the SoS. The SoS will treat these global goals as inputs, analyse them, plan their allocation amongst elements of the SoS, assess the SoS capability to achieve the goals, and monitor achievement of the goals by the SoS. Where the SoS believes it does not have the capability to meet a goal, it could quite possibly present necessary goals or requirements to the human operator which identify new SoS elements that need to be incorporated into the SoS architecture to add desired functionality.

The self-configuring behaviour is represented by the ability of the SoS to manage its own requirements and goals, thus reducing the complexity of the workload on the human operator. Furthermore, the ability of the SoS to ensure meaningful utility of its system elements illustrates the self-optimising behaviour of the SoS.

We see the above approach addressing certain key issues that exist in engineering of SoS. Questions that immediately arise from the central issues we consider in this paper are:

(1) In what ways might we imbue SoS with the ability to accurately capture, within goals and requirements, evolving SoS needs?

(2) How may we improve means of mapping the goals and requirements to the realised SoS implementation?

We return to these questions in later sections of the paper.

The remainder of this paper is arranged in the following manner. In Section 2 we review the literature on the different types of SoS that have been identified, their key characteristics, and the engineering challenges they present. We then go on to present what we perceive as the key characteristics of SoS in enabling self-x requirement behaviours. Section 3 summarises the nature of artefacts under different approaches to requirements engineering, as background to later discussion. We then marry our thoughts from these two sections in Section 4 ("Requirements in SoS") and discuss how enabling self-x behaviour of goals and requirements may address some of the current SoS challenges. We then briefly discuss our intended research and further work in Section 5. In the final section we present our conclusions from this paper.

2. SYSTEM OF SYSTEMS

Definition of an SoS? The definition of an SoS is, in itself, a matter of some conjecture. Maier noted in 1998 that there was "no widely accepted definition of its meaning" (Maier 1998) and we would suggest this still holds true today. A key issue in the discussion relates to how we differentiate between the definition of a system and the definition of an SoS.

The ISO/IEC 15288:2008 Systems and software engineering — System life cycle processes standard defines a system as:

System - combination of interacting elements organized to achieve one or more stated purposes.

Further, the standard then defines each element of a system as:

System Element - member of a set of elements that constitutes a system.

Notes in the standard clarify that a "system element can be hardware, software, data, humans, processes (e.g., processes for providing service to users), procedures (e.g., operator instructions), facilities, materials, and naturally occurring entities (e.g., water, organisms, minerals), or any combination."

We take the ISO/IEC 15288:2008 definitions of a system and system element as sufficient, as these definitions are very mature and have resulted from extensive consideration, both prior and during, the preparation of the standard.

Now we consider SoS definitions from the literature. (Jamshidi 2009) presents a number of definitions sourced from the literature, including the following:

"System-of-systems integration is a method to pursue development, integration, interoperability, and optimization of systems to enhance performance in future battlefield scenarios [Pei, 2000];

Systems of systems exist when there is a presence of a majority of the following five characteristics: operational and managerial independence, geographic distribution, emergent behavior, and evolutionary development [Jamshidi, 2005];

Systems of systems are large-scale concurrent and distributed systems that are comprised of complex systems [Jamshidi, 2005; Carlock and Fenton, 2001];

SoSE involves the integration of systems into systems of systems that ultimately contribute to evolution of the social infrastructure [Luskasik, 1998]."

Jamshidi then proceeds to offer a further definition whereby "systems of systems are large-scale integrated systems which are heterogeneous and independently operable on their own, but are networked together for a common goal"(Jamshidi 2009).

In addition, (Sauser and Boardman 2008) include the term of autonomy. Their definition of the desired autonomy is related to the SoS constituent systems having the "ability to make independent choices".

A final definition worth reviewing is that from the US DoD who define an SoS as "a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities" (DoD 2008).

We see there are a number of common themes in these definitions including the integration of constituent systems and a sense that the constituent systems still operate to some extent independently, although all should be collaboratively working toward SoS goals. Additionally, the notion of evolutionary behaviour is included to highlight one of the key differences from classic systems; namely, that the lifecycle is not a single pass through the engineering process but is more akin to natural systems, where change is an expected behavioural trait. Associated with this evolutionary behaviour is the property of emergence where some SoS behaviours are derived from the combination of the constituent systems, but not attributable in a direct sense to one or more of these constituent systems.

There is also debate surrounding the characteristics that may be associated only with SoS, as opposed to being associated with any system. The suggestion here is a taxonomic one, whereby SoS may be differentiated from other non-SoS systems by the inclusion or exclusion of certain characteristics. (Maier 1998) suggests that geographical location and component system complexity, are not appropriate classifiers as they do not meet "the test of being discriminating characteristics for distinctly different design approaches, when the appropriate examples are considered." However others, as evidenced in (Jamshidi 2009) do see these characteristics as forming part of a generic SoS definition.

The preferred definition that we would propose is of an SoS being:

A networked group of multi-scale SoS elements, which exhibit independence and diversity, but evolve together for a set of common goals.

We introduce the term 'SoS Element' in an attempt to remove the ambiguity and confusion that often occurs through the term system, or even constituent system. In cases, where an SoS element is itself an SoS, a suitable nomenclature for each SoS will assist in removing confusion about which SoS is the subject.

Furthermore, we choose the use of system-of-systems as opposed to the plurality of systems-of-systems as we suggest that in a practical sense, as opposed to a theoretical sense, there remains a boundary to our SoS, even though it is made up of very many individual SoS elements. Therefore, ultimately there is an overall singular

SoS boundary between the SoS and the external environment. By multi-scale we acknowledge that an SoS may have certain SoS elements that do not warrant the 'large-scale' definition. However we would suggest that these smaller scale SoS elements may be just as relevant and significant to the SoS because of other characteristics, such as complexity or evolutionary nature etc. For example, an autonomous Unmanned Aerial Vehicle (UAV), in some hypothetical Future Combat SoS, may be viewed as an essential element due to its multi-role capabilities but would not necessarily be described, in itself, as a large-scale system. Certain independent SoS elements may additionally for example, at some point in time, evolve into an SoS within the overall SoS e.g. our previous individual UAV may become grouped with other unmanned, autonomous systems to form an SoS within the Future Combat SoS. Indeed the converse may equally occur. Independence and diversity reflect, and allow for, autonomous behaviour within the SoS. Explicit in our definition is the notion that the SoS is evolutionary in nature and change is expected as a usual element of the SoS lifecycle. Finally we allow for the SoS to exist in response to the achievement of one or more goals. Otherwise, why would the SoS exist? 'Goal' and 'requirement' are both used within requirements engineering. Our approach is to primarily talk of goals, with 'requirement' viewed as a specialisation of 'goal'.

So, now in the next section, we turn to reviewing the key characteristics that have been identified in the SoS definitions.

Characteristics of SoS. The term 'characteristics' is used here interchangeably with other terms in the literature, such as 'properties' (Bjelkemyr, Semere et al. 2007).

One characteristic of many SoS's is that one or more of their elements are managed and/or operated by different organizations. For example, (Maier 1998) gives this as one of the defining characteristics of an SoS. Indeed, (Maier 1998) asserts that a system without operational and managerial independence of its elements is not an SoS. Additionally, he claims that the validity of these two properties is the definitive classification of a system as an SoS, "no matter the complexity of the subsystems." The essence of Maier's operational and managerial independence is the concept that the SoS elements are capable of, and further, do operate independently. So, while the SoS elements have some joint purpose, they are capable of separate operation, with individual purpose, if at some point they were no longer part of the SoS. By way of contrast, Maier does not believe that other proposed characteristics such as complexity are suitable for differentiating between SoS and non-SoS instances. Maier is suggesting that complexity, whether low or high, is not a discriminating characteristic between SoS and non-SoS systems; not that SoS or non-SoS may in fact be complex entities. This view is supported by (Sheard and Mostashari 2009) who propose in their recent work that "systems-of-systems are often, but not always, complex systems." However, this view on the relationship between SoS and complexity is questioned by other research, including (Bjelkemyr, Semere et al. 2007) where their proposed taxonomy does suggest SoS and non-SoS may be differentiated by the degree to which the system in question exhibits complexity. Indeed, (Bjelkemyr, Semere et al. 2007) have, in their work, suggested that operational and managerial independence are a subset from the set of complex properties that may arbitrarily be exhibited by both SoS and non-SoS. The characteristics of operational and managerial independence have been discussed by others in furthering the characterisation of SoS (DeLaurentis 2008; Keating, Padilla et al. 2008; Lewis, Morris et al. 2008).

Interestingly, operational and managerial independence have been associated with the SoS characteristic of autonomy (Sauser and Boardman 2008). This is due to the sense of independence which is engendered by the term autonomy, with its obvious links to the operational and managerial independence characteristics. There are a couple of points worth mentioning in relation to the Sauser et al suggestion of autonomy as an SoS characteristic; the first explicitly raised by Sauser et al, while the second we suggest based on their work. In the first case, Sauser et al state that "the autonomy characteristic might express extremes of creative disobedience and conformed acquiescence." This suggests that SoS behaviour, as determined by the autonomy characteristic of the SoS, may exist on some continuum. (DeLaurentis 2008) supports this viewpoint with the autonomy characteristic as part of a three dimensional taxonomy, where a control axis may extend from fully centralized (no autonomous behaviour) to fully autonomous.

The evolutionary nature of SoS, with the non-coherent changes within SoS elements, changes in the SoS environment and unknown order of these various changes is discussed elsewhere by (Sauser and Boardman 2008). However, what is not stated explicitly, but which, as our second point, we would suggest, is the relation of autonomy to the evolutionary characteristic. Whilst it is desirable that the SoS elements retain the independence attributes of the autonomy characteristic, for the SoS is to be meaningful we need the SoS elements to exercise their autonomous behaviour such that the evolutionary behaviour of the SoS is towards its nominated purpose. There needs to be some collaborative means by which this balance is managed or, if need be, a determination is agreed that the SoS element needs to be replaced or removed from the SoS. Requirements engineering (RE) of the SoS may form part of the solution in addressing the issues raised by these two points, related to the autonomy characteristic.

Collaboration between SoS elements. We have so far made indirect reference to a key SoS characteristic that warrants further discussion, namely collaboration. Collaboration between SoS elements is vitally important if the SoS is expected to achieve some higher order goals. It is necessary via collaboration to support SoS element awareness of other SoS element functionality, expected SoS element physical and logical states, levels of autonomy and impact of emergence on SoS elements. (Maier 1998) proposes a taxonomic grouping of SoS's into three classifications based on the nature of how their elements collaborate to set and achieve overall SoS goals. These are *directed*, *collaborative* and *virtual*. (DoD 2008) added a fourth type: *acknowledged*. The 4 types will be explained briefly below.

Virtual SoS lack any central purpose and are generally, at most, informally guided by the users. Maier points to the World Wide Web as a virtual SoS example, where there is minimal control and "the purposes the system fulfills are dynamic and change at the whim of the users." Acknowledged SoS have structure at the SoS level, constituting managerial resource and operational objectives, as well as fully independent SoS elements, in the sense of ownership, funding and purpose. Some fielded communication systems could be described as acknowledged SoS, as centrally coordinated data fusion takes place on communications transceived by different types of transmission (Very High Frequency, High Frequency, Satellite). There is a fine line distinction between acknowledged and collaborative SoS types. Collaborative SoS are basically the same as acknowledged SoS, except the collaboration is more expected and inherent between the SoS elements. However, the central SoS authority still does not mandate execution of the SoS elements. In contrast, a directed SoS is typified by low levels of collaboration. In this case the central SoS authority controls, mandates and so directs the formulation and lifecycle of the SoS elements.

Collaboration within SoS occurs for many purposes and is required in the realisation of many other characteristics. Some of these characteristics have already been mentioned, while others can be found elsewhere in the literature (Bjelkemyr, Semere et al. 2007; Lewis, Morris et al. 2008; Boxer and Garcia 2009).

Engineering of SoS. Some suggest that the engineering effort to enable collaboration within SoS may be obtained through employment of readily available and classical systems engineering approaches. Clark is one proponent of the view that SoS may be managed using "processes as documented in the SE standards: IEEE1220, EIA/IS-632, ISO 15288, and the guide: ISO TR 19760" without the need for additional processes, specifically for SoS (Clark 2009). Others such as Dallal-Shwartz et al. state that "classical methods address 'single (stand alone) system' development and do not include a network layered view, and as such, they are ineffective for SoS execution" (Dallal-Shwartz, Rabinowitz et al. 2009).

It is worth noting that an engineered SoS not only comprises physical SoS elements but the attendant design artefacts, which may include Concept of Operations (CONOPS) and 'As Built' specifications. All these artefacts require ongoing management by owners.

An integral facet of classical systems engineering which is used to capture and manage functionality, physical attributes and necessary interfaces within and between non-SoS system elements is requirements engineering. We believe that in SoS, requirements engineering will continue to play a vital role in enabling the necessary collaboration but, the implementation and realisation of the requirements engineering approach may be quite different. Table 1 lists the four types of SoS and illustrates the respective requirements engineering relationship we believe exists in each case.

The breakdown of requirements engineering characteristics by SoS type is only a first step and a number of questions become apparent if we are to determine appropriate requirements engineering techniques for SoS, and correspondingly when we should use a particular approach. At this point, we remind the reader of two questions posed in the Introduction regarding implementation of SoS:

(1) In what ways might we imbue SoS with the ability to accurately capture, within goals and requirements, evolving SoS needs?

(2) How may we improve means of mapping the goals and requirements to the realised SoS implementation?

When we consider these questions in the context of Table 1, some further questions arise, including:

(a) As an evolving SoS generates new needs, which in turn drive new global goals, what structures are required to facilitate collaboration of goals and requirements between SoS elements and, where applicable, central authorities?

(b) What methodologies will enable this collaboration to occur in a timely manner, and additionally, in an optimal manner?

(c) Which roles may requirements engineering play in SoS assessment of extant implementation capabilities against evolving needs?

(d) How may we structure SoS Element-to-SoS Element collaboration so that it will inform the SoS of additional functionality required to achieve desired global goals?

Questions (a) and (b) follow on from question (1), while questions (c) and (d) relate to question (2). Before we are able to go any way towards answering these questions, we need to assess what requirements engineering representations are already available for consideration. This we do in Section 3, after which, we will return to consider these questions in the Section 4.

SoS Type	RE Relationship
Directed	Classical RE methods; Each SoS element clearly defined by central RE authority; SoS RE evolution controlled and coordinated by central authority; Central allocation of requirements.
Acknowledged	RE performed by SoS central authority; RE also performed independently by SoS elements; Infrequent collaboration of RE artefacts.

Collaborative	RE performed by SoS elements;
	Central authority limited to expression of global SoS goals;
	High levels of RE collaboration;

Virtual	No central authority RE input; SoS element RE informal and irregular, if at all.

Table 1: SoS Type to RE Characteristic

3. REQUIREMENTS REPRESENTATIONS

This section briefly summarises different approaches to requirements engineering and the nature of the key artefacts, tools and processes involved, as background to our discussion of capability improvements required for SoS.

Traditional Textual Approach. The requirements engineering methodology most associated with classical systems engineering is that of text based representations. This approach is ubiquitous across many industries and is still the method used on many projects today to capture a customer's desired needs in a contractual format. However, even with the use of data based tools such as CORE and DOORS, the limitations of this methodology is being recognized due to increasing complexity in implementation (Alexander and Stevens 2002; Weber and Weisbrod 2002). This may be due to various factors including the number of system elements or systems, the number of stakeholders who may be widely distributed and where there is incomplete definition of operators' needs at the beginning i.e. the system in question has to incrementally evolve over time.

For the reasons identified above and as a result of an observed increase in evolutionary rate of change, due to the rise and rise of computational and electronic capabilities, traditional text based approaches are not directly suitable for SoS implementations. By the term directly, we mean that text based semantics may still play some role but we would not expect classical text based requirements specifications to be sufficient. The knowledge management task quickly overloads human capabilities, so tools and methods are required for structuring the information, and enabling support in the understanding and management of it.

Scenario/Use Cases Approach. Scenarios and use cases are further tools that are used extensively in classical systems engineering. Furthermore, while initially used independently within environments such as requirement identification workshops, they have more recently been incorporated into software applications, such as the OMG UML (OMG Cited 2010) and SySML (OMG Cited 2010). They assist in providing pictorial views of how systems are used by various stakeholders, and the sequence of events within the operation of a system. (Alexander and Stevens 2002) have raised what may be a limiting issue whereby "when there are many use cases, fitting them together is a serious problem." Various scenarios may be created from the combination and permutation of the many use cases.

The discussion above does not imply that use cases and scenarios have reached some "used-by" date. They may always have a place in illustrating some of the ways the SoS may operate, but on their own they do not adequately capture system dynamics such as, for example, the range of different reconfigurations possible. In an SoS context, there is a combinatorial explosion in the number of different cases that need to be considered and this also drives the need to investigate better ways of structuring such information. Indeed, there is research being undertaken into improved techniques that aim to address the problems associated with these issues, such as scenario to requirements mapping (Alrajeh, Ray et al. 2007).

The advent of use cases and scenarios highlighted another instance of language implementation which has generated further confusion at times within the field of systems engineering as mentioned in Section 2; namely the use of 'goal' versus 'requirement'.

Modelling Techniques. Goal-oriented requirements analysis has become a rich source of study, especially with its ability to assist in formulating modelling techniques of requirements and goals. Two important approaches to goal oriented representation of requirements engineering are KAOS (Dardenne, Van Lamsweerde et al. 1993) and Non-Functional Requirements (NFR) framework (Mylopoulos, Chung et al. 1999). KAOS presents formal semantics for requirements modelling which are useful in describing many facets of systems, including stakeholders, or agents in the artificial intelligence sense, goals and events. The NFR framework and KAOS share some common concepts; namely those of goals, agents and AND/OR/XOR relationships. However, while KAOS is more so concerned with the investigation of design possibilities from high level goals, the NFR framework looks more specifically at the non-functional requirements and goals. (Letier 2001), who employs KAOS in his work, suggests that the goal oriented approach to requirements engineering is appropriate as goals "are well suited to support the exploration of alternative designs involving multiple agents and the handling of agent misbehaviours." Furthermore, he recognises that "the introduction of a new agent arises from the need to fulfill some system-wide goals." Therefore, although an SoS was not under consideration here, the goal oriented approach seems to offer promise in SoS applications.

UML and SySML (OMG Cited 2010; OMG Cited 2010) are two modelling tools that have found extensive use, in the first instance within the software domain, and then broader use within the larger systems engineering domain. They provide the practitioner with object oriented visual representations of a system, through the use of diagrams. The types of diagram available include behaviour, structure, activity, use case and block – all representing different views in an attempt to elicit greater understanding for system stakeholders and designers. However, UML and SysML have weaknesses for use in requirements engineering including, weak support for diagram connectivity, weak support for allocation hierarchy and, weak mathematical foundations (Fogarty and Austin 2009).

Use of Meta-Models. In addition to the work on KAOS and the NFR framework, there has been very active research in the application of meta-modelling for requirements engineering (Navarro, Mocholi et al. 2006; Brottier, Baudry et al. 2007; Goknil, Kurtev et al. 2008). Meta-modelling is useful as it allows us to capture core goal concepts and relationships. The goal metamodel is typically constructed by identifying goal artefacts, or characteristics, together with any attributes associated with these artefacts, and the relationships between artefacts. (Goknil, Kurtev et al. 2008) for example, construct a core metamodel where 'Requirement', 'Stakeholder' and 'Relationship' are some of the artefacts. The 'Requirement' artefact has 'ID', 'Name', 'Description', 'Priority', 'Reason' and 'Status' as attributes, and has a relationship to the 'Stakeholder' artefact of one to many. Furthermore, meta-models may be "tailored according to the specific needs of expressiveness." (Navarro, Mocholi et al. 2006). This, in turn, enables multiple models to be generated from a single meta-model for different customizations or instances.

A common theme in the literature is the incorporation of other requirements engineering techniques in the creation of meta-models, in attempts to address shortfalls which otherwise exist in these other techniques. So, we see (Brottier, Baudry et al. 2007) suggest a multi-part process where firstly, two textual specifications are parsed into specification models, which are instances of an Input Language Meta-model. Secondly, these discrete specification models undergo model transformation into intermediate models, which are instances of a second Core Requirements Meta-model. Finally these intermediate models are combined into a global requirements model. Similarly, (Goknil, Kurtev et al. 2008) propose a meta-model of the SySML tool with mapping to a Core Requirements Meta-model which "contains common concepts identified in existing requirements modelling approaches." These "existing requirements modelling approaches" include goal oriented and of course SySML.

Although modelling and, more specifically, meta-modelling is very useful as described above, it is by definition not reality, and this can lead to relationships between elements of a system being allowed to deviate from a true representation of the desired system. This is recognised and mathematical foundations are offered as a possible solution (Fogarty and Austin 2009).

Mathematical Treatments. We briefly touch upon current uses of mathematical treatments ("formal methods") within requirements engineering research. Logical mathematical treatment or modelling is of benefit as it assists in ordering relationships in a structured, precise and non-ambiguous manner. This is in contrast to natural language usage, as typified within classical requirements specifications, where concepts are often defined in terms of each other, in a circular manner. A presented analogy is the dictionary where "words are always defined in terms of other words, which can lead to definitions directly or indirectly referring back to themselves." (Dickerson 2008)

Within the context of requirements engineering, set-theoretic and logical semantics are used to define rules and structure between requirement relationships and attributes. Typical expressions use first order logics, semi-lattices to order sets of requirements and event calculus, which is particularly useful in capturing temporal impacts of discrete events. Precise semantics are a pre-requisite for tool support.

In this section we have briefly discussed some of the different representation approaches to requirements engineering. We will now bring together some of the relevant points we have raised in this and the previous section, with the aim of suggesting possible avenues of interest regarding requirements engineering of SoS.

4. REQUIREMENTS IN SOS

Some Concepts. In Section 2 we identified some questions which highlight some of the issues faced in improving requirements engineering of SoS. In this section we present a preliminary review of those questions in the context of the self-x management model initially discussed in the Introduction. This will only be a first step along the path of further studying our ideas to formulate solutions to these questions.

We will first discuss question (1), together with the associated questions (a) and (b), which are repeated here for reference.

Question (1). In what ways might we imbue SoS with the ability to accurately capture, within goals and requirements, evolving SoS needs?

(a) As an evolving SoS generates new needs, which in turn drive new global goals, what structures are required to facilitate collaboration of goals and requirements between SoS elements and, where applicable, central authorities?

(b) What methodologies will enable this collaboration to occur in a timely manner, and additionally, in an optimal manner?

In these questions, we are not concerned with the physical pathway or communications protocol, although these are equally challenging in their own right. Our interest lies in considering the possible ways of structuring the goals and requirements information for collaboration. What defines or limits the goal and requirement information we need to share between SoS elements? Would the structure be dependent on whether collaboration was between two SoS elements, as opposed to an SoS element and the central authority?

We believe Maier's concept of managerial and operational independence needs to exist where goal and requirement changes will be detected and can be managed promptly, and as necessary. We noted previously that the concept of managerial and operational independence is related to the SoS characteristic of autonomy. We suggest this is where self-x requirements engineering capabilities may offer some benefits to the SoS issues underlying the above questions. Autonomy is the essential characteristic of the autonomic computing model shown in Figure 1. In Figure 2 below, we apply the intent of Figure 1 to a proposed 'autonomic requirements manager'.

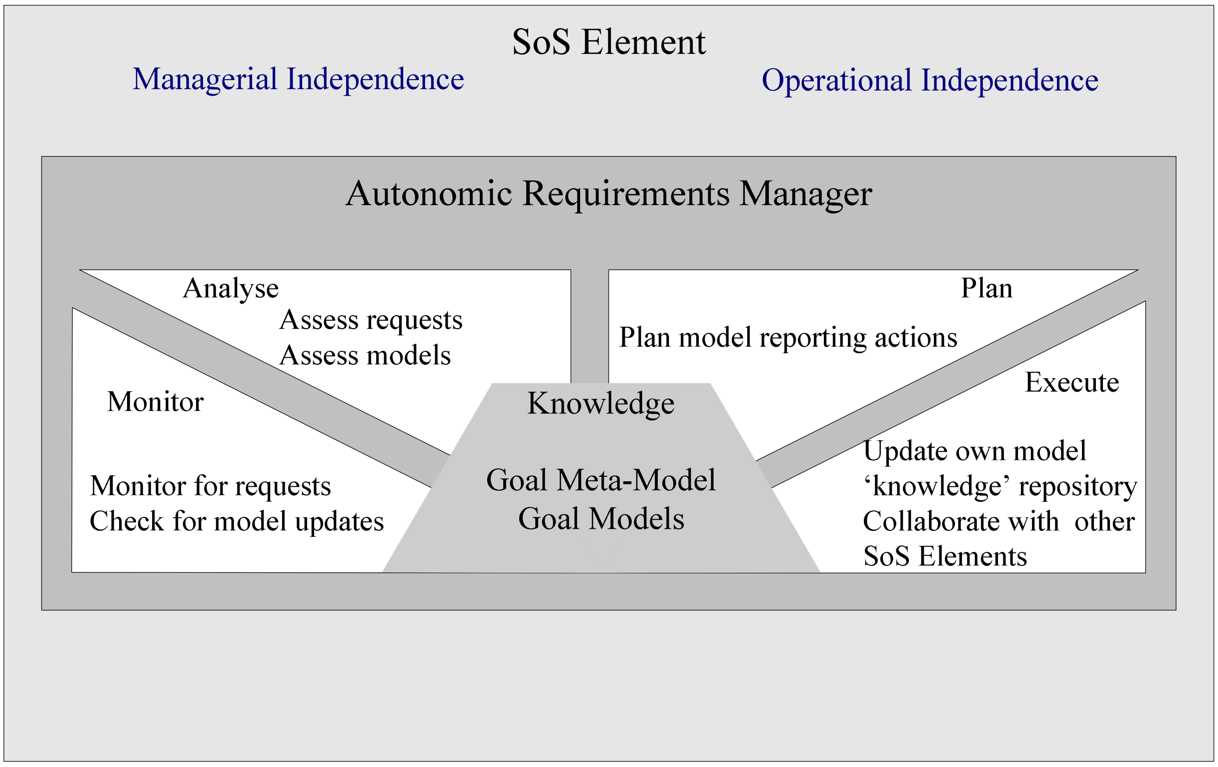

Figure 2. Autonomic Requirements Manager

We discuss Figure 2 by way of example.

Imagine, if you will, an Unmanned Aerial System (UAS) with radar and infra-red imaging capabilities. This UAS is managed and operated by an air platform organisation, and forms part of a border protection SoS. The UAS would conform to our definition of an SoS element, as detailed previously in the paper. Other SoS elements could include a national coordination centre, which performs the role of an 'SoS central authority' as mentioned in Table 1; ground patrol personnel within various organizations including police, emergency services and customs; health systems for medical checks and maintenance of medical records; and, sea going systems managed and operated by the Navy and Coastguard.

The goals assigned to the UAS may take the form of models, where each goal model is an instance of the goal meta-model. More goals would exist for other components of the air platform organisation. These models may be thought of as the 'knowledge' component within the autonomic models for each SoS element. The SoS element 'autonomic requirements manager' continually performs the Monitor-Analyse-Plan-Execute cycle. As depicted in Figure 2, self-configuration of the air platform organisation goals and requirements would occur through operation of the 'autonomic requirements manager'. Furthermore, we would propose that the contents of the 'knowledge' block imbues the SoS element with the characteristic of self-knowledge. The collection of goal and requirement models provides a description of the SoS element contribution to global SoS needs.

Figure 3 extends the concept introduced in Figure 2 to a number of SoS elements and illustrates some of the collaboration exchanges which aim to address our posed questions.

The ability of the SoS to collaborate on such issues as goal decomposition and allocation, via the autonomous sharing and management of goal models, brings us closer to realizing self-configuration of the SoS. Additionally, because this process is significantly performed through these model driven engineering methods, we would expect new goals to be processed in a quicker time by the SoS, compared to current methodologies which rely predominantly on human assessment.

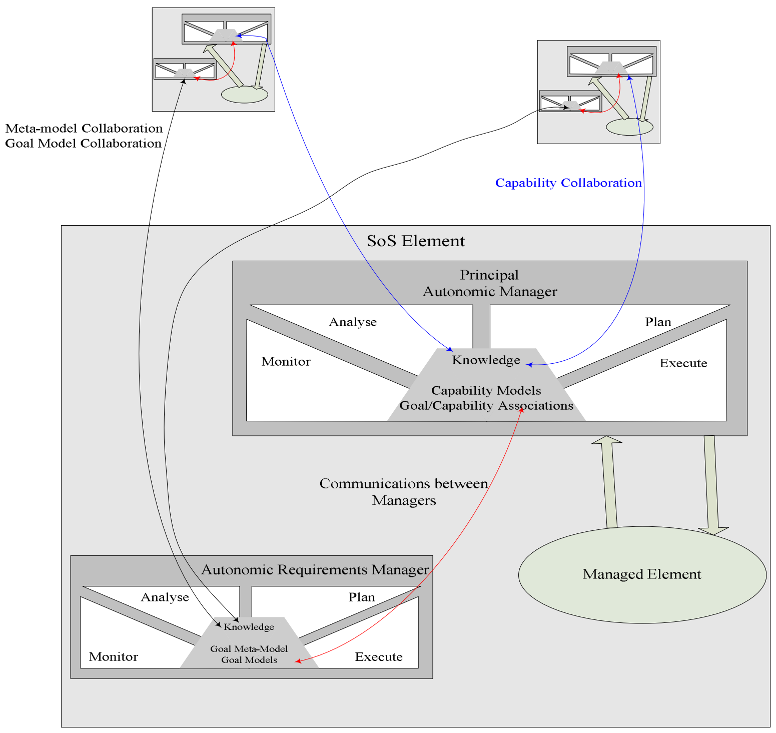

Figure 3. SoS Element Collaboration

We now turn to discuss question (2), together with the associated questions (c) and (d) from Section 2, which are also repeated here for reference.

Question (2). How may we improve means of mapping the goals and requirements to the realised SoS implementation?

(c) Which roles may requirements engineering play in SoS assessment of extant implementation capabilities against evolving needs?

(d) How may we structure SoS Element-to-SoS Element collaboration so that it will inform the SoS of additional functionality required to achieve desired global goals?

If we return to our example, the case may exist where the UAS infra-red capability was not implemented in the initial fielding of the equipment. However, due to increases in nocturnal border incidents, the national coordination centre determines that an infra-red capability within the SoS is a new goal.

In Figure 3, we show a 'principal autonomic manager' and a 'autonomic requirements manager'. The 'principal autonomic manager' represents the core manager described by (Kephart and Chess 2003) in Figure 1, with the interfaces to the physical components; in our example the hardware and software interfaces of the UAS. We suggest the 'principal autonomic manager' contains a library of capability models, as well as goal-to-capability associations. These associations give a possible insight into how self-assessment by SoS elements may assist the SoS in determining whether an extant capability is available to satisfy some new goal, or whether new SoS elements or components may be necessary. In our example, the UAS SoS element would determine, through communications between the 'autonomic requirements manager' and the 'principal autonomic manager', as well as via collaboration with other SoS element 'principal autonomic managers', that its infra-red capability would meet the SoS goal. The allocated goal model in the UAS 'autonomic requirements manager' would be associated with the infra-red capability model in the UAS 'principal autonomic manager'.

Awareness of options within the SoS, for a given global goal implementation would allow for optimal self-organisation of the SoS. The capabilities of an infra-red system on a sea-based SoS element may be superior to the UAS infra-red capability in our border protection SoS. Hence, self-optimising requirements engineering behaviour may be shown by the choice of SoS element goal allocation.

Our discussion in this section is preliminary and has the purpose only of highlighting how self-x requirement engineering behaviour may assist in addressing certain SoS issues. This paper serves to prompt further questions, and illustrates the effort still to be undertaken. How, for example, would a non-functional requirement fit into this self-x requirements engineering paradigm? The structure of the necessary goal or requirement meta-model may hold the key, but questions like this require further consideration. It is worth noting that discussion also highlights other related issues, such as configuration management of implemented SoS, that also require consideration. Perhaps, in time, the UAS, like all SoS elements, will be a Commercial-Off-The-Shelf (COTS) item with the capability models and autonomic goal interfaces as part of the baseline purchase, pre-installed into the software and hardware. Will we see, as with other interfaces, the goal meta-model, capability models and autonomic interfaces being defined within agreed international standards?

5. FUTURE RESEARCH

Future Work. We intend to continue our research into understanding the requirements engineering issues of SoS. This research will based around the development of a goal meta-model which enables self-x goal and requirement reasoning behaviour. There are other areas of interest to us which we plan to research, including supply chain management and configuration management, where we believe similar issues and questions to those raised in SoS, may exist.

An example SoS will be chosen to investigate validity of the proposed goal meta-model.

6. SUMMARY & CONCLUSION

In this paper we have undertaken some preliminary steps in our journey of discovering whether SoS can be given self-x requirements engineering capabilities. We introduced the autonomic computing architecture and discussed its structure. Additionally, we posed a couple of central questions related to the challenges of goal and requirement engineering within SoS. Following this, we proceeded to give a comprehensive review of SoS definitions, highlighting the many existing in the literature. We have offered our own definition, which includes a number of terms that we believe add value to the ongoing discussion. It was noted in our literature survey that there does not appear, at this time, to be any internationally recognised standard for SoS.

In our section on characteristics of SoS, we showed how Maier’s operational and managerial independence has been associated with the characteristic of autonomy. We suggested that autonomy is, in turn, related to the evolutionary characteristic of SoS, in as far as autonomous behaviour of SoS elements is such that the evolutionary behaviour of the SoS is towards its nominated goals. This lead to discussion around the collaboration classification of SoS. We reviewed the four classification types given in the literature; directed, acknowledged, collaborative and virtual. Then we suggested a breakdown of requirements engineering characteristic by these four SoS classification types and, considered our initial SoS goal and requirements engineering questions in the context of this breakdown.

After a brief review of some different requirements representations being presently researched and applied within SoS, particularly the use of model driven engineering approaches, we proceeded to a discussion on how we suggest the use of an autonomic goal management model may enable self-x goal and requirements engineering behaviour. We posited how this behaviour may assist in meeting the SoS challenges given in our central questions of the paper. However, we demonstrated that there are further questions and research efforts which need to be considered on our journey.

REFERENCES

Alexander, I. F. and R. Stevens (2002). Writing Better Requirements, Pearson Education Ltd.

Alrajeh, D., O. Ray, et al. (2007). "Extracting requirements from scenarios with ILP." Inductive Logic Programming **4455**: 64-78.

Bjelkemyr, M., D. Semere, et al. (2007). "An engineering systems perspective on system of systems methodology." 2007 1st Annual IEEE Systems Conference: 185-191.

Boxer, P. J. and S. Garcia (2009). Enterprise architecture for complex system-of-systems contexts. 2009 IEEE International Systems Conference Proceedings, Vancouver, BC.

Brottier, E., B. Baudry, et al. (2007). "Producing a global requirements model from multiple requirement specifications." 11th IEEE International Enterprise Distributed Object Computing Conference, Proceedings: 390-401.

Clark, J. O. (2009). System of Systems Engineering and Family of Systems Engineering From a Standards, V-Model, and Dual-V Model Perspective. New York, IEEE.

Dallal-Shwartz, I., G. Rabinowitz, et al. (2009). "Fan-out" model and methodology for "system of systems" development. 2009 IEEE International Systems Conference Proceedings, Vancouver, BC.

Dardenne, A., A. Van Lamsweerde, et al. (1993). "GOAL-DIRECTED REQUIREMENTS ACQUISITION." Science of Computer Programming **20**(1-2): 3-50.

DeLaurentis, D. A. (2008). "Appropriate modeling and analysis for systems of systems: Case study synopses using a taxonomy." 2008 IEEE International Conference on System of Systems Engineering, SoSE 2008.

Dickerson, C. E. (2008). "Towards a logical and scientific foundation for system concepts, principles, and terminology." 2008 IEEE International Conference on System of Systems Engineering, SoSE 2008.

DiMario, M. J. (2006). "System of systems interoperability types and characteristics in joint command and control." Proceedings of the 2006 IEEE/SMC International Conference on System of Systems Engineering: 222-227.

DoD (2008). Systems Engineering Guide for Systems of Systems. S. a. S.

Engineering. Washington, DC: ODUSD(A&T)SSE, Office of the Deputy Under Secretary of Defense for Acquisition and Technology.

Doyle, R. P., D. L. Kaminsky, et al. (2005). Autonomic computer system managing system, has processor to transit management of managed system from manual control of administrator to autonomic control of adaptive process, when certain level of trust is built, Int Business Machines Corp; IBM Corp.

Fogarty, K. and M. Austin (2009). "System Modeling and Traceability Applications of the Higraph Formalism." Systems Engineering **12**(2): 117-140.

Goknil, A., I. Kurtev, et al. (2008). "A metamodeling approach for reasoning about requirements." Model Driven Architecture - Foundations and Applications, Proceedings **5095**: 310-325.

Hata, Y., Y. Kamozaki, et al. (2007). "A heart pulse monitoring system by air pressure and ultrasonic sensor systems." 2007 IEEE International Conference on System of Systems Engineering, Vols 1 and 2: 104-108.

Hata, Y., S. Kobashi, et al. (2009). "Human health care system of systems." IEEE Systems Journal **3**(2): 231-238.

Jamshidi, M. (2009). Systems of Systems Engineering: Principles and Applications, CRC Press.

Keating, C. B., J. J. Padilla, et al. (2008). "System of Systems Engineering Requirements: Challenges and Guidelines." Engineering Management Journal **20**(4): 24-31.

Kephart, J. O. and D. M. Chess (2003). "The vision of autonomic computing." Computer **36**(1): 41-+.

Letier, E. (2001). Reasoning about Agents in Goal-Oriented Requirements Engineering. Departement d'Ingenierie Informatique, Universite Catholique de Louvain. **PhD Thesis**.

Lewis, G., E. Morris, et al. (2008). "Engineering systems of systems." 2008 2nd Annual IEEE Systems Conference: 403-408.

Lewis, G. A., E. Morris, et al. (2009). Requirements engineering for systems of systems. 2009 IEEE International Systems Conference Proceedings, Vancouver, BC.

Maier, M. W. (1998). "Architecting principles for systems-of-systems." Systems Engineering **1**(4): 267-284.

Markose, S. M. (2005). "Computability and evolotionary complexity: markets as complex adaptive systems (CAS)." The Economic Journal **115**(504): F159-F192.

Murch, R. (2004). Autonomic Computing, Prentice Hall.

Mylopoulos, J., L. Chung, et al. (1999). "From object-oriented to goal-oriented requirements analysis." Communications of the ACM **42**(1): 31-37.

Navarro, E., J. A. Mocholi, et al. (2006). "A metamodeling approach for requirements specification." Journal of Computer Information Systems **46**: 67-77.

OMG. (Cited 2010). "SySML Specification." from http://www.omgsysml.org/.

OMG. (Cited 2010). "UML Specification." from http://www.uml.org/.

Sauser, B. and J. Boardman (2008). "Taking hold of system of systems management." EMJ - Engineering Management Journal **20**(4): 3-8.

Seebach, H., F. Ortmeier, et al. (2007). Design and construction of organic computing systems. 2007 IEEE Congress on Evolutionary Computation, Vols 1-10, Proceedings. New York, IEEE**:** 4215-4221.

Sheard, S. A. and A. Mostashari (2009). "Principles of Complex Systems for Systems Engineering." Systems Engineering **12**(4): 295-311.

Simpson, J. J. and C. H. Dagli (2008). "System of systems: Power and paradox." 2008 IEEE International Conference on System of Systems Engineering, SoSE 2008.

Weber, M. and J. Weisbrod (2002). "Requirements engineering in automotive development - Experiences and challenges." IEEE Joint International Conference on Requirements Engineering, Proceedings: 331-340.

White, S. R., J. E. Hanson, et al. (2004). An architectural approach to autonomic computing. Los Alamitos, IEEE Computer Soc.

Wickramasinghe, N., S. Chalasani, et al. (2007). "Healthcare system of systems." 2007 IEEE International Conference on System of Systems Engineering, Vols 1 and 2: 312-317.

BIOGRAPHY

Alisdair MacDiarmid is currently undertaking a PhD in Systems Engineering at the University of Queensland. His present research interests include the application of requirements engineering methods to complex systems, including System of Systems (SoS); autonomous systems; and, formal methods. Alisdair has extensive experience as a practicing systems engineer, gained from involvement on defence projects and from work performed in the commercial aerospace industry.

Dr Peter Lindsay is Boeing Professor of Systems Engineering at the University of Queensland. He has held academic and research positions at the University of NSW, the University of Manchester and the University of Illinois at Urbana-Champaign. He is co-author of two books on formal specification and verification of software systems. He has been involved with safety and security critical applications in areas such as air traffic control, embedded medical devices, ship-board defence, emergency service dispatch systems, and an international diplomatic network. His current research interests include engineering of complex systems, safety-critical systems, and formal methods of system development.