Grant Agreement: 287829

Comprehensive Modelling for Advanced Systems of Systems

Modelling Patterns for Systems of Systems Architectures

COMPASS White Paper WP05

Date: October 2013

Public Document

http://www.compass-research.eu

Contributors:

Claire Ingram, UNEW
Richard Payne, UNEW
Simon Perry, Atego
Jon Holt, Atego
Finn Overgaard Hansen, IHA
Luis D Couto, IHA

Abstract

This white paper is an initial report on modelling patterns and architectures for system of systems (SoSs) and their constituent systems (CSs). We explain the background to architectural styles or patterns and introduce an initial set of five architectural patterns suitable for SoS design. SoS engineering presents some particular challenges for the architect, which should be taken into account. Each pattern that we introduce is illustrated with an SoS example, and we also identify the aims of the pattern.

The work outlined in this white paper is part of a tranche of work examining architectural design in SoS. This white paper presents a subset of research carried out on the COMPASS[1] project into model-based techniques suitable for SoSE. Further details of architectural patterns can be found in [PHP+13].

[1]http://www.compass-research.eu/

Contents

1 Introduction

The concept of an architecture is fundamental to any systems engineering undertaking. We define architecture as 'the structure of components, their relationships, and the principles and guidelines governing their design evolution over time' (taken from IEEE Std 610.12 and DoDAF[2]). Reusing an architecture 'allows projects to quickly identify conceptually similar existing architectures and quickly interpret them to the chosen application.' [DM09]. Studying architectural patterns can be helpful for supporting re-use of architectures, as well as learning from the common successes and challenges that a particular architectural pattern might accompany. For this reason we have begun the process of studying existing systems of systems (SoSs), and cataloguing some common architectural patterns suitable for SoS engineering (SoSE).

A pattern has been defined as 'an idea that has been useful in one practical context and will probably be useful in others' [Fow97]. Architectural styles, design and patterns overlap significantly, but *design patterns* (e.g., see [GHJV95]) and *architectural styles* are distinct. An architectural view provides a high-level view that enables the analysis of 'emergent system wide properties' [MKMG97], whilst design patterns are concerned much more with lower-level questions.

2 Architectural principles for SoS

SoSs present particular challenges for the SoS architect, including:

- Accurately predicting and accounting for all the possible emergent behaviours is prohibitively time-consuming, or is not possible due to lack of information disclosure by constituent systems (CSs).
- There are very long lifecycles and the presence of legacy or off-the-shelf components that cannot be adapted to enable optimal solutions.
- CSs have other pressures to evolve outside the SoS.
- There is a high degree of technical and managerial complexity.
- There is often no central decision-making authority.

[2]http://dodcio.defense.gov/dodaf20.aspx

- SoS boundaries are blurred - in some cases it is not clear which systems might be considered constituents and which are part of the environment.
- SoSs typically span multiple domains, which makes misunderstandings and functional gaps more likely.
- Socio-technical issues can complicate the architecture and design.

Architectural patterns suitable for SoS should provide strategies for coping with these issues. An architecture that supports loose coupling of constituents allows constituents to evolve separately and reduces the risk that a change will propagate should a constituent system implement some changes or depart the SoS suddenly. Patterns suitable for SoSs should also ensure that it is easy to substitute CSs when necessary - for example, replacing one constituent which is currently compromised or uncontactable with another constituent that can deliver an equivalent service. When this occurs at runtime it is referred to as dynamic reconfiguration.

Our architectural patterns have their roots in the study of architectural styles. An architectural style 'defines a family of such systems in terms of a pattern of structural organization' [GS94]. Identifying the architectural style of a system facilitates reasoning and understanding about a system's design [Gar00]. System designers can benefit from 'lessons learned' on similar systems and can reuse standard frameworks if a recognised architectural style is adopted. Numerous architectural styles have been documented (e.g., see [GS94, MKMG97]).

The fields of architectural styles, design and patterns overlap significantly. However, *design patterns* and *architectural styles* are distinct. An architectural style is a high-level view [MKMG97] whilst design patterns tend to be concerned with lower-level questions. Some architectural styles have been proposed for distributed systems (for example, see [Wei97]), which may be adaptable for SoSE. We build on the principle to develop the notion of architectural patterns for SoS in Section 3.

3 Initial SoS architectural modelling patterns

We introduce here our initial collection of SoS architectural modelling patterns. Following the example of [GHJV95], we describe a pattern by identifying the following basic properties: background; aims of the pattern; and an illustrative example.

3.1 Centralised Architecture Pattern

Background A centralised architecture has a central point of control. The central CS (the 'hub') is connected to the other CSs and is responsible for ensuring SoS behaviour. There may be degrees of centralisation; e.g., a fully centralised SoS will connect all CSs to a hub, whilst a partially-centralised SoS will see a hub connected to a subset of CSs. Constituents are still capable of exhibiting autonomy despite the centralisation. For example, the hub can make decisions about functionality, whilst CSs may be unaware of the SoS (e.g., they may be commercial off-the-shelf systems) and their ability to make autonomous decisions continues unabated.

We distinguish between a centralised architecture with a hub, and a CS which is commonly employed by several other CSs to provide a service; the latter enacts no (or little) control and does not address the SoS goals. A hub may connect to a CS which is itself considered a mini-hub in a sub-SoS, controlling/managing another collection of constituents. In a hybrid centralised-distributed SoS, the central hub may be distributed over several constituents.

Aims The main aims of this pattern are to support:

- Centralised control and management of SoS. This is achieved through the use of a hub specifically created for the SoS.
- Reuse of pre-existing systems. This is achieved, as the custom-designed hub can interoperate with existing or off-the-shelf systems.

Example Examples of a centralised SoS often inhabit a domain with a strong requirement for 'command and control'. The anti-guerrilla operations SoS example, described in [HMK06], has a central 'theatre command' system (with a strong human aspect), a system comprised of UAV scouts, and CSs including artillery, troops and the required communication infrastructures. The theatre command system makes operational decisions based upon data sourced from UAV scouts and other sources, to give commands to the various troops and artillery. The goals of the SoS are achieved due to the commands of the central hub, which takes responsibility for delivering SoS functionality.

3.2 Service Oriented Architecture

Background Employing a Service Oriented Architecture (SOA), applications are constructed through the use of third-party services. Services are stateless (i.e., they have internal state, but do not share it), behaving like functions acting on supplied parameters. Service providers may develop systems in any way as long as they provide standardised descriptions and expose a means for provision of the service.

The SoS CSs expose their interfaces, which act as points of interactions between constituents, and conform to a specified protocol, subject to a security policy. Services publish a service contract constituting a service description with details of functionality, and a service-level agreement that details quality of service (QoS) guarantees. The pattern requires two separate notions of an interface: a CS interface and a service interface. An SoS designed with the SOA pattern is centralized in nature, because a single system is involved with selection of services, and can be considered to be directed [Mai98] or acknowledged [DB08].

Aims The main aims of this pattern are to support:

- Analysis of SoS emergent behaviour. This can be achieved through the analysis of the service descriptions.
- SoS/CS evolution. This is enabled through the loose coupling of SoS management; service providers need not know how a service is being used, only that they must meet the guarantees made in the service contract.
- Central SoS authority. A central system is used to compose services to achieve the SoS goals and functionality.
- Cross-domain SoS development. This is realized through the separation of the service contract and the underlying service implementation. A service consumer may use a service without requiring knowledge of the service logic or implementation.
- Long SoS lifecycle. Explicit separation of service interface and service implementation allows CS developers to apply different methodologies and development processes, enabling a long lifecycle.

Example A travel agent booking SoS has a central front-end system that receives requests from its environment (either a consumer or travel agent) to book a trip consisting of a hotel, flight, etc (shown in Figure 1). The

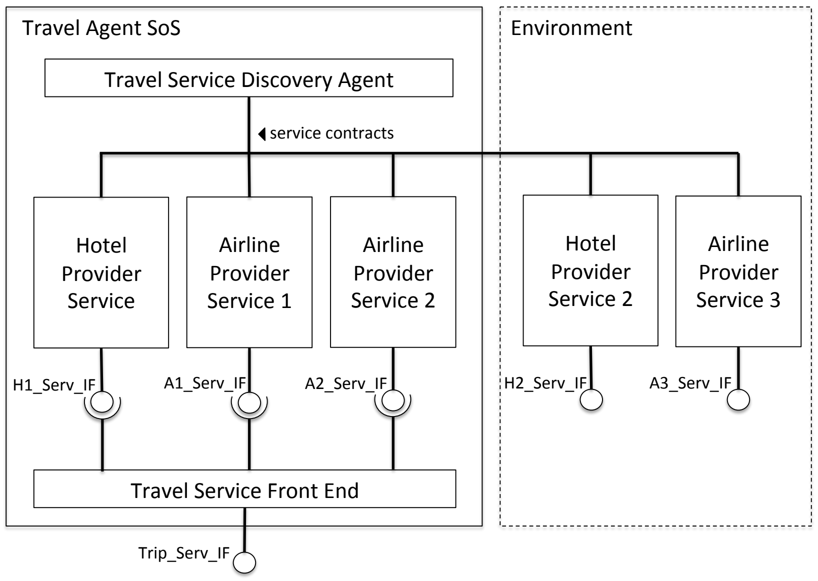

Figure 1: Travel agency SoS implementing an SOA architectural pattern

front-end system delivers the SoS functionality of receiving trip requests and responding with trip details, in the process employing those services provided by other third-party systems. A discovery system retrieves service contracts from service providers. The front end may change service providers dynamically by selecting different suppliers. The loose coupling of the SOA pattern ensures that the SoS is amenable to such reconfigurations.

3.3 Publish-Subscribe Architecture Pattern

Background This pattern can be divided in two different subgroups: an Event-Based Publish-Subscribe pattern (EBPS) and a Data-Centric Publish-Subscribe (DCPS) pattern. We focus at the moment on the data-centric publish-subscribe version.

'Data Distribution Service for Real-Time Systems' (DDS) is a standard published by OMG which specifies a Data-Centric Publish-Subscribe (DCPS) model. DCPS provides the functionality required for an application to publish and subscribe to the values of data objects of given types. In this pattern, a Topic describes a Data-Object with a unique name in the given domain, a

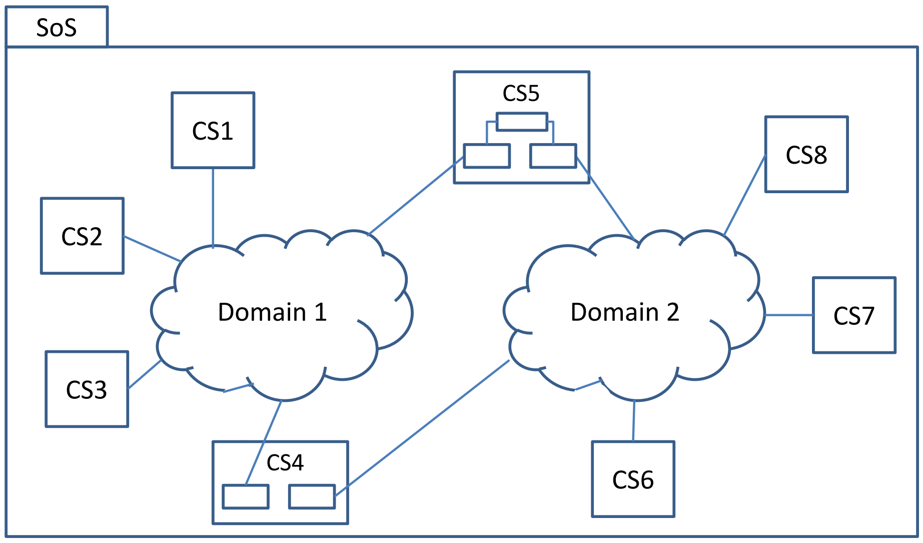

Figure 2: Example of a publish-subscribe model with two domains

data-type and a set of Quality of Services (QoS) related to the data. A Publisher is responsible for data distribution to a set of registered Subscribers and publishes data on one or more Topics. A DataWriter is used by the Publisher to publish data associated with a unique Topic. A Subscriber receives published data on Topics and makes these available to the application at the Subscriber site. A DataReader is used to access the received data from the attached Subscriber. Both a DataWriter and a DataReader have typed interfaces for a given topic, acting as a mediator on the publishing site to the publisher and on the receiver site to the subscriber.

Each CS in this type of SoS can play the role of a Publisher, a Subscriber, or both, on one or more Topics. A given SoS can be defined as one or more communication domains each with its own set of CSs, as indicated on Figure 2. In this example, constituents CS1-CS5 participate in domain 1 and constituents CS4-CS8 in domain 2. CS4 and CS5 participate in both domains, CS5 provides a potential gateway between the two domains if needed, whilst CS4 participates in the two domains without interactions between the domains. This pattern can be useful for a collaborative [Mai98] SoS; each CS agrees on a common data model, comprising a set of topics for exchanging data. If a CS Subscriber wants to join or leave a given communication domain, it can register or de-register on a given Topic in the domain.

Aims The main aims of this (DCPS) pattern are to support:

- Loose coupling between publisher and subscriber CSs in the SoS with time, flow and space decoupling between the CSs. Time decoupling is achieved as the Publisher and Subscribers do not need to be online at the same time (the middleware system can store data). Flow decoupling is achieved, as a publisher can publish its data asynchronously without waiting for a subscriber to receive it, whilst a subscriber does not need to wait but can be notified asynchronously when a change in subscribed data occurs. Space decoupling is achieved as neither the publisher or the subscriber needs to know each other's identity.
- One to many and many to many communications between the CSs in the SoS. This is also achieved, as there can be one or more publishers on the same topic and one to many subscribers on the same topic.

Example A Medicine Card topic contains the actual medicine prescriptions for a given citizen along with the history of prescriptions. Several CSs can update this information as Publishers to the Medicine Card information whilst other CSs subscribe to the Medicine Card information and receive updates when the medicine prescription changes for a specific citizen. The underlying interaction mechanism is a push-mechanism, where changed information is pushed to all the registered subscribers. This pattern has a very loose coupling between publishers and subscribers, where it is very simple to add new publishers or subscribers.

3.4 Pipes and Filters Architecture Pattern

Background This pattern includes Filters connected serially by Pipes. Filters represents the processing steps, where data (or materials) are processed from one input form to an output form along a serial processing line. Pipes represents connections between Filters for transferring the data. There is an Input Source, representing the first step where data enters the SoS, and also an Output Sink that indicates where data exits. Filters are independent and do not share state or know each other's identities.

This pattern can be applied for either directed [Mai98] or acknowledged [DB08] SoS, and in principle could also be used for collaborative [Mai98] SoS if a CS joins a processing chain to provide additional services.

Aims The main aims of this pattern are to support:

- Independent processing steps on a flow, realized through the series of independent CSs.

- Configurable transmission of the flow between processing elements. This is achieved through the use of CSs as ‘pipes’.
- Dynamic change of processing steps and connectors. This is achieved as both filters and pipes can be exchanged at design time, or sometimes dynamically during runtime.

Example A Local Monitoring system monitors a patient’s vital signs in a local setting, with the possibility of storing and/or displaying results and giving local alarms. The signals monitored are forwarded through a Pipe constituent system to a Central Monitoring system, for further analysis (e.g., monitoring by a doctor). The Central Monitoring CS receives inputs from many other CSs. The processed signals can be forwarded to another CS performing the role of a Pipe for Central Storage. In some situations Pipe CSs could be implemented by standard middleware, thus disappearing as independent CSs. This architecture ensures that the CS are decoupled, as the receivers of a given flow can be redirected dynamically to another Filter component.

3.5 Blackboard Architecture Pattern

Background The Blackboard architecture pattern is suitable for SoSs solving problems with a certain degree of uncertainty, e.g. in expert based or fuzzy logic based systems, or where central knowledge is obtained from several sources. The components are the Knowledge Sources, a Blackboard data structure and a Control component. The Blackboard component is a central data store with an interface for reading and writing data. Elements of the solution space are written to (or removed from) the Blackboard by the Knowledge Sources. Knowledge Source CSs are specialized for either solving a part of the overall problem or delivering input data; they work independently and usually in parallel. Each has a condition part, that evaluates the Blackboard state to see if it can make a contribution, and an action part, that produces a result and updates the blackboard state.

The Control component evaluates the current state of the blackboard and uses this data to coordinate the Knowledge Sources. The Control component searches for a possible solution to the problem. This pattern is applicable for directed [Mai98], acknowledged [DB08] and possibly also for collaborative [Mai98] SoSs.

Aims The main aims of this pattern are to support:

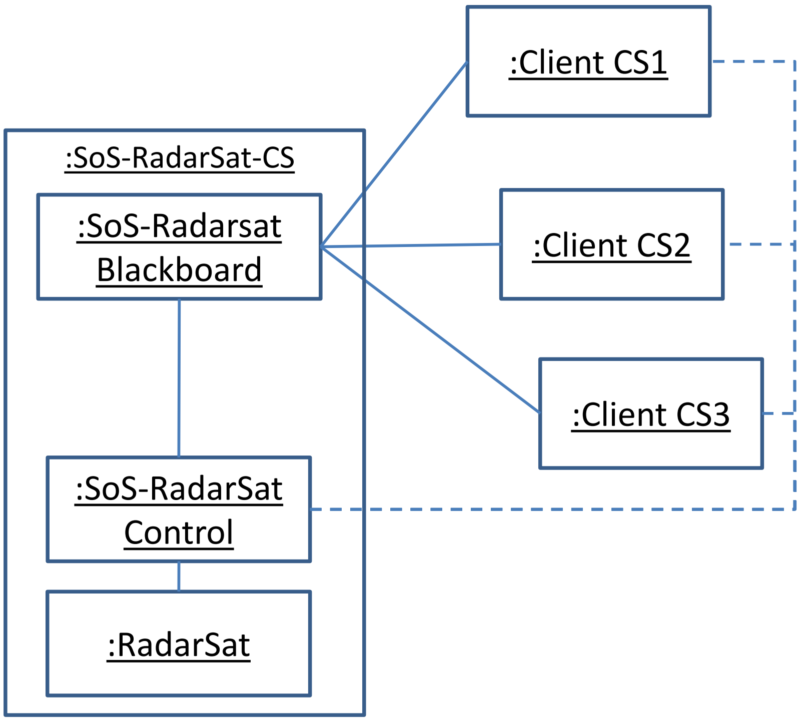

Figure 3: A satellite SoS implementing a blackboard pattern

- Development of expert or knowledge based systems. The Blackboard component houses the 'expert' constituent system, where the information and hypotheses are stored and modified by the Knowledge Source constituent systems.
- Loose coupling. This is achieved through the independence of the Knowledge Sources, which may be unaware of the existence of other sources.
- Separation of concerns. This is achieved through the use of three different system roles. A Knowledge Source is free to pursue its own goals outside the SoS, including in other SoSs.

Example RadarSat-1 (based on a real-life example called RadarSat-1 [Cor97]) is an earth-observation satellite (shown in Figure 3), equipped with an aperture radar to allow end users to connect, submit requests and receive the results. A blackboard is used to realize an advanced planning component, which controls the radar measurements. This system has over 140 constraints, which makes the planning process very complex. The SoS-RadarSat-CS is a CS and controls the access and use of the satellite radar, which is a shared resource in the community of RadarSat users. Each Client-CS in this SoS is an independent system with its own purpose that participates in the SoS as a Knowledge Source. RadarSat SoS allows each Client CS to perform its own measurements and experiments with the possibility of sharing the results in the community.

4 Conclusions and future work

In our future work we would like to consider patterns related to the evolution and dynamic behaviour of SoSs, addressing:

- CSs dynamically entering or leaving the SoS, resulting in the reconfiguration of the SoS and new emergent properties at the SoS boundary.
- Dynamic contract negotiations. This may be required at the formation of a SoS, or over its lifetime.
- Control structures required for enacting system changes (either behavioural or structural) based on the state of the SoS and its environment.

References

[Cor97] Daniel D. Corkill. Countdown to Success: Dynamic Objects, GBB, and RADARSAT-1. *Communications of the ACM*, 40(5):49–58, May 1997.

[DB08] Judith Dahmann and Kristen Baldwin. Understanding the Current State of US Defense Systems of Systems and the Implications for Systems Engineering. In *IEEE Systems Conference*. IEEE, April 2008.

[DM09] Charles Dickerson and Dimitri N. Mavris. *Architecture and Principles of Systems Engineering*. CRC Press, 2009.

[Fow97] M. Fowler. *Analysis Patterns: Reusable Object Models*. Addison-Wesley, 1997.

[Gar00] David Garlan. Software architecture: a roadmap. In *Proceedings of the Conference on the Future of Software Engineering: ICSE00*, pages 91–101, 2000.

[GHJV95] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns Elements of Reusable Object Oriented Software*. Addison Wesley, 1995.

[GS94] David Garlan and Mary Shaw. An introduction to software architecture. Technical Report CMU/SEI-94-TR-21, Software Engineering Institute, Carnegie Mellon University, 1994.

[HMK06] M. Hall-May and T. P. Kelly. Using agent-based modelling approaches to support the development of safety policy for systems of systems. In J. Gorski, editor, *Proceedings of the 25th International Conference on Computer Safety, Reliability and Security (SAFECOMP 06)*, volume 4166 of *LNCS*, pages 330–343, Sep 2006.

[Mai98] Mark W. Maier. Architecting Principles for Systems-of-Systems. *Systems Engineering*, 1(4):267–284, 1998.

[MKMG97] R.T. Monroe, A. Kompanek, R. Metlon, and D. Garlan. Architectural styles, design patterns, and objects. *IEEE Software*, 14(1):43–52, 1997.

[PHP$^+$13] Simon Perry, Jon Holt, Richard Payne, Claire Ingram, Alvaro Miyazawa, Finn Overgaard Hansen, Luis D. Couto, Stefan

Hallerstede, Anders Kaels Malmos, Juliano Iyoda, Marcio Cornelio, and Jan Peleska. Report on Modelling Patterns for SoS Architectures. Technical Report D22.3, COMPASS, February 2013.

[Wei97] Charles Weir. Architectural styles for distribution: Using macro-patterns for system design. In *Proceedings of the 1997 European Pattern Languages of Programming Conference, Irsee. Available as Siemens Technical Report 120/SW1/FB*, 1997.