

Facilitating System-of-Systems Evolution With Architecture Support

Pin Chen

DSTO C3 Research Centre
Department of Defence
Canberra, ACT 2600, Australia

Email: Pin.Chen@dsto.defence.gov.au

Jun Han

School of Network Computing
Monash University
Melbourne, Vic. 3199, Australia

Email: jhan@monash.edu.au

ABSTRACT

The evolution of system-of-systems (SOS) is an emerging challenge and requires systematic architecture capabilities and support. This paper discusses the features of SOS evolution and introduces a key concept, *Architecture Evolution Environment*, to facilitate the evolution. We argue that an architecture solution for specific evolution requirements can be reached only when its evolution environment is addressed.

Keywords: Architecture, system evolution, architecture interfaces and evolution environments.

1. INTRODUCTION

With technology advances and changes in business needs, the evolution of the enterprise information system that in most cases is a System of Systems (SOS) is unavoidable and has become one of the main challenges facing a large organization. Research efforts addressing a single system's evolution and in particular software evolution have produced a range of methods and techniques. Due to the immaturity of practice and the increasing complexity of SOS, however, there have been obvious needs to develop systematic methodologies that enable evolution of such systems.

A system evolution occurs when any of the following three basic situations happen: 1) *Self-Evolution*: a change is introduced into a system as a consequence of redesign, redevelopment, modification or improvement, i.e., the evolution of a single system; 2) *Joint Evolution*: two or more systems are to be integrated for improved business support; or 3) *Emergent Evolution*: a new system is to be developed on the basis of or in relation to existing systems with new functionalities or capabilities. Some complicated evolution could involve a combination of the above situations. In addition, the related systems in a SOS context may evolve simultaneously.

It is important to distinguish a stand-alone system's evolution from SOS evolution and to treat software evolution as part of the system evolution in order to develop proper methodologies and techniques. This paper explores a key concept, *Architecture Evolution Environment (AEE)*, which can help clarify architecture issues related to system evolution in a SOS context and facilitate architecture design tasks to reach an optimized evolution solution.

2. RELATED WORK

Most existing research and development efforts relevant to SOS evolution are made individually in many different areas including: software architecture, business architecture, system interoperability, component-based architecture, evolutionary development, system integration, software engineering, domain

engineering and reverse engineering. However, it is not clear how these areas can be jointly addressed in a systematic manner that is highly desirable for coping with the challenges of SOS evolution since a specific area often deals with only either a particular aspect or a particular scenario of SOS evolution.

For example, the concept of software architecture could mean different things in different studies from programming styles, application design solutions to middleware integration. Meanwhile, it is related to many if not all those areas mentioned. This leads us to believe the importance of investigation into the nature of SOS evolution and its environments.

A system evolution task involves many other activities apart from architecting and selecting a solution. As illustrated in Figure 1, system evolution is driven by two main inter-related factors, i.e., business changes and technology advances. New technologies have obviously attracted the most attention, while playing a dual role in system evolution -- *constraints* and *support*.

Furthermore, system evolution needs necessitate certain capabilities of architecture evolution. Consequently, there are many architecture-related issues that need to be looked at before reaching an evolution solution. To devise a generic and broadly applicable approach to system evolution in a SOS context, we need to examine all the architecture-related issues and consider its architecture evolution environment which, to date, has been a missing link. Examination of failures and difficulties in SOS evolution practice reveals that an AEE is a key supporting element to the success of the two main activities of system evolution: interoperability checking and solution generation (see Figure 1).

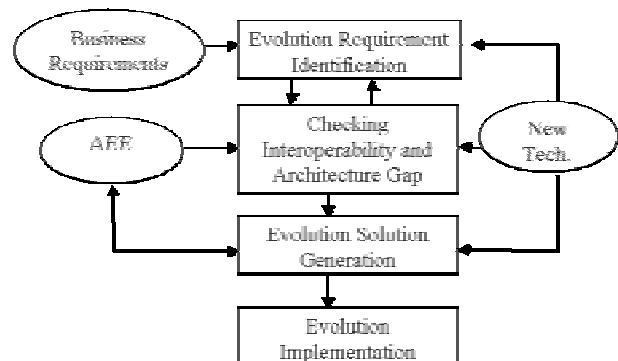


Figure 1. Activity context of SOS evolution.

3. WHAT IS AEE?

In devising a system evolution solution, we need to examine the *openness* and *readiness* of the systems involved. The introduction of an AEE provides such an examination environment that is based on three other important features of the systems, i.e., *autonomy*, *modularity* and *interoperability*. An AEE is an architecture knowledge environment of all systems or subsystems involved in or related to a system evolution, which has the following main capabilities:

- Provision and management of individual system architecture knowledge; and
- Supporting derivation and analysis of composed system architectures as required by the various evolution scenarios.

An AEE does not capture architecture knowledge in all aspects of systems but only what is necessary for enabling system evolution in a SOS context. It means an AEE is not simply a collection of the architectures of related systems although these architectures are the main resources for constructing the AEE.

As commonly recognized, system architecture knowledge is represented by multiple views. To ensure completeness of the architecture knowledge concerned, an AEE requires a well-designed structure that can handle not only architecture knowledge from different systems but also different views.

The knowledge required about a component system in achieving integration and interoperation with its peers should be sufficient but necessarily abstract. This abstract knowledge of the component system forms the basis for construction of the composed systems' architectures and is therefore referred to as the component system's *architecture interface*. In traditional practice of architecture, the architecture interface is not developed explicitly as part of system architecture.

The architecture interface of a system expresses the externally observable characteristics of the system, and is oriented towards what the system provides and requires, and how the system is to interact with its operating environment or peers. It is defined based on analysis of the component's own architecture or design, but has no direct reference to its internal composition. It is important to distinguish a system architecture and its interfaces in terms of what is captured and how they are represented.

In a particular evolution context with specified requirements, its AEE is defined as a set of architecture interfaces of all component systems involved in the evolution. Let $AI_{s(i)}$ indicate the architecture interface of System (i) and assume that an evolution $E(j)$ involves System (1), ... System (n), then its AEE can be denoted by:

$$AEE(E_j) := \{AI_{s(1)}, AI_{s(2)}, \dots, AI_{s(n)}\}.$$

Understanding what the AEE is for a particular evolution in an SOS context is critical before an architecture solution is reached for the evolution.

The determination of aspects covered by architecture interfaces is related to strategies used to handle the concepts of system architecture and interoperability since both of them mean different things to different people. Based on many architecture related works such as ISO 7-layer model and [4, 7, 8], we present a comprehensive definition of the architecture interface in a layered structure as shown in Figure 2.

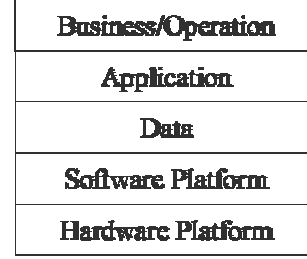


Figure 2. A layered structure for AI.

In general, there could be five aspects (main views) in a system architecture, that is, business operation (BUS), application (APP), data (DA), software platform (SP) and hardware platform (HP). Corresponding to the layered structure, the architecture interface of a system should address all these aspects, but with different focuses. The system architecture concerns the internal organisation of the system while the architecture interface (AI) is about the externally observable characteristics of the system. In the following discussion, we detail these aspects from the viewpoints of defining the architecture interface.

$$AI := \{BUS_{ai}, APP_{ai}, DA_{ai}, SP_{ai}, HP_{ai}\}$$

Business operation. The *business operation interface* concerns the operational role of the system and the operational procedures or policies governing the system's operational behaviour. It characterises some business functions (BF) of the system which are externally available and specifies the business and information interaction (BInteraction) of the system with external agencies.

$$BUS_{ai} := \{BF, Binteraction\}.$$

In the architecture interface, the business is specified in terms of the business processes and functions that the system *provides* to its (external) business environment and the *assumptions* that the system *makes* about its external business environment or that required. Both the business functions of the system and the environment assumptions are expressed in terms of information, capabilities and services (ICS) exchanged between the system and the environment.

$$Binteraction := \{ICS_{provide}, ICS_{assume}\}$$

$$ICS_{provide} := \{\text{information, capabilities, services}\}$$

$$ICS_{assume} := \{\text{information, capabilities, services}\}$$

In specifying the external business interaction, the protocols/dependencies/order of the information, capability and service exchanges are spelt out. For example, certain information needs to be exchanged before other information, or certain services need to be carried out before other services.

Application. The *application interface* defines the *mechanisms* that allow the system to exchange information, capabilities and services with its environment. More specifically, this aspect of the architecture interface makes concrete how the business intent and policies of the business operation aspect can be realised in terms of message exchanges or interactions between the system and its environment. According to our rich interface specification model for software components [Han, 1998], this aspect should include the following characterisations: *Interface signature/elements*; *Interface semantics*; *Interaction protocols*; and *Quality properties*.

In general, we have

$$\text{APP}_{\text{ai}} := \{\text{Sig, Sem, Config, Iprotocols, Qualities}\}$$
$$\text{Config} := \{\text{SI}_1, \text{SI}_2, \dots, \text{SI}_m\}$$

For more specific formulations of the application interface, please see [6]. In specifying the characteristics of the application interface, it is important to include their traceability to the business operation interface. With such information, we can easily trace how the system design (as reflected by the application interface) addresses the business objectives (as reflected in the business operation interface).

Data. As in the case for LISI [4], the *data interface* defines data elements, information formats, data protocols and/or databases that enable the exchange of data and information between the system and its environment. The specification of the data interface takes the form of data type definitions (DTD), database schema definitions (DSD), and constraints on data format and processing (Dconstraints) such as encryption. In general, the data interface specification reflects the information identified for exchange in the business operation interface, and relates to the application interface in the sense that it provides the definition of the information/data manipulated by the application interface.

$$\text{DA}_{\text{ai}} := \{\text{DTD, DSD, Dconstraints}\}$$

Software platform. The *software platform interface* specifies the run-time software platform and describes the communication protocols used by the software platform to realise the system-environment interactions and exchanges.

The run-time software platform may include the following as appropriate: operating system (OS), database management system (DBMS), web server/browser (WebS/B), middleware (MW), messaging (MS) and groupware (GW). The inter-component and system-environment communication protocols are those that are used for the components or the system and environment to interact, and are usually dependent on the software platform chosen. For example, CORBA middleware products communicate with each other using the Internet Inter-ORB Protocols (IIOP) for communication.

Note that the specification of the above software platform features should include a full specification of the relevant standards, vendors, versions, configurations, and so on. In addition, the software platform interface should clearly specify the expected software platform and communication protocols that its operating environment uses for interaction with the system. This clear specification of the system's dependency on the environment software platform and related communication protocols is essential to the interoperation of the system with its operating environment (including its peer systems).

$$\text{SP}_{\text{ai}} := \{\text{SP}_{\text{provide}}, \text{SP}_{\text{require}}\}$$
$$\text{SP}_{\text{provide}} := \{\text{OS, DBMS, WebS/B, MW, MS, GW, CommProts, ...}\}$$
$$\text{SP}_{\text{require}} := \{\text{OS, DBMS, WebS/B, MW, MS, GW, CommPros, ...}\}$$

Hardware platform. The *hardware platform interface* also identifies the hardware that the system depends on but describes the network characteristics of the hardware for communication with its operating environment, including the network protocols. As in the case for the software platform, the specification of the hardware platform interface makes explicit the hardware infrastructure needs of the system and of its networking with the operating environment. They may include hardware architecture

(Harch), network, capacity (Speed, Memory, Storage), network protocols (NetProtocols), and so on.

$$\text{HP}_{\text{ai}} := \{\text{HP}_{\text{provide}}, \text{HP}_{\text{require}}\}$$
$$\text{HP}_{\text{provide}} := \{\text{Harch, Network, Capacity, NetProtocols, ...}\}$$
$$\text{HP}_{\text{assume}} := \{\text{Harch, Network, Capacity, NetProtocols, ...}\}$$

From the above discussion, it is clear that the different aspects of the system architecture are related to each other, in the sense that a lower-level interface supports or implements a higher-level interface. Sometimes, a higher-level interface property determines the choice of a lower-level property. For Example, a particular middleware runs only on a specific operating system and machine with preset communication and network protocols.

4. AEE CONSTRUCTION

Knowing what needs to be captured at each aspect of the system architecture for the creation of the architecture interface is only the first step towards generation of the interface. Different aspects require different construction strategies for the interface generation. The interfaces generated for different aspects have different degrees of dependency on their corresponding views of the architecture. Generating interfaces for some aspects may be easier than others. Generally speaking, for instance, architecture interfaces for the software platforms and the hardware platforms of the involved component systems are relatively easy to create because architecture descriptions of their external features are usually well presented in documents by producers.

Whether a similar level of architecture interface descriptions for the other three aspects of existing systems can be easily generated depends on how the legacy systems were developed and maintained from architecture viewpoints. The better the system architecture is developed and maintained, the easier the construction of its interfaces is. There have been many interesting efforts made by the software architecture community to address various issues of the application interface, including [3, 6, 8, 9]. For architecture interface-based support for system evolution to become a reality, more practice and experience is required to develop adequate methods and tools for the construction and representation of architecture interfaces.

5. AEE APPLICATIONS

Interoperability checking. When two or more systems are to be integrated to form a composed system, a key issue for consideration is whether the component systems are compatible with each other under the intended architecture solution for integration. The interfaces of the component systems should satisfy each other as they are joined together according to the integration architecture. This involves that the required features of one side match the provided features of the other side at any connection point. This feature-matching requirement applies to all aspects of the architecture interface. Only when the component systems are compatible with each other can they fully interoperate. A properly generated AEE will provide natural support for checking the interoperability of the component systems and for identifying the level(s) of interoperability through using LISI [4].

Depending on the levels of architecture knowledge defined in the component systems' interfaces, the extent of interoperability checking may vary as pointed out by LISI [4]. Ideally, all the

component systems are compatible at all aspects of the architecture knowledge as required by the intended integration architecture. In the situations where component systems were developed or acquired independently, incompatibility is almost inevitable. That is, at some connection points, the provided capabilities of one side do not satisfy the required capabilities of the other, which is called a *component mismatching* or an *architecture gap*.

Rectifying the architecture gap is necessary to achieve the intended system integration. This could be achieved through a number of means, from simply choosing alternative component systems, to modifying some of the component systems, and to using component wrappers or interceptor components to change or transform the architecture interfaces. If there are gaps in terms of system functionality, additional components need to be developed/acquired and introduced into the integration architecture.

System integration and evolution. The composition or integration of a large system from a number of *existing* and *new* systems involves interconnecting the component systems under a particular architecture to form an inter-related whole, that is, the composed system. Based on the specifications of the architecture interfaces of compatible component systems, we can define the system architecture and derive the architecture interface for the composed system. Figure 3 depicts the use of the architecture evolution environment in SOS integration and evolution.

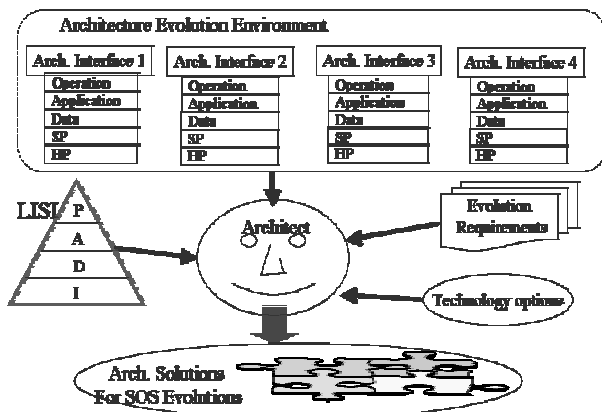


Figure 3. AEE enabling SOS evolution.

The system architecture of the composed system is defined as an architecture solution. It involves a series of compositional architecture analysis against all aspects of all AIs to arrive at the system architecture specification for the composed system in terms of the various architecture views. In general, different evolution scenarios require different ways to generate architecture solutions. In both the joint and emergent evolution, we consider all the existing systems involved and the part to be generated as component systems. In order to maintain the identity and autonomy of the component system, if necessary, the concept of the composed system can be considered as a temporary one used only for evolution discussions.

After the internal system architecture of the composed system is determined, its external architecture interface can be generated. The different aspects of the architecture interface reflect the corresponding aspects of the system architecture, but emphasizing their externally observable characteristics.

6. CONCLUSIONS

An AEE-based approach has been introduced to facilitate SOS evolution. This approach can not only provide methods for the construction of a knowledge environment from an architectural viewpoint but also create a framework for integrating many software integration strategies or techniques to deal with complicated SOS evolution challenges.

7. REFERENCES

- [1] R. Allen and D. Garlan. A formal basis for architecture connection. *ACM Transactions on Software Engineering and Methodology*, 6(3):213-249, July 1997.
- [2] L. Bass, P. Clements and R. Kazman. *Software Architecture in Practice*. Addison Wesley, Reading, MA, USA, 1998.
- [3] J. Bradshaw, S. Dufield, P. Benoit and J. Wooley. "KaoS: Toward An Industrial-Strength Open Architecture", to appear in *Software Agents* (Edited by J. M. Bradshaw), MIT Press.
- [4] C4ISR Architecture Working Group, "C4ISR Levels of Information Systems Interoperability", DOD, USA, 1998.
- [5] P. Clements. A survey of architecture description languages. In *Proceedings of the 8th International Workshop on Software Specification and Design*, Paderborn, Germany, March 1996.
- [6] J. Han. "A comprehensive interface definition framework for software components", in *Proceedings of the 1998 Asia-Pacific Software Engineering Conference*, pages 110-117, Taipei, Taiwan, December 1998. IEEE Comp. Society Press.
- [7] J. Han, P. Chen and A. El-Sakka. "Representation and management of system architecture at the enterprise level". In *Proceedings of the 2nd Australasian Workshop on Software Architecture*, pages 9-23, Melbourne, Australia, November 1999.
- [8] M. T. Mallkoun and E. A. Kendall, "CLAIMS: Cooperative Layered Agents for Integrating Manufacturing Systems", in the Proceedings of PAAM'97, London.
- [9] R. Sanlaville, Y. Ledru, J. Estublier, and J.M. Favre, "Architecture Environment for the Evolution of Complex Software", to appear in *Software Architectures, Components and Frameworks* (edited by M. Fayed, et al).
- [10] M. Shaw, R. DeLine, et al. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4): 314-335, April 1995.
- [11] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, Upper Saddle River, NJ, USA, 1996.