Carnegie Mellon University
Software Engineering Institute

AN EARLY LOOK AT DEFINING VARIABILITY REQUIREMENTS FOR SYSTEM-OF-SYSTEMS PLATFORMS

John Klein, Gary Chastek, Sholom Cohen, Rick Kazman, and John McGregor

September 2012

Original Publisher: IEEE

This paper was published by IEEE in the proceedings of the 2012 Second IEEE International Workshop on Requirements Engineering for Systems, Services, and Systems-of-Systems (RESS), Chicago, IL, 2012, pp. 30–33.

Abstract

In the commercial domain, platform-based approaches, in which a set of functions or services are bundled to form the basis of many products, have enabled efficient development of systems and their composition into systems of systems. A successful platform must balance sufficient commonality to support economical reuse while also providing variability and extensibility to enable innovation in system and system-of-systems (SoS) capabilities. These commonality/variability tradeoffs for SoS platforms are frequently tacit decisions, since there are no accepted techniques for analyzing such decisions at the scale and degree of requirements uncertainty that characterize most SoSs. The objective of our work is to develop a method for analyzing decisions about requirements for common platforms for SoSs. The method begins with the requirements tasks of identifying and selecting appropriate variabilities (variation points, variation ranges, and variation decision binding times) to support immediate SoS needs and enable innovation and controlled evolution. We are currently conducting a workshop and interviews with SoS experts to define the essential technical problems in SoS common platform development and identify solution constraints. We will then define a simplified SoS with limited capability requirements to use as a model problem. We will use the model problem to assess the fit of existing scope, commonality, and variability methods from software product lines to the SoS context and extend existing economic models using real options and probabilistic models to model uncertainty in evolution requirements. While it is too early to draw firm conclusions about the effectiveness of our approach, it is based on proven technologies from the mature field of software product lines, so we have confidence that we can build successful SoS techniques from this foundation.

Keywords—component; system of systems; software requirements; platform engineering; software product lines; system of systems architecture

INTRODUCTION

System of Systems (SoS) architectures based on common platforms have been successful in enabling rapid development and composition of independent systems, and promoting innovation, while allowing controlled evolution as requirements change. For example, platforms from Apple, Facebook, and eBay provide common capabilities (e.g., authentication and authorization, geo-location, social graph, payment processing, and software distribution), and define integration and interoperation mechanisms for constituent systems, resulting in robust and dynamic ecosystems [4].

One strategy for defining a platform is software product lines. A *software product line* is a set of software- intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [6]. A platform for a SoS represents a larger scale of reuse across an open set of independent system products that may individually address different market segments or missions, but are also composed into a SoS to achieve particular capabilities. Each type of constituent system product may also be a member of a market segment- or mission-specific software product line. The issues arising from *intersecting* product lines are not addressed by current methods used for software product line scoping, commonality, and variability analysis.

A distinct feature of our research is our going-in assumption that it is not only the functional requirements or even the usual quality attribute requirements that have led to the success of commercial platforms such as those listed above. The requirements have broadened beyond a single organization to the needs of a set of interacting partners. These partners perform complementary functions that the central platform must be prepared to support. It is these variations among systems and organizations that stimulate the need for new variation mechanisms.

Our informal research hypothesis is "Identifying and understanding the requirements for multiple common platforms, intended to support systems of systems, can be realized through enhancements to existing software product line practices such as scoping and understanding relevant domains." We intend to attempt to validate this hypothesis by examining successful common platforms, focusing on variation mechanisms that have been, or are being, used successfully, and understanding the architectural mechanisms that will support their successful creation, interoperation, and evolution.

In addition, we have several more detailed hypotheses:

1. SoS context (independence of development and operational authorities across constituent systems [7]) leads to different variability realizations and binding times, compared to existing software product lines.
2. Cost/benefit economic modeling can be applied to make effective common platform architecture decisions.
3. Use of the product line-based scoping approach for SoS common platform architectures will achieve a meaningful reduction in SoS development, integration, and sustainment cost, schedule and/or risk.

BACKGROUND

Several terms will be used in this paper for which multiple definitions can be found in the research and practitioner literature. In this section we provide definitions for these terms. Some of the concepts that are key to understanding our work are: variation, software product lines, platform, and system of systems.

All software-intensive products vary from one another by the features they provide and the characteristics they exhibit. The *variations* among products include differences in the functionality provided, the implementations of that functionality, the quality attributes of the resulting products, and when the choices among variants are made and bound to the system (binding time). The mechanisms used to permit easy binding of variants enhance certain quality attributes and degrade certain others. We are interested in variation within a set of products that share a common design and implementation - i.e., the software product line - and across a set of product instances being integrated to provide specific capabilities, i.e., the SoS. For example, two products may vary based on the level of security built into the products, but all of the products in a SoS may be required to work together to provide a specified level of security.

When a set of products addresses a particular market segment or mission, the features of the products (and the consequent variations) can be managed to create a *software product line* [6]. A software product line is based on a common architecture, which is designed with appropriate variation points. The product building assets can be factored to achieve the maximum reusability by capturing the maximum commonality across the set of products.

Strategic levels of reuse can also be achieved by defining an environment that includes, at a minimum, a set of hardware, software interfaces, and component implementations. These elements are a "*platform*" that will form the basis for implementing a set of products [3]. We will designate a platform that is made available to and used by a number of organizations a "common" platform. The .NET framework from Microsoft is an example of a common platform. A platform may provide (or be accompanied by) a software developer's kit (SDK) that includes reusable assets such as code libraries, test cases, policies, standards, documentation, and patterns. The platform provides assistance in product composition and often encapsulates the operating system or at least a runtime environment. The platform itself becomes a standard, if successful, and may be given over for community ownership. This has happened for example with the Java language and runtime environment. Figure 1 shows a typical SoS in which each of the constituent systems is based on a separate platform.

A SoS results when independently useful systems are integrated into a larger whole that delivers unique capabilities [7]. To facilitate the deployment of several instances of the SoS a common platform is defined and deployed at multiple sites and potentially on multiple target platforms. For example, Fig. 1 shows an in-car navigation SoS constructed by integrating signals from GPS satellites with information from Google Maps (and perhaps other sources such as live traffic feeds) to give detailed real-time travel directions. A similar SoS might be deployed on a smart phone.

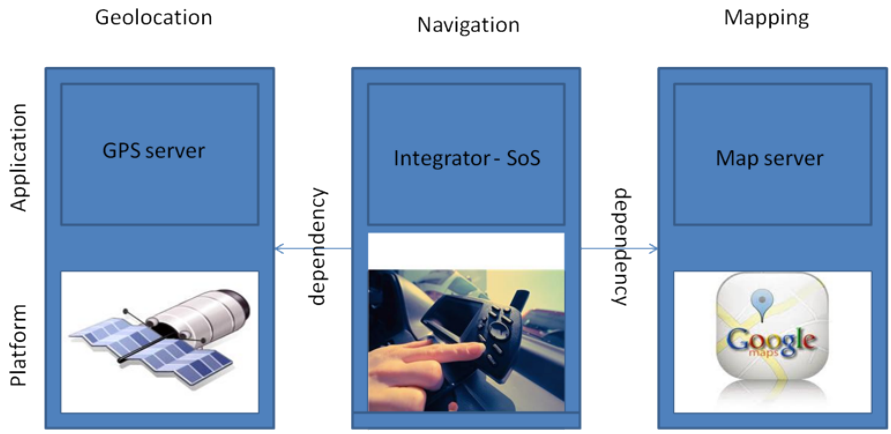

Figure 1: Example SoS

Some SoSs are intentionally built with all the constituent systems designed from the start to work together. More commonly, however, the systems are all initially built under a variety of independently defined assumptions and requirements. Figure 2 shows a platform- based approach to developing systems and composing them into a SoS. In the leftmost panel of Fig. 2, the individual systems are each built to run on the platform. The SoS is then constructed by linking the independent systems using a variety of integration mechanisms provided by the platform. The individual systems may be deployed on a complex arrangement of interconnected processors – each supporting a different platform, and communicate with each other either through planned modes and channels of communication or through post hoc mechanisms.

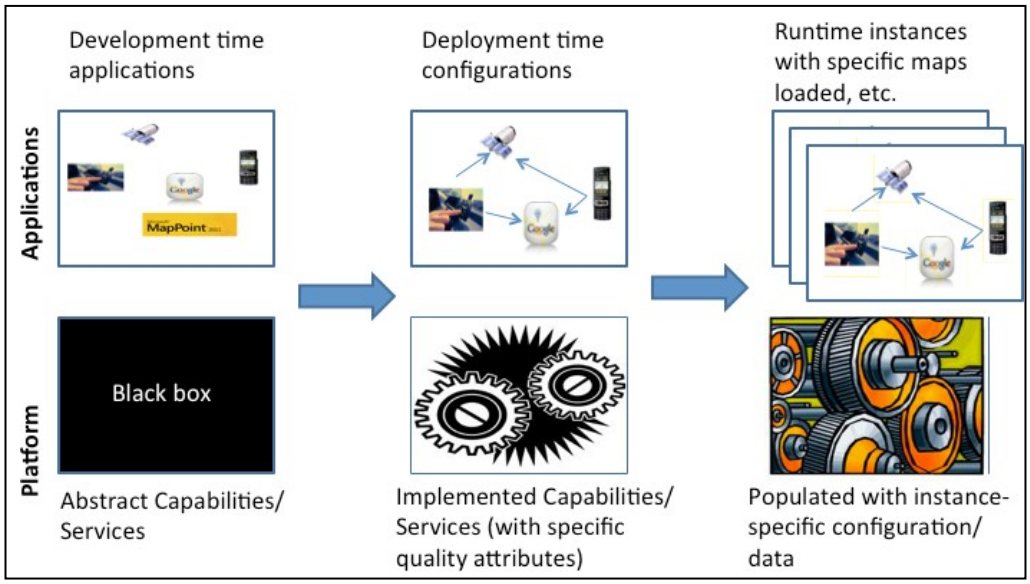

Figure 2: Levels of Definition

We call a specific set of systems interoperating at runtime to form a SoS a *configuration*. Multiple *instances* of that configuration may be deployed on multiple physical machines (see Fig. 2). A system may participate in more than one SoS configuration at a time. Two systems of systems are

"different" if they have different configurations. For example, in Fig. 2, multiple mapping services are illustrated in the left panel. Each SoS configuration would have one of the two services included, as shown in the middle panel. Multiple instances of that configuration would be deployed, as shown in the right panel. The degree of difference ranges from the SoSs having the same systems but having differences in the versions of some of the systems to each system of systems having totally different systems. These differences can include differences in hardware, operating systems, quality attributes, and features.

A set of configurations may be enabled by inserting explicit variation points in the SoS so that new configurations can be derived easily. One of the goals of our research effort is to investigate variation points in the common platforms so that new SoS configurations to be easily derived.

METHOD

Our method of investigation has two concurrent threads, as illustrated in Figure 3. All of this work will take place in the context of a model problem, which will be identified through a challenge workshop to ensure that the problem is significant to our stakeholders and that it is realistic in its scope and content.

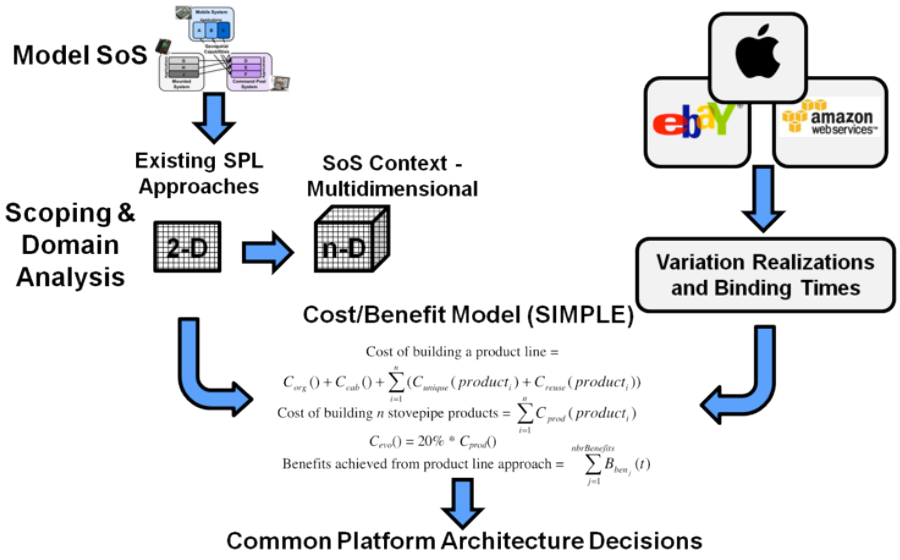

Figure 3: Method of Investigation

Our initial candidate for the cost/benefit economic model is an extended version of the Structured Intuitive Model for Product Line Economics (SIMPLE), which has been widely used in software product line organizations [5]. This model supports cost and benefit estimations covering multiple products over time. Variations on this basic model have been used to compute the Return on Investment (ROI) for a software product line [2]. For use with SoS the SIMPLE model will be supplemented to account for the additional dimensions outlined above, as well as additional categories of costs and benefits associated with the SoS scale and context.

The cost/benefit economic model must also address the factors that arise because SoSs are large and complex, require long development times, endure frequent modifications and substitutions, and are long lived. As a result, early requirements models and development plans are often developed with inadequate knowledge [Maier 2005], and using a deterministic cost/benefit model applied at a single point in time is insufficient. These models and plans are virtually always wrong and must be modified. Even when modified, the models are changed many times during the life of the system of systems.

- One thread of the investigation begins with existing software product line practices, particularly *scoping* and *understanding relevant domains* in the SEI's Framework for Software Product Line Practice [10], and seeks to scale those practices to a SoS context. Intuitively we view existing software product line approaches as working in a two dimensional space (products × features) while a SoS exists in an n- dimensional space (products × interactions × configurations × features ×…).

  We have deep experience in software product line practices. The Software Engineering Institute has developed, field tested, and published many of the techniques used for many aspects of software product line practice including economic modeling, requirements engineering, and architecture design.
- The second thread examines the architectures of successful common platforms to identify variation mechanisms and binding times, through interviews with platform architects and product managers, and review of open literature. This thread focuses on detailed hypothesis #1, that the SoS context of independence of development and operational authority leads to different variability realizations and binding times, compared to existing software product lines.

Detailed hypothesis #2 is addressed as these two threads provide input to a cost/benefit economic model that supports an objective decision process for making architecture decisions about the common platform, as shown in Fig. 3. The primary inputs to the cost/benefit model are the potential requirements for commonality and variability of the common platform, and the repertoire of variation mechanisms and binding times (each with a particular cost and set of quality attribute tradeoffs) available to realize the variabilities. are often created with inadequate knowledge [8], and using a deterministic cost/benefit model applied at a single point in time is insufficient. Furthermore, the models are changed many times during the life of the SoS as the system evolves [1]. So the cost/benefit model will be augmented to incorporate uncertainty.

We will approach the issue of frequent requirements changes in two ways. First, the use of SoS relevant variation mechanisms and strategic reuse techniques will allow an organization to anticipate the trajectory of a domain and to plan assets that will meet the needs as they emerge. Second, the model techniques we develop will anticipate the uncertainty common in SoS. Real options is one approach to reasoning about uncertainty in cost/benefit analyses and is one of the techniques we are already investigating for quantifying design choices at variation points [9].

Our investigation also includes activities to assess the efficacy of our approach, addressing detailed hypothesis #3, that the approach will achieve meaningful reduction in SoS development, integration, and sustainment cost, schedule, and/or risk:

- Ongoing review by Challenge Problem Workshop participants to assess whether technical solution conforms to non-technical context constraints
- Collaborations with on-going DoD and non-DoD projects
- SPRUCE[1] "SoS Architecture" community of interest survey and collaboration
- Create a model SoS for 3-4 capabilities spanning multiple systems
  - Analyze the variability required for the capability – system interactions, quality attributes, economic concerns, and any constraints.
  - Design common platforms that support the variabilities needed for the capabilities.
  - Evaluate the designs of the platforms and estimate cost and benefit to application development from adopting the platforms

SUMMARY

Our goal is to create an approach for analyzing decisions about requirements for common platforms for systems of systems. Our hypothesis is that we will be able to do this by starting with the software product line practices and scaling them to cover the larger scale, more open-ended development of systems of systems. We will continue to report to the community as we carry out our research and evaluation activities.

ACKNOWLEDGMENT

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

REFERENCES

1. Abbott, R. "Open at the top; open at the bottom; and continually (but slowly) evolving," *IEEE/SMC International Conference on System of Systems Engineering*, 2006.
2. Günter Böckle; Paul Clements, John D. McGregor, Dirk Muthig, & Klaus Schmid. Calculating ROI for Software Product Lines, *IEEE Software*, Volume 21, Issue 3, May- June 2004, pages 23-31.

[1] https://www.sprucecommunity.org/SitePages/Home.aspx

3. Booz Allen Hamilton. Systems-2020 Study, Booz Allen, Hamilton, 2010.
4. Gary J. Chastek and John D. McGregor. It takes an ecosystem, Systems and Software Technology Conference (SSTC 2012), April 2012.
5. Paul C. Clements, John McGregor, and Sholom G. Cohen. The Structured Intuitive Model for Product Line Economics (SIMPLE), Technical Report, CMU/SEI-2005- TR-003, February 2005.
6. Paul C. Clements, Linda M. Northrop: Software Product Lines: Practices and Patterns, Addison-Wesley, 2002.
7. Mark W. Maier "Architecting principles for systems-of- systems." *Systems Engineering 1*, 4 (1998): 267-284.
8. Mark W. Maier, "Research Challenges for Systems-of- Systems," *IEEE International Conference on Systems, Man and Cybernetics* 2005, pp. 3149 – 3154.
9. John D. McGregor, J. Yates Monteith, Jie Zhang: Quantifying value in software product line design. SPLC Workshops 2011: 40.
10. Linda M. Northrop, Paul C. Clements: A Framework for Software Product Line Practice, Version 5.0, http://www.sei.cmu.edu/productlines/frame_report/, visited May 9, 2012.

Contact Us

Software Engineering Institute
4500 Fifth Avenue, Pittsburgh, PA 15213-2612

Phone: 412/268.5800 | 888.201.4479
Web: www.sei.cmu.edu
Email: info@sei.cmu.edu

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM18-0774