Toward an Evolutionary System of Systems Architecture

Scott A. Selberg
Agilent Technologies, Inc.
1400 Fountain Grove Parkway
Mailstop 4USE Col 26HH
Santa Rosa, CA 95403

Mark A. Austin
Institute for Systems Research
University of Maryland
College Park, MD 20742

Abstract. While recent advances in computing/communications technology have enabled the development and managed evolution of large scale system of systems (SoS) applications, lessons learned from industry indicate that these projects are not always successful. A key problem is the lack of a formal framework from which the development and management of SoS architectures can be studied. This paper describes the key ingredients and approaches to formal analysis that we believe will end up in this architectural infrastructure.

Introduction

The world is full of systems that are being upgraded from industrial- to information-age capability. System upgrades are commonly driven by the need for enhancements, either by expanding functionality or improving performance. Often, present-day systems are limited by their ability to: (1) sense the surrounding environment in which they are operating, (2) look ahead and anticipate events, and (3) control system behavior. In an effort to relax, or even remove, these constraints, next-generation systems are incorporating advances in computing, sensing, and communications. Sensors gather data which is fed to computers for advanced processing. The enriched information leads to better decision making which in turn is fed back to automated control systems. Through this process, the aforementioned barriers are overcome, thereby providing the desired results.

The first important consequence of this trend is that over time, present-day reliance on centralized management of operations will be replaced by operations that are partially or fully decentralized. Design of the latter class of systems is significantly more difficult than for centralized control. The second important observation is that there exists a growing class of engineering applications for which long-term "managed evolution" of multiple individual systems in the primary development goal (vs. the more common build, design, operate, retire cycle). One particularly vexing problem lies at the intersection of these observations – that is, in supporting continuous change in decentralized systems - either to rejuvenate them, reduce support cost, or again enhance capability. Looking forward, this problem will only be aggravated the ever increasing system complexity. The hypothesis of our work is that these challenges can be kept in check through disciplined application of existing solutions and extension and adaptation of techniques currently under development. Accordingly, the purposes of this paper are to explain: (1) The context which requires an evolutionary architecture, (2) The key architectural characteristics that position a system of systems to evolve over time, (3) The

extent to which these characteristics have already been architected.

Systems of Systems

A system of systems is one of many names describing a particular decentralized architectural paradigm. While there is no universal definition (Sage 2007) there is a general consensus about several characteristics and the growing importance of these systems. In "The Art of Systems Architecting" (Maier 2000) Maier and Rechtin define a system of systems as one in which its components:

1. Fulfill valid purposes in their own right, and continue to operate to fulfill those purposes if disassembled from the overall system
2. Are managed (at least in part) for their own purposes rather than the purposes of the whole; the components systems are separately acquired and integrated but maintain a continuing operational existence independent of the collaborative system.

Two key characteristics which derive from this definition are:

Emergence: Properties which do not belong to any of the constituent parts will emerge from the combined system of systems.

Evolution: The system of systems will change over time as constituent systems are replaced.

Due to advances in system automation, systems of systems are an ever increasing reality of our world. We see them in weather monitoring systems, where information is synthesized from large arrays of heterogeneous and geographically distributed automated weather sensors. They are appearing as smart homes and so-called smart spaces with automated lighting, heating (Clothier 2005). Looking ahead in field of Robotics are fully autonomous cars (Wikipedia 2007a} and hazardous environment search teams (Robotic Search and Rescue 2007). Two important present-day examples are the Internet and business Enterprise Resource Planning (ERP) systems.

The Internet is the classic example of a system of system done well. It is in a constant state of growth and flux as computers and websites come and go, yet it continues to provide the richest source of information on the planet. Business ERP systems are often excellent examples of improperly designed systems of systems. Many ERP systems have been assembled ad-hoc over time with no systems of systems engineering. As a case in point, in 2000, Ford Motor Company decided to replace several custom mainframe based supply chain applications with a unified solution from Oracle. In 2004, after investing millions of dollars, Ford Motor Company decided “to transition back to the proven, current system.” (Ford 2004).

Ford's situation illustrates a very important and common problem. Systems of systems tend to be large, expensive, and live for a long time. In this paradigm, dealing with obsolescence becomes a major issue. For example version 10.20 of the HP-UX operating system is not only out of support life but it is not even possible to purchase new computers that run it; however, it is still actively used in many places. Migrating the computing platform is very expensive. Companies know the day is coming when it can no longer be avoided. However, they will use extreme measures to delay it for as long as possible. For example, the government has paid up to $20,000 for a particular model of obsolete HP-UX hardware of the same vintage as the Pentium processor.

Lessons learned from industry clearly indicated that not all migration efforts succeed. The critical success factors are the skill of the engineers and the complexity of the existing systems. Ford was very determined to make the Oracle ERP system work and threw lots of money and talent at the problem; however, the effort still failed. Many companies have been or will be where Ford was - faced with a very complex problem and no easy answers. To reduce both the likelihood and severity of failure in SoS projects, there is a strong need for techniques that will enable reductions in SoS complexity and enable managed evolution.. While there may be little help for existing systems of systems, it is important to add architectural features into the systems of systems design that facilitate evolution during the migration of old systems or in the creation new ones, thereby avoiding Ford's predicament.

Key Characteristics of an Evolutionary Architecture

All systems of systems evolve by nature, so we need to be precise in defining an evolutionary architecture. For the purpose of this paper, an evolutionary system of systems architecture will be defined as one which conforms to the following two principals:

- The complexity of the system of systems framework does not grow as constituent systems are added, removed, or replaced.
- The constituent systems do not need to be re-engineered as other constituent systems are added, removed, or replaced.

To understand this definition better, it is useful to look at the origins of many systems of systems and how their architecture does not facilitate evolution. A system of systems commonly begin as a single system. As additional functionality is needed, additional systems are added. Over time the number of systems, interconnections, and interface protocols grows making the system increasingly complex and difficult to maintain. During the evolution, while each of the constituent systems may be documented, it's not uncommon that the overall system of systems documentation is ignored. In the end it becomes a huge, mission critical, convoluted, and undocumented mess as depicted in Figure 1.

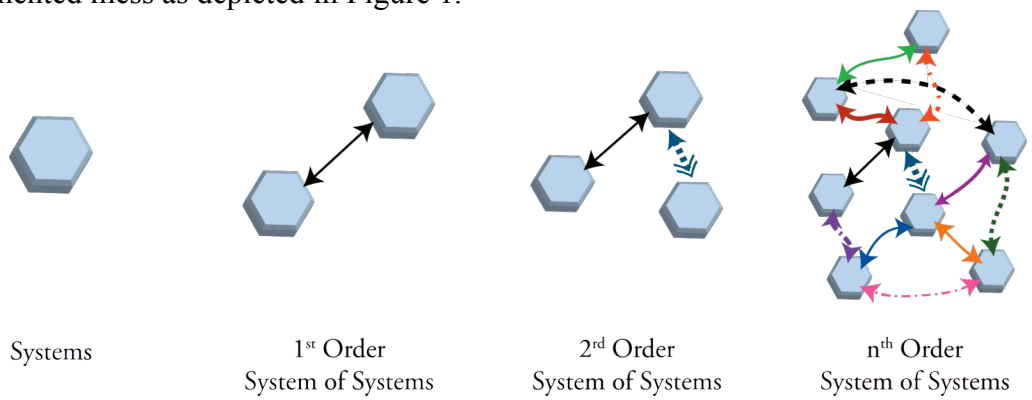

Figure 1: Unmanaged Systems of Systems Evolution

Obsolescence will often drive the need to replace some constituent system. If we start with the above example in the n^{th} order state and try to replace one of the systems, we can immediately see some of the problems. The first difficulty is that the new system either needs to support

every interface protocol of the old system (natively or through an adapter), or the partner systems will need to be updated to support newer protocols. In either case, this is a lot of work, as shown in Figure 2.

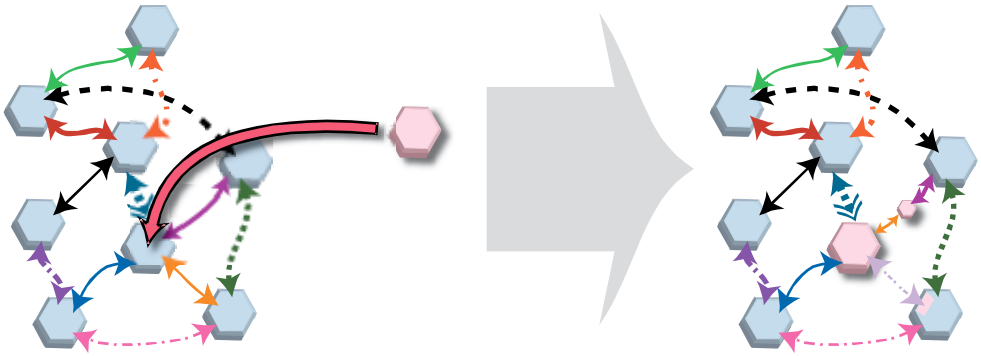

Figure 2. Replacement of a Constituent System in an Unmanaged N-th Order System of Systems

We postulate that an evolutionary system of systems architecture, which can avoid this situation, will have the following three characteristics: standard interfaces, interface layers, and a continual system verification and validation process.

Standard Interfaces

The first step toward making life easier is to architect the system of systems around a universal standard. The existence of such a standard means that when a system needs to be replaced, re-architecting of interfaces will not be required. As shown in Figure 3, this is a huge advantage and the first key to building an evolutionary architecture..

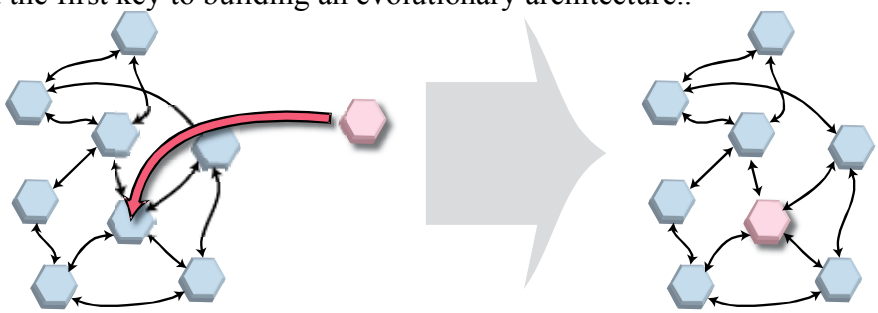

Figure 3. Replacement of a Constituent System in a Common Standard N-th Order System of Systems

Maier and Rechtin observed that in the context of systems of systems, standards are network goods. "For example, telephones are network goods. A telephone that doesn't connect to anybody is not valuable. Two cellular telephone networks that can't interoperate are much less valuable than if they can interoperate (Maier 2000)." It is therefore important to select standards which are broadly used rather than expect a custom definition to become an industry standard.

While using a single standard interface provides a huge advantage, in practice it neither completely solves the problem, nor is it completely realistic. If the new system has the same

interfaces as the old, then the partner systems will each need to be guided to the new system. This is the first limitation. As an example, consider moving a database from an old machine to a new one. While relational database providers such as Oracle have provisions to handle this scenario, it conceptually illustrates the point. The new server can be brought online and host up the exact same database as the old server; however, to make it work each of the partner systems will need to be taught the name of the new server. This can be difficult for various reasons such as the source code of the legacy system has been lost, the developers of the legacy system have left the company, or the legacy system will need a full qualification to comply with governmental regulations.

The second limitation with using a single common standard interface is accommodating the need for interfaces themselves to change over time. For example, IEEE 488.2 (GPIB) has been the instrumentation connectivity standard for decades; however, it is being replaced by LAN due to cost, speed, and expandability reasons (LXI 2007). It is therefore vital that a system of systems have some means of migrating not only the constituent systems, but also the interface standards.

Interface Layers

A solution to these limitations is the inclusion of an abstraction or interface layer. In some cases the layer can be very thin. For example, it is a common practice to alias the name of a web server to the actual machine name. Therefore, when the time comes to change, the alias can be redirected to the new server and the clients don't need to be aware of the change. In other cases the layer needs to be more advanced. For example, the interface layer may need to negotiate a physical interface boundary such as GPIB and LAN. The interface layers are the system of systems engineer's primary control point to facilitate change. A system of systems with an interface layer is depicted in the Figure 4.

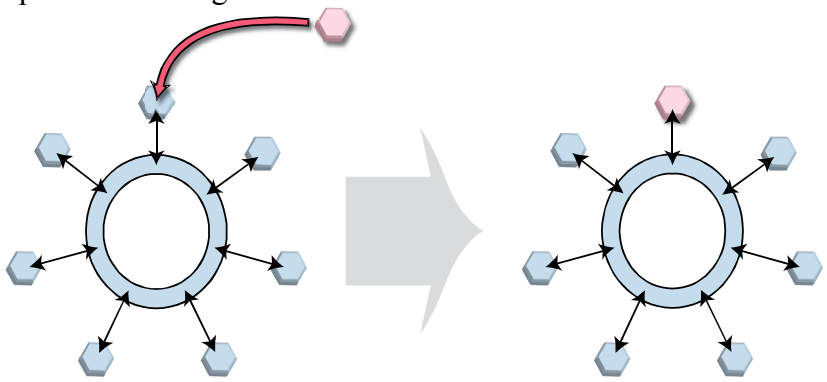

Figure 4. Replacement of a Constituent System in a Interface Layer Based N-th Order System of Systems

One of the complexities with evolving a system of systems is that they are often very sensitive to downtime. For example, production lines are often systems of systems and cannot tolerate periods of downtime for more than a few hours. Therefore, when the time comes to change it is important that it can be done quickly and successfully. In the nth-order system of system in Figure 1 this task is a nightmare. There can be so many change points that making the change quickly is impossible. The testing of the new constituent system is limited because the

emergent behavior of the system of systems can only be observed when all the pieces are in place. Because of these issues, systems changes are usually very stressful periods of throwing the switch and fixing the inevitable bugs as quickly as possible.

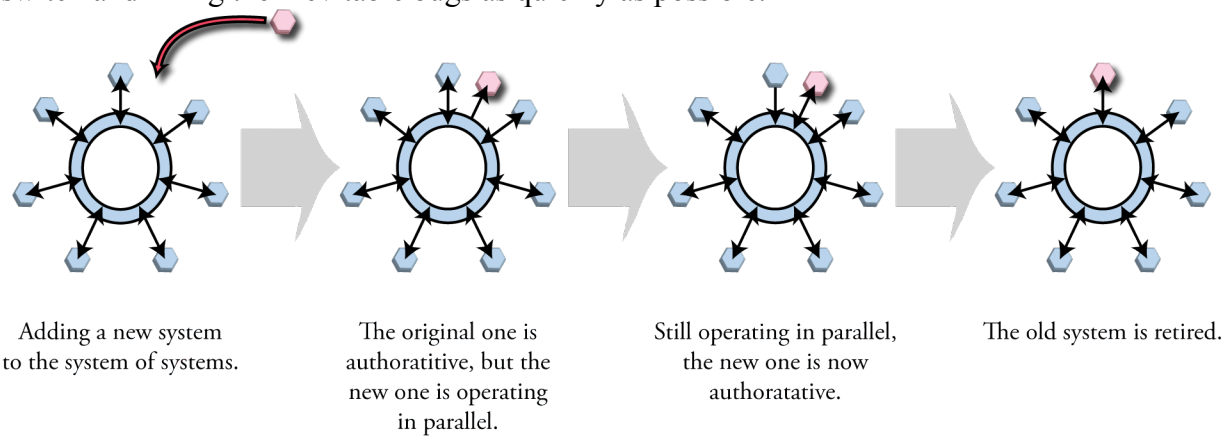

Figure 5. Transition of a Constituent System in a Interface Layer Based N-th Order System of Systems

An intelligent interface layer can greatly assist this task. One approach is to bring both the new system and the legacy system online in parallel with one of them being declared the authoritative one. Messages to the old system can be replicated by the interface layer to the new system allowing the developers to see some of the emergent system of systems behavior during the testing phase without touching the partner systems. The interface layer then provides a single control point for making a change - thus it is a fast process. When the change is made, messages can still be sent to both systems, but now the new system, as the authoritative one, responds. Thus, should unexpected behaviors be found when the switch happens, the interface layer provides a mechanism for quickly reverting back to the old system. This process is shown in Figure 5. This is a limited, but potent scenario, provided by having the interface layer. And while interface layers won't solve all the evolution problems, it is a necessary control point to being able to address many them.

Continual System Verification and Validation Process

When SoS behaviors and structures are small, easy to understand, and relatively static, then the application of system standards and interfaces is likely to be sufficient for keeping the difficulty of managed evolution in check. For all other cases, good long-term solutions to the systematic and managed evolution of SoS architectures will employ a ``continual system verification and validation process'' as an integral part of ensuring design correctness, preventing the managed system evolution straying from its intended pathway, and containing recurring costs both in money and time. Moreover, with the ongoing advancement of information-age systems, verification of design correctness and implementation will soon demand approaches to design that are based on automated verification mechanisms (Jackson 2006, Sangiovanni-Vincentelli 2000). These mechanisms include:

- **Formal Models.** We need ways to capture the design representation and it specification in an unambiguous ``formal language'' that has precise semantics.

- **Abstraction.** Abstraction mechanisms eliminate details that are of no importance when evaluating system performance and/or checking that a design satisfies a particular property.
- **Decomposition.** Decomposition is the process of breaking a design at a given level of hierarchy into subsystems and components that can be designed and verified almost independently.

For both individual systems and SoS, standards and interfaces play the role of simplifying a problem domain through abstraction. The use of formal models extends the range of problems for which automated verification of correctness and decision making is possible.

Architectural Implementation

While each system of systems is unique and will need to select its own standards, interface layers, and verification and validation processes, it is useful to examine some specific implementations to understand the current state of the art. Where possible, methodologies and tools to support architectural implementation and evaluation should build upon SoS infrastructures already in place. We strongly believe that future Internet technologies will play a central role in future SoS architectures.

Standard Interfaces

There are hundreds of standards available encompassing a wide diversity of fields (e.g., metric measurements, 120V power, English writing). For really large SoS, expecting everyone to buy into small set of standard interfaces is likely to be overly optimistic. History indicates, for example, that the use of language enables communication, but can also help to define the identity of a society. Hence, rather than insist that all interfaces conform to a standard, an alternative approach is to develop architectural frameworks so that they can work with heterogeneous standard and interfaces. The World Wide Web is perhaps the most successful SoS that follows this philosophy.

In his original vision for the World Wide Web, Tim Berners-Lee described two key objectives: (1) To make the Web a collaborative medium; and (2) To make the Web understandable and, thus, processable by machines. During the 1990-2000 time frame, the first part of this vision has come to pass. Today's Web is designed for presentation of content to humans. Humans are expected to interpret and understand the meaning of the content. The second part of this vision is still in development and is called the Semantic Web. Briefly, the Semantic Web aims to give information a well defined meaning, thereby enabling machine-to machine communication.

Figure 6 shows the layers of standards currently in development for the Semantic Web (Berners-Lee 2000). The lower layers define mechanisms for multi-lingual support of languages and an ability to represent and link documents. The middle layers focus on the definition of graphs of resources (this, after all, is the Internet), use of ontologies to define key concepts in problem domains, and an ability to logically reason with collections of facts and rules. This infrastructure will enable proofs – you can make an assertion, and the result will be true or false

– and, eventually, provide guidance on issues related to trust. Can you always trust what you read on the Internet? Of course not.

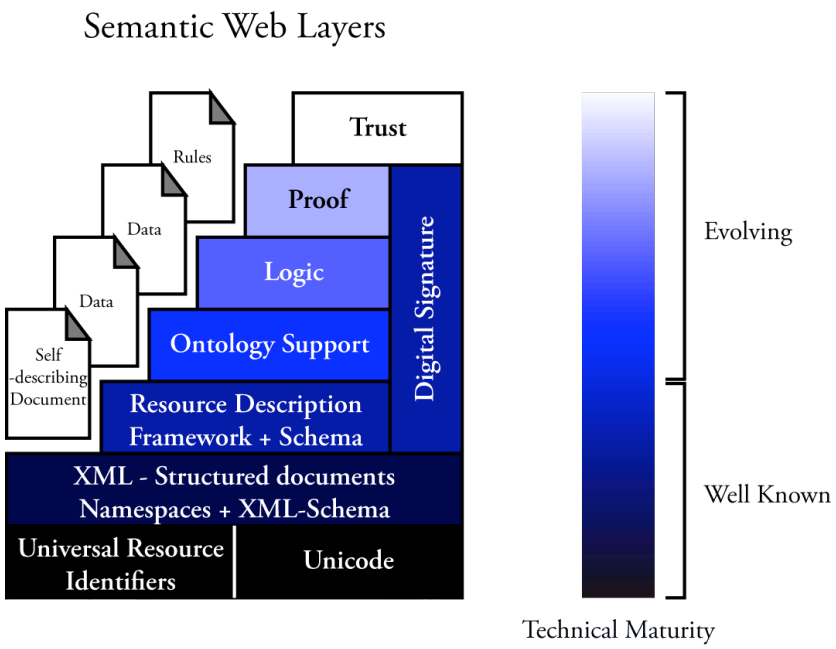

Figure 6. Semantic Web Layers

A second very interesting design feature for the Semantic Web is an ability or tolerance to work in the presence of faults. An ideal system has no faults. Computer scientists have learned over the past few decades, however, that if you want a system to be popular and well used, the system has to work, even if it can't work perfectly. A well designed system architecture should be tolerant to gaps in specifications, and where critical metrics of SoS functionality, performance and/or properties will not be compromised, fill in missing details for imperfect applications.

Interface Layers

An interesting feature of the Internet is the ability to support a large number of simultaneous standards. The key is multiple interface layers. Figure 7 shows the five layers of the Internet reference model (Wikipedia 2007b).

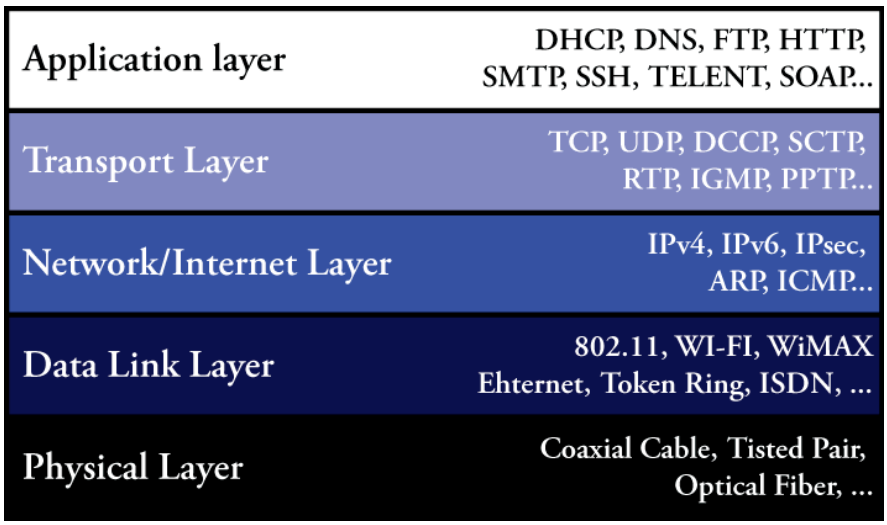

Figure 7. TCP/IP Internet Model

Business ERP systems are also starting to employ an abstraction layer described as the Enterprise Service Bus (ESB). An ESB piece of software, “that lies between the business applications and enables communication among them." (Wikipedia 2007c). The ESB has been a hot area of growth in recent years. Dennis Bryon of ebizQ notes, ``In my 2007 MOM/ESB research, I found almost a dozen ESB/MOM OSS projects that began in 2003 or since that have matured from the community stage to production (Byron 2007)." The promises of the enterprise service bus are inline with the needs of the evolutionary system of systems architecture: Faster and cheaper accommodation of existing systems, increased flexibility, scalable, re-factorable (Wikipedai 2007c).

Continual System Verification and Validation Process

At present, techniques for automated verification and validation of complex systems of systems is an undeveloped field. Fortunately, there are several related areas from which key technologies may be leveraged. The first is the Internet, a key example of a successful system of systems, that solves this problem. The second is to examine work surrounding the automated and formal verification and validation of complex monolithic systems. We believe that these seemingly disparate fields will soon be linked through implementation of Semantic Web technologies.

Example 1: The Internet. In considering how the internet performs continual verification and validation, it’s important to consider separate what it does from how it does it. In terms of maintaining the integrity of the information, the Internet's solution is to break, as anyone who has stumbled upon "Error 404: Page Not Found" can attest. Under the hood, however, the internet does have some continual processes running. Two styles worthy of note are the DNS system and OSPF network routing scheme. The 13 root servers at the top of the DNS system are a single point of the Internet infrastructure which if lost would disrupt the entire Internet. On occasion, the servers have come under attack. The continual verification and validation process for these servers is provided by server redundancy and the 118,000 queries per second of normal use.

(DNS 2007) Stability of the internet is provided by the redundancy, and an fault tolerant architecture in which the internet can use cached information to slowly come to a stop giving the system engineers time to repair the break. Another automated verification and validation system is used by the network routers. There are multiple network paths to get from one point to another. The networks use algorithms to pick the best one, and automatically reconfigure when a path breaks.

Example 2: Mechanisms for Formal Validation and Verification. In established approaches to system design, and as illustrated along the left-hand side of Figure 8, present-day procedures for "system testing" are executed toward the end of system development. The well known shortcoming of this approach to validation/verification is the excessive cost of fixing errors. Emerging approaches to system design (Magee 2006, Sidorova 2007, Uchitel 2004) are based upon formal methods and selective use of design abstractions. The new approach benefits system design in two ways: (1) Concepts and notations from mathematics can provide methodological assistance, facilitating the communication of ideas and the thinking process, and (2) Formal methods allow us to calculate some properties of a design. by building design logic into requirements and using formal models for synthesis of architecture-level representations. The goal is to move design processes forward to the point where early detection of errors is possible and system operations are correct-by-construction (Sidorova 2007). The new pathway of development is shown along the right-hand side of Figure 8.

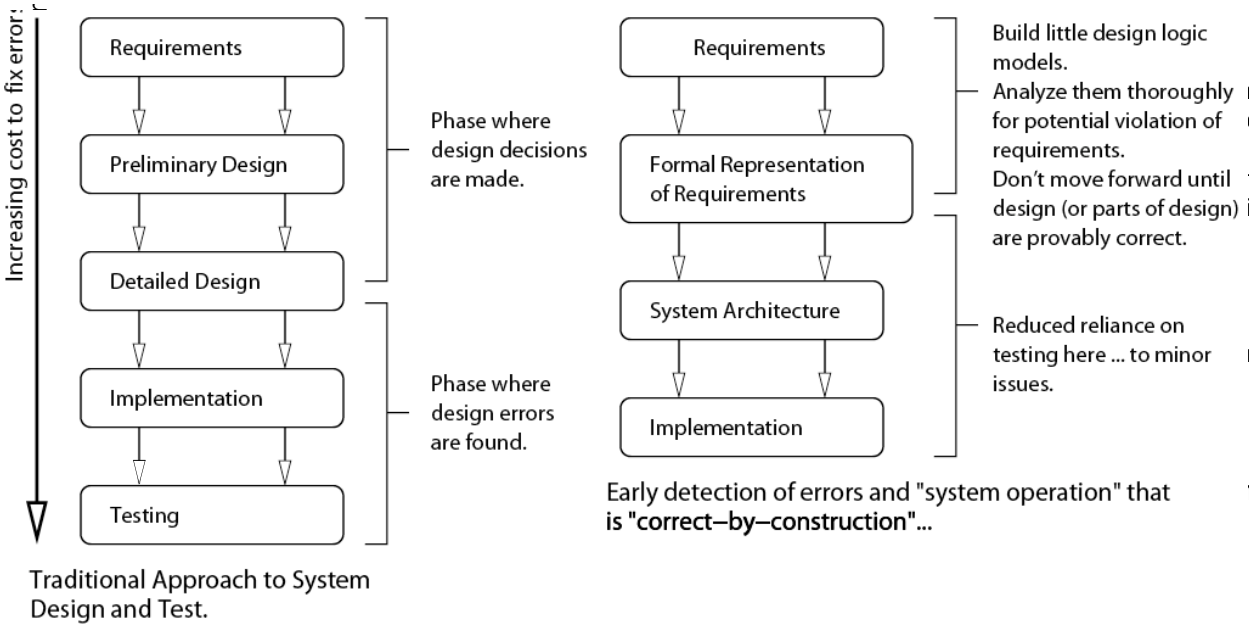

Figure 8. Pathways of Traditional and Model-based System Development (Adapted from Sidorova 2007)

Mechanisms for Formal SoS Synthesis. Ad-hoc approaches to system of systems synthesis are limited by the size/complexity of problems that are tractable. The key benefit in formal approaches to synthesis/validation is streamlining of both the efficiency and long-term accuracy of this process. Each iteration of evolution simply builds on the previous version. The incremental modification of a system of systems design and its formal specification will be much

less work than returning to scratch each time.

Figure 9 shows a framework for system of systems synthesis, evaluation, and managed evolution. The pathway from system of systems goal, scenarios, and requirements to a system-level architecture will be simplified through decomposition of goals and requirements. However, the main challenge lies in system of systems synthesis -- primarily a bottom-up process -- from already existing and operational systems. Each operational system will be described by a formal model covering standards and interfaces (e.g., pre-conditions for operation; required protocols) , simplified models of behavior (e.g., finite state machine model), as well as properties the system supports (e.g., guarantees on safety, progress, and so forth).

The system of systems architecture will support behaviors from the constituent systems, in this case, systems A and B, as well as emergent behaviors. Emergent behaviors can be classified into two groups: (1) those that are needed to fulfill the system of systems goals, and (2) those that need to prevented (so-called side effects) Additional constraints on modeling/communication may be required to prevent the second category. For details on an framework for incremental modification of systems to satisfy these requirements, see the work of Magee, Kramer and Uchitel (Uchitel 2004).

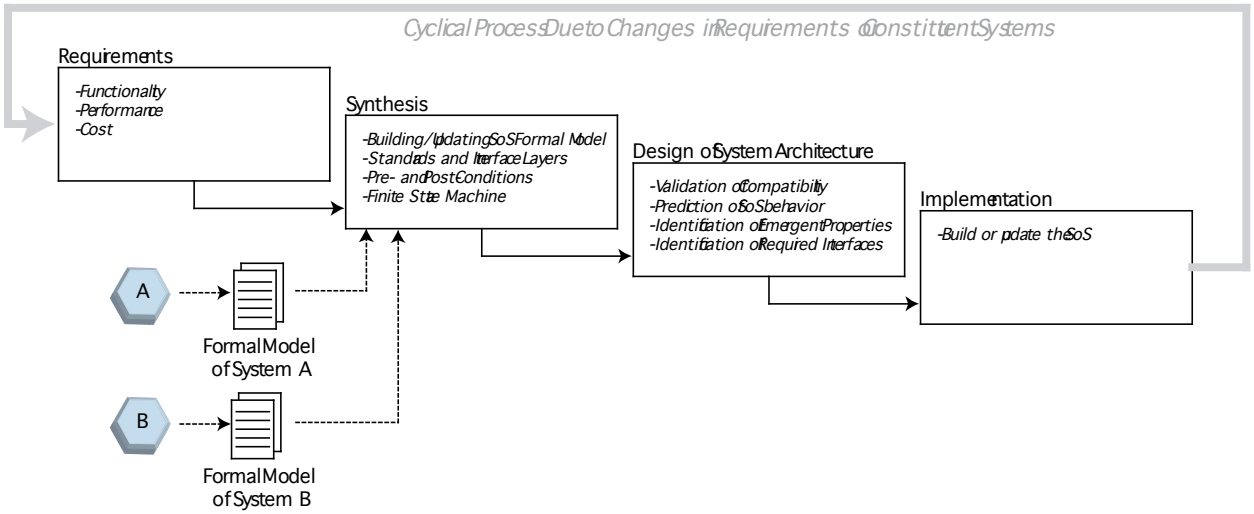

Figure 9. Framework for System of System Synthesis, Evaluation, and Managed Evolution.

Linkage to the Semantic Web Capability. Semantic Web Technologies provide a convenient framework for the representation, gathering, and logical evaluation of system capabilities, as represented in the System Formal Models. Preliminary work in the direction, involving description of problem domain ontologies, and logical reasoning between components has recently been conducted by Austin and co-workers (Austin, Mayank and Shmunis, 2006a, 2006b). This work is currently being extended to multi-level representations of systems.

Conclusion and Future Work

Our work has been motivated by the observation that although system of systems applications are now well integrated into our modern world, formal methods for system of systems design do not yet exist. Nonetheless, some of the systems have been designed well and can evolve easily; others have not been designed well and changes are expensive and risky. Moving forward it is important to adopt design principals which will facilitate change over time.

In this paper we defined an evolutionary system of system architecture as one in which the complexity of the framework does not grow as the system scales and the partner systems do not need modification as the system evolves. To realize an architecture of this type we need standards, interface layers, and a continual verification and validation process. In looking at the industry, we can find several example implementations of standards and interface layers. Lacking in the industry is an automated validate and verification process. This shows the pathway forward for development of this field.

References

Austin M.A., Mayank V., and Shmunis N. Ontology-Based Validation of Connectivity Relationships in a Home Theater System. *International Journal of Intelligent Systems*, 21(10):1111–1125, October 2006.

Austin M.A., Mayank V., and Shmunis N. PaladinRM: Graph-Based Visualization of Requirements Organized for Team-Based Design. *Systems Engineering: The Journal of the International Council on Systems Engineering*, 9(2):129–145, May 2006.

Autonomous Cars 2007. Referenced on November 1, 2007.
See http://en.wikipedia.org/wiki/Driverless car.

Balasubramaniam R., Jarke M.,. Toward Reference Models for Requirements Traceability. *IEEE Transactions on Software Engineering*, 27(1), January 2001.

Berners-Lee, T., 2000. XML and the Web.
Keynote address at the XML World 2000 Conference.

Byron, Dennis. Esbs in 2007: Taking the open source bus to soa (part i of ii). 2007.
Referenced on November 1, 2007.
See http://crasar.csee.usf.edu/MainFiles/index.asp.

Clothier, Julie. 'smart' homes not far away. CNN.com. Referenced on November 1, 2007.
See http://www.cnn.com/2005/TECH/05/27/vision.home/index.html

CORE. See http://www.vitechcorp.com/productline.html. 2003.

Dynamic Object Oriented Requirements Systems (DOORS).

See http://www.telelogic.com/products/doorsers/doors/. 2003.

Enterprise Service Bus. Referenced on November 1, 2007.
See http://en.wikipedia.org/wiki/Enterprise service bus.

Ford kills 'everest' Procurments Software System 2004. Referenced on October 30, 2007.
See http://www.computerworld.com/softwaretopics/erp/story/0,10801,95335,00.html.

Jackson, D., Dependable Software by Design. *Scientific American*, 294(6), June 2006.

Karrenberg, Daniel, DNS Root Name Servers Frequently Asked Questions.
Referenced on November 1, 2007. See http://www.isoc.org/briefings/020/

Maier, M.W. Architecting Principles for Systems of Systems. *Systems Engineering*, 1999.

Maier, M.W., and Rechtin E. *The Art of Systems Architecting*, 2nd Edition. CRC Press, London, 2000.

Sage, Andrew P., Beimer, Steven M. Process for system family architecting, design, and integration. *IEEE Systems Journal*.

Sangiovanni-Vincentelli A. Automotive Electronics: Trends and Challenges. In Presented at Convergence 2000, Detroit, MI, October 2000.

Sangiovanni-Vincentelli A., McGeer P.C., Saldanha A. Verification of Electronic Systems : A Tutorial. In Proceedings of the 33rd Desgin Automation Conference, Las Vegas, 1996.

Selberg, Scott, and Austin, Mark. Requirements Engineering and the Semantic Web.
ISR Technical Report, 2003. See http://hdl.handle.net/1903/6356.

Sidororva N., Lecture Notes on Process Modeling, Graduate Course, Department of Mathematics and Computer Science, Eindhoven University, Netherlands, 2007.

SLATE. See http://www.eds.com/products/plm/teamcenter/slate/. 2003.

TCP/IP Model 2007. Referenced on November 1, 2007. See http://en.wikipedia.org/wiki/TCP/IP model.

Uchitel S., Kramer J., and Magee J. Incremental Elaboration of Scenario-Based Specifications and Behavior using Implied Scenarios. *ACM Transactions on Software Engineering and Methodology*, 13(1):37–85, January 2004.

Biography

Scott Selberg is manufacturing systems engineer for Agilent Technologies and has held that role for the past 11 years. He has a Masters in Systems Engineering from the Institute for Systems Research and a Bachelors in Applied Physics from Yale University.

Mark Austin is an Associate Professor in the Department of Civil and Environmental Engineering, University of Maryland, College Park. He currently holds an affiliate appointment with the Institute for Systems Research. During the past ten years, Mark has taught extensively in the Master of Science in Systems Engineering (MSSE) program, and conducted short courses in Systems Engineering at US companies, and in Europe and South America. He has a Bachelors degree in Civil Engineering from the University of Canterbury, New Zealand, and Masters and Ph.D. degrees in Civil Engineering from UC Berkeley.