Abstract—Complex “system-of-systems” architectures are subject to a myriad of issues arising from the dynamic inter-operability these systems are intended to provide. Many of these issues can be addressed or avoided by considering the messaging interactions between system nodes prior to and during the construction of the component systems. Standardization of messages and interfaces is an ideal way to provide a consistent, vendor agnostic vehicle for interaction and interoperability of systems in this class of complex architectures.

Index Terms—Standardization, system engineering, system integration, system-of-systems (SoS).

I. INTRODUCTION

The concept of a “system-of-systems (SoS)” is quite old. One real-world example of this concept is a city being viewed as a system composed of independent but interacting systems. The electrical, water, and sewer systems that serve a city are all independent systems that provide services to the city. This is the essence of an SoS architecture. The concept of the collective whole (a city) being viewed as a system composed of independent systems (electrical, water, sewer) can be nebulous in this context, but the litmus test of a SoS is that the component systems contribute to the whole through some physical interaction. Another important point is that these physical interactions are provided at discrete points (e.g. a fire hydrant or an electrical outlet) which hides or abstracts the complexities of the independent systems that contribute to the larger whole.

A key aspect of many SoS development projects is the nature of the environment in which the individual systems are constructed. It is common that one or more of the component systems are pre-existing (legacy) and complex in their own right. This means that the concept of connecting these individual systems to a larger collection of systems is both an integration challenge and a potential source of discontinuity to the purpose of the overarching system.

Beyond the technical aspects of the architectures there are many constraints imposed upon the overarching system in terms of competing vendors, inter- and intra-organizational politics and security to name a few. It is extremely beneficial to develop a plan for designing the architecture so that it addresses these anthropogenic constraints prior to the start of the technical design phase of the project.

The remainder of this paper will focus on both the technical and human aspects of SoS architectural planning. The intent of this paper is not to propose an architectural pattern or practice, but instead to illustrate a conceptual plan of execution for the overarching system planning and design that can reduce issues with development by addressing each of these human factors in a technical construct.

II. NON-TECHNICAL CONSIDERATIONS

A. Core Purpose

In the design and development of an SoS, as in the design of any system, it is imperative that there is a clear statement of purpose. Prior to initiating any substantive work on the SoS project, it needs to be clear what the intent of the project, and ultimately the system, really is. No system, regardless of size or complexity, can do all things. It is, therefore, of critical importance to have a scope for the project. It is this scope that will serve as the focus of analysis to determine which pre-existing systems should be considered for inclusion in the project as component systems.

It is important at this time to qualify what a component system is. Each system that participates in the larger SoS is considered a component system if it is directly part of the overarching SoS. Any system that supports a component system, but does not directly support the overarching SoS is a supporting system. Regardless of whether a system is a constituent or supporting system, it can still be critical to the successful execution of one or more capabilities provided by the SoS. In a SoS architecture, component systems can serve the following various roles.

- Core system, which is required for the functioning of the overarching system. This is an infrastructure provider such as a directory service or security provider.
- Primary system, this is a “first-class” system in the overall architecture which fully participates in the operation of the overarching system providing a business capability of the SoS.
- Consumer system, is a system that does not participate fully in the overarching system. This class of system can be considered a “consumer” that receives benefit from the overarching system but generally does not provide to the system.

Of important note here is that any system may indirectly provide to the overarching system, but if that system provides or collaborates through another system, it may not be considered as part of the overarching system (in which case it is opaque to the overarching architecture). These upstream systems can be points of failure to a capability area offered by the SoS.

Manuscript received September 17, 2007; revised April 12, 2008. First published July 25, 2008; current version published September 17, 2008.

The author is with the Department of Computer Science, George Washington University, Washington, DC 20052 USA (e-mail: michael.corsello@ieee.org).

Digital Object Identifier 10.1109/JSYST.2008.925972
B. Organizational Support

The success of any project will depend on the support of participating organizations. In SoS projects, there are generally multiple organizations that participate, many of which may be direct competitors. In this type of project it is not uncommon to have multiple stakeholder organizations that have a vested interest in the system being implemented. These organizations may be competing for additional resources or influence on the direction and scope of the project. Each of these organizations will also have existing systems that support the current means of conducting business that are to be addressed in part or in whole by the new SoS. This, therefore, means that these organizations have an interest in their current systems being adopted as part of the architecture.

For each of the current systems used by participating organizations, there may be a different vendor that produced that system, resulting in direct competition for vendor support of component systems. Each of these systems must be considered at least as candidates for inclusion in the overarching architecture if they fulfill some portion of the requirements covered in the scope of the project.

By maintaining a consistent focus on the scope of the project for the overarching system, it is easier to form quantitative criteria to determine which of these pre-existing systems should participate in the project and in which role. These determinations tend to be political in nature because each participating organization has a vested interest in the adoption of their current system(s) as a component system. This often results in the “marketing” of a system that may only partially fulfill any requirement to be addressed by the new SoS. The quantitative evaluation of systems nominated as candidates for inclusion in the architecture is therefore key to maintaining support from member organizations.

C. Vendor Neutrality

In the construction of an SoS, each system role can be categorized by a set of intended interactions. This set of interactions is the basis for the overarching system interface or application programming interface (API).

Given that there are likely to be many competing vendors, each of which may have products that are fundamentally incompatible, it is imperative that these interfaces are defined in a consistent manner that component systems can be adapted to (via individual integration projects). If these interfaces are designed based upon any specific vendor’s pre-existing system, there is likely to be vendor and organizational disharmony due to these biases. It is therefore important to either adopt existing industry standards from known standards organizations (e.g., IEEE, ISO, W3C) or to develop project specific standards via a collaborative effort with the participating vendors and sponsor organizations.

Of specific note, the creation of these interface standards should include data standards such as encoding and logical data model standards as well. By maintaining vendor neutrality to this extent, each participating system in the overarching system will have to develop an integration strategy to only the interfaces and logical models which can happen in isolation from the overarching group.

D. Organizational Politics

The development of any system will be affected by support from the participating organizations. Even in small projects that support only a single organization, departments within that organization may disagree or compete for resources or representation in the design and development of the system. In complex SoS architectures this is generally amplified many times over. It is not uncommon for an SoS to support not only multiple organizations, but many of those organizations may be multinational or even be multiple national governments. This raises issues of organizational or even national politics over the interactions and sharing of capabilities between these organizations. There are concerns over financial responsibilities, legal or regulatory compliance (and compatibility) and of course security.

In the design and development of an SoS that is constrained by this type of environment, the isolation of individual component systems may need to be considered to support the diverse set of rules that these political constraints may require. In these scenarios, an additional layer of complexity may need to be added to address these constraints. To address this form of political and legal complexity, the design phase of a system may best be served by design governance communities that can evaluate the planned architectures to ensure that each organization’s political and legal needs are adequately represented.

When a system engineering project of any type involves multiple organizations, financial responsibility can become an issue. In terms of a project that spans organizations of varying financial means, such as in a multinational project that is intended to support nations of high and extremely low gross national means, special consideration must be given to reduce the cost of participation. In some cases, this may include the incorporation of long-running transactional capabilities that permit human transfer of data via means such as CD, DVD, removable drive, or tape. The implementation of these means in an SoS construct should be up to each independent component system, but the interaction of these systems may be affected at the overarching level.

To accommodate these types of barriers to entry for participating organizations, each capability to be provided by the overarching system should have a concept of operations that addresses these organizational limitations. It may be directly addressed via financial support of shared technology infrastructure being provided by a weighted “means to pay” participation schedule. These forms of financial considerations may be crucial to the success and longevity of the project and must be considered from the inception of the project.

E. Security

In the development of multi-organizational systems security is a critical factor to be considered. Each participating organization may have limited trust of the other organizations. This may not be a limited trust of the organization as a whole, but instead be a limited trust as it pertains to the member personnel, data, or business processes and methodologies codified in the systems being constructed. In this way, each organization’s security requirements will have to be addressed individually in their own systems, in the component systems of the architecture and in the overarching SoS core architecture.
Aspects of Security: System architecture is subject to multiple aspects of security including the following:
- authentication, is this actually “who” they claim to be;
- authorization, is this “entity” allowed to:
  - access, a specific resource, data, or otherwise;
  - perform, a specific action, such as executing code;
- non-repudiation, this “entity” requested this access or activity at this time with these consequences.

These aspects are the basis for security considerations and may apply to multiple sections of the overarching system as well as to the component systems in the architecture. In an SoS construct this can have significant ramifications as to the control and partitioning of authority for the designation and maintenance of these permission sets.

Trust: Expectations of trust between organizations within the context of a system (or an SoS) is no more realistic than in any other context, so trust will tend to become a key factor in the design of security features and in the segmenting of system capabilities in the interactions of component systems.

Enforcement and Authority: An important consideration in the implementation of security is the realm of enforcement and the definition of who can enforce and where enforcement can be conducted. It is common that security will be isolated within each component system and a set of security capabilities will be additionally applied in the overarching system. The design considerations and politics of which are too complex to cover in this paper.

III. TECHNICAL CONSIDERATIONS

SoS architectures are generally very complex and bring together a collection of new and existing systems within a loose framework of overarching technologic capabilities to fulfill a larger set of requirements.

As in all system engineering projects considerations have to be made in regard not only to the capabilities required, but also to the capacity required for each capability. This may be realized through the need to host multiple instances of a specific system type or (capability class system). In the case of any distributed form of system, the overall capability of a SoS is hosted on multiple nodes, each of which will host one or more component systems (or capabilities).

When transforming to an SoS an increase in demand for each component system should be anticipated as a result of the larger community, this may result in a need to increase the number of nodes for each component system.

A. Component Systems

In the construction of an SoS, each of the individual systems that participate constitutes some portion of what is required to fulfill the overall requirements. It is not uncommon also that there are multiple individual component systems that all provide the same capability, potentially from different vendors with slightly different algorithmic implementations. When interfacing these component systems into the SoS construct, it is important that they all connect via a common set of interfaces provided through the overarching system.

1) Legacy Systems: Each legacy system that participates in an SoS may in fact contribute a unique capability only fulfilled by that specific system. To accommodate the increased demand for that capability, which results from expanding the user community to the full SoS user base, additional nodes may be required. In the determination to include such a legacy system in the overarching architecture, it is important to consider the strategy by which such nodes are added. Questions to consider in this scenario include the following.

- Inter-nodal activity, was this legacy system constructed to support additional nodes for the supporting systems this system relies upon?
- Platform support, is this legacy system supported on equipment that is still available?
- Source access, is the original source of this system available for consultation to address scalability needs?

The legacy systems used in an SoS architecture will often provide the backbone of the initial operating capability (IOC) when the overarching system goes live. It should, therefore, be anticipated that it will be necessary to expand these legacy systems to accommodate additional capabilities and capacities as needed. These expansions should be planned in advance to ensure operational capabilities and capacities are always available when needed.

2) New Systems: In the case of the development of new systems to provide some required capability, there are still the traditional constraints imposed by the hosting organization’s infrastructures. These constraints will need to be addressed during the development of those systems and may result in limitations to the deployment of multiple instances of this system to different hosting organizations. This underlying system infrastructure constraint may result in increased cost, time, and development efforts to support multiple platform implementations of a capability to achieve the desired capacity.

One of the aspects that favor the development of new component systems to support an SoS is the lack of imposed constraints on technical implementations other than environments. This enables new systems to follow the current industry’s best practices and the specific guidance of the project architecture without the need for a specific integration layer to connect the system to the overarching architecture. Even in these systems, however, it is a best practice to still abstract the interfaces and interactions with the overarching system to enable external changes to the requirements to be implemented as close as possible to where they occur.

B. Core Capabilities

The SoS is defined as such by the fact that each of the component systems, regardless of role, has some interaction with one or more of the other component systems in the overarching system. This basic requirement indicates that there is generally some amount of arbitration or orchestration at a minimum that must be provided. All of these infrastructure requirements are provided as core capabilities through core component systems.

In architecting an SoS, each of the requirements for infrastructure support will be bound to one or more of these core systems. It is generally best to architect core capabilities in the most flexible, adaptable, and distributed manner to ensure the greatest level of scalability for these systems. This often means that these core capabilities are mapped on a one-for-one basis...
to a system. It is also generally best to leverage existing standards for these systems especially to maximize the likelihood for current support from existing component systems while minimizing the likelihood of vendor biases.

Common core requirements will include capabilities like the following:

- login/authentication services;
- directory services, this is a broad category that includes organization directories, vendor capability directories, system directories as well as component or services directories;
- data services, it is not uncommon that “cloud-based” data services will be required to facilitate the dissemination of information to other services. This may actually be a pass-through service to other provider systems;
- generic brokers, are services that provide a central location to request some specific service that is then arbitrated between actual providers of the service. An example of this capability may be a round-robin load balancer;
- informational services, are generic services that provide a catalog of informational constructs such as service schemas, system bindings, etc. These services are quite diverse and may include custom capabilities unique to the domain of the overarching system.

C. Provider Systems

All component systems that are not providing core (infrastructure) capabilities are considered provider systems. This is the majority of systems in the overarching system. Any functional business capability that is offered by the overarching system is provided via a provider system. These provider systems may themselves be distributed to provide an enclave of capability that is exposed to the overarching system as a single entity (as in load balancing), or distributed as distinct nodes of capability that are integrated by the overarching system. Each of these patterns of distribution for a capability has different impacts on the larger system design, including monitoring of health and capacity and more significantly, data synchronization across nodes.

If the nodes that provide a capability are exposed separately, it is possible (as noted earlier) that each node may have a specific implementation that will yield a slightly different result when used. In this circumstance, the adhesion of a client request to a specific node implementation may be of significant importance for valuation integrity. This is accomplished through algorithmic implementations of the core services, but needs to be accounted for early in the architectural design phase.

Each hardware node may provide multiple capabilities to the overarching system, potentially including core capabilities as well as functional capabilities. In this case, it is important to know the hardware bindings of capabilities for operational assurance as covered in Sections VI and VII of this paper. If a single hardware node is implemented in a load balancing scenario, multiple physical nodes can act as a single node to provide greater capacity and reliability for the services hosted. However, accounting for this configuration is difficult for management interfaces and may affect the overarching system operation if not adequately accounted for (e.g., sessions for long-running transactions managed externally of the node).

When constructing provider systems consideration must be given to all aspects of the potential environments these nodes will be hosted in. This includes all aspects of hardware configurations, network defenses, systems management at the node level, availability (especially in poor regions of the world), legacy software bindings, software implementations, etc. The construction of a distributed system is always difficult and the addition of the dynamic environments often extends these difficulties for SoS architectures.

IV. STANDARDIZATION

In the development of an SoS it is critical that all inter-system communication is standardized to facilitate integration of the component systems. Many times, the overarching SoS will be based on standards that conflict with the standards already implemented by some of the component systems. Or put another way, it is likely that the component systems will be based on established standards that are different but overlapping. The overarching system will have to adopt a single standard for each interaction or set of interactions that all component systems will have to comply with. This is both a technical and nontechnical issue, as the existing standards may be fundamentally incompatible, but must be adapted to interoperate.

The standardization of a system’s interactions should be based upon established, open, supported standards. It is often difficult to find standards that can be adapted to suit the overall needs of the larger SoS and the existing domain specific standards already in use by the component systems. This is a problem domain that is not directly solvable, but instead is worked around through careful and thorough design. In this area, the standardization of message interchanges and data models is the key to overall success.

A. Standardization Areas

Some of the fundamental reasons for standardization in any system include the following:

- upgrade paths enable the upgrade of systems and standards between versions in a dependable, repeatable fashion;
- extensibility, of the systems and standards to accommodate changing requirements;
- customization, of the implementations participating in the system (Black-Box);
- translation, by external organizations and vendors of both data and programmatic messages;
- integration, of third-party systems, components, and data now and in the future.

To accomplish these goals, the standardization process must be well thought out to avoid these following common pitfalls:

- complexity, of data types and procedures that result from standards developed at a high-level;
- evolution, of requirements and technology over time which will likewise require additions and extensions to standards;
- perspectives, of the communities leveraging the standards;
- concepts need to be translatable.
In order to achieve success in an SoS architecture the standards used need to address all of these areas. A list of domains that may be standardized in this manner includes the standardization of the following:

- logical data models, including primitive data types, entity types (e.g., tables), and complex structures (e.g., graphs, trees);
- business domain perspectives, or profiles that map logical data types to a domain specific data type for each domain;
- logical associations, including rules on how logical data types can be composed to create business data entities;
- rules, including how data can be converted between logical representations and how to represent a vendor specific binding;
- encodings, how to physically represent information in a file, stream, or opaque entity.

V. INTEGRATION

The foundation of any distributed system is based upon the integration of information in some form across system boundaries. This integration can be accomplished simply through procedure calls, or specifically through the marshalling of information used as inputs and outputs of remote procedure calls. Or, in a more complex environment this may include the integration of full data sets in some form between systems. A relational database management system (RDBMS) is a simple example of this complex form of interaction, where data is stored in the RDBMS and marshaled into and out of the RDBMS to externally distributed systems that use the stored data. The implementation of most RDBMS applications places the responsibility of translating (or encoding) the data into a form the RDBMS understands on the client system. This is a vendor-specific binding for that RDBMS application. In many SoS architectures, this type of application of technology would not be permitted due to vendor-specific encodings on the "wire." Instead, in this form of architecture, the data could be translated into an intermediary form that is standardized, such as XML for transmission between nodes. This is the approach taken by many service-oriented architectures (SOA) including some that are also SoS architectures.

A. Data

Data integration is the most complex challenge to integrators. Data can be voluminous in size which poses transmission constraints. Data is structured differently for different purposes which pose compatibility issues and it is often complex in structure which causes access issues. The ultimate goal in a system is to process data to produce information which a user can comprehend as knowledge. This makes the integration of data an important part of systems interactions. All system interactions will involve some amount of data transfer, which will often involve a need for data integration. There are multiple types of data integration that can be used in complex systems such as the following.

- Association, which has the following two basic forms.
  - Overlay (see Fig.1), data from different systems is not combined in any fashion. Instead the data is displayed side-by-side from the different systems. Updates may or may not be allowed. This avoids all complexity of data interactions. As an example, data from system 1 could represent real property assets such as land parcels and data from the local system could represent logistic assets such as vehicles. In the overlay scenario, the two sets of data are represented as distinct and only overlaid contextually. In a further example, data from both systems could represent information about sales from two companies. In this latter case, the information represented is similar in nature and context but treated separately. In the overlay scenario, these data sets are not integrated in any way, even though in the sales example they may have some common customers.
  - In-lining (see Fig.2), data from different systems are combined into an integrated view of information, but each source system’s data is treated as distinct. Updates may or may not be allowed. This avoids all complexity of data interactions other than compatibility of data models. An example of in-lining can likewise be sales data from two companies. In the case of in-lining however, the data is interspersed such that records from each system are simply displayed together in a common picture. In this scenario, the data models must have enough in common to allow the data to be displayed together (such as both data sources having a name and description fields).
- Integration, which has the following three basic forms.
  - Interleaving (see Fig.3), data from different systems are combined into an integrated view of information, with corresponding data entities from each system matched to provide a view of the authoritative source of each record. Updates are allowed locally but may or may not be allowed.
be allowed back to the source. In this scenario, updates are generally local in nature and the updates are persisted as a delta from the source. This avoids the complexity of data interactions required for dynamic updating.

An example of interleaving would be the display of parcel data from system 1 interleaved with the parcel data from the local system, with the capability of the users of the local system to make changes to the parcel data from either system. In this manner, the changes made to the data from system 1 are not posted back to system 1, but instead only persisted in the local system for local system users to benefit from.

— Dynamic (see Fig. 4), just as in the interleaving form, data is integrated from multiple sources; however, updates are not only local but transmitted to the source data system. Additionally, inconsistencies between data sources for matching elements may be reported.

An example of dynamic data exchange will extend the previous example of parcel data from the two systems, with the added capability of posting all changes back to the originating system. In this way, a change to a parcel originating in system 1 by a user at the local system will be posted back to system 1 for all users to benefit from. However, there is no posting of data originating in the local system to system 1.

— Synchronization (see Fig. 5), all data from all systems are treated as a single source with no distinction as to the originating system. All updates are transmitted in real-time to all partners. This is a complex process that addresses the issues of data interactions. In full synchronization, all updates are critical in time. The operation of changing data can have cascading effects especially if multiple synchronization partners are involved with different areas of responsibility for subsets of data.

An example of synchronization would be the generation of parcel data from each of systems 1 and 2, with the local system integrating all records from each system and allowing changes to all data to be posted back to both systems 1 and 2. In this example, the data that originates in the local system is also posted to systems 1 and 2. In this way, the data in system 1 is equal to the data in system 2 and in the local system.

There are many interesting complications in this model as agreements for the individual systems may not permit the exchange of information between the individual systems. In the parcel data case described before, systems 1 and 2 may not allow their data to flow through the local system to each other, instead requiring the local system to maintain a pair of synchronization processes to ensure that the data does not flow across system boundaries.

B. System Interfaces

Other than the data itself that is exchanged between systems, the interfaces for calling procedures and requesting informational resources is the primary means of interaction between component systems. The integration of systems occurs at the exposed system interfaces as this is all that is available to external systems. The interfaces may be exposed through one or more protocol, such as the WS-* stack for web services using SOAP with bindings to XML over HTTP/HTTPS. The bindings may be either generically textual like in the XML over HTTP mechanisms or explicitly binary as in the use of CORBA, Java RMI, .NET remoting, etc. The design and development of these system interfaces should take into consideration multiple aspects of the interfaces themselves, including the bindings we just covered.

Interface Considerations: When planning for SoS integration at the interface level, it is a good plan to anticipate a requirement to support multiple interface bindings for each interface definition. In this way, a single system can have an internal implementation with multiple exposed bindings, or an orchestration system can provide proxy bindings to enable compatibility between incompatible systems. This is an example of the concept of least capable design, where interfaces are designed in such a way as to maximize the bindings that can support the interface. This is further supported via encoding standards, rule standards, and business perspective standards.

VI. EXTERNAL SYSTEMS

In SoS architectures, there will often be external systems that simply consume the services of the overarching system. These external systems are considered clients of the system and bring their own concerns.

A. End Users

Human users of an SoS may or may not ever directly interface with any node in the SoS. Instead, human computer interactions (HCI) may be accomplished via an externally connected system such as a web server via a web application. With the development of smart client technologies, which blend lightweight thick client applications with back-end systems providing the functional services, it is entirely possible that many HCIs will be directly conducted with a service node via a smart client.

In any manner of interaction, human end users need to perceive that they are being provided value from the system and
that the system is responsive to their actions. This pair of requirements may tend to blend the SoS architectural designs with more traditional user level application designs at these boundary interfaces.

B. Clients

Clients of an SoS may not be required to conform to the same set of interfaces as the constituent provider systems. In this manner, the public facing client interfaces are often less rich and more diverse. Clients that are to be supported may have user interface considerations, data latency, or transaction time constraints and generally impose a different set of requirements on the overarching system. This is often handled by specialized clients that serve as service brokers for the clients at large. This added level of indirection can impose additional legacy impacts that need to be considered in the design phases to ensure these latencies are accounted for.

System interface brokers that support client interactions to an SoS will have to additionally serve as the focal point for authentications and many other security aspects of the system. This can result in a need to scale these broker systems more rapidly than expected in the design phase due to user loads. The cost of such brokers will have to be considered in terms of which organizations host the broker systems for a given client community, thereby incurring the costs. Considerations also need to be given to concepts of caching and integration of results at these client broker sites. Finally, decisions will be made in the design phase as to how clients can access the overarching system. The broker systems may be the solely permitted client access points to the system. Or, no brokers may be used at all, in favor of a discovery service that is published at a specific resource node that indicates where systems are accessible. The latter may affect the mobility of nodes if not further abstracted across the system domains.

C. Dependent (Opaque) Systems

Systems outside of the primary SoS boundary that are required for the successful operation of one or more nodes in the SoS (e.g., a database server, LDAP server) are still considered external systems that the overarching system indirectly depends upon. This class of dependent system is upstream to the SoS provider nodes and may result in single points of failure to the capability provided by a supported node. These dependent systems are often overlooked in analyzing the supportability and dependability of the overarching system. In many cases, this is due to the core design group(s) having lack of access to the individual organizational units that are supporting the physical systems that in turn support the overarching system. Since these systems are opaque in terms of access from the SoS as well as to the organizations involved in the design and development of the system, these systems are of special concern to designers.

While full accountability in an architectural design is desirable, the organizational politics and general logistics of acquiring such information are often unrealistic. Instead, to ensure that critical supporting systems are maintained and available as needed, service level agreements are used to ensure whatever is not seen by the system design will still not affect the availability of the overarching system. This form of assurance is still far from complete as it is enforced via levying fines or through other contractual means which may still result in extended downtimes for the system users.

D. Availability

The primary means of ensuring availability of system capabilities is via redundant nodes that offer redundant capacity of some measure beyond peak demand. This concept of course comes with additional cost for maintaining this continuity of operations (COOP) capability, but will tend to alleviate the problems that may arise when a required capability becomes unavailable. In addition to providing a redundant capability, this capability should be adequately separated by geography to ensure natural disasters would most likely not disrupt all nodes providing the specified capability.

When designing for COOP it is important to consider the impact each decision made. It is not uncommon to implement COOP as a manual process such that when a node fails, users are expected to map to the redundant node manually with an expected latency in data interactions that may result in a loss of up to several days’ worth of data if a failure occurs. In many SoS architectures, no system failure is allowed to impact the operation of the overarching system, including no tolerance for any data losses. This greatly affects the design of the component systems to ensure that all nodes can interact and exchange information in near real-time so that data is committed to multiple nodes prior to being considered stored. This is a large design consideration that must be made prior to construction of the system nodes and must be considered for supporting legacy systems as provider or upstream dependent nodes in the architecture.

Mechanisms such as coherent caches, canonical sources or chains of custody may provide a timely solution to the data distribution and integration challenges arising from dispersed storage nodes. This is a fundamental area of research that poses tremendous challenges to distributed systems in general.

VII. SoS INFRASTRUCTURE

The management of complex systems that are loose aggregations of other independent systems is a daunting task. Moreover, there is often no one organization overseeing the system at large as it is not physically a resident system anywhere. Instead, management of the infrastructure of these systems is generally accomplished on a nodal basis, where each node is managed individually by the custodian of a specific node. This is cost effective but lacks a central oversight to visualize the availability and health of the overarching system. The area of distributed system management is still being actively developed by many communities and progress is being made.

One means of managing this type of complex system is through the use of agent services that communicate the availability of node networks and continually analyze that information. This is accomplished via both statistical and genetic algorithms hosted on various core nodes in a system. Projects like Globus, JXTA, BOINC, and groups such as the Global Grid Forum are advancing these technologies for inclusion in complex distributed architectures, including ones that are based upon an SoS approach.
A. Considerations

In order to manage an SoS it is important to know the status and health of each capability and the capacity utilization of that capability more so than to know the status of the nodes providing that capability. This is a paradigm shift from traditional system management where it is the machines that are managed and not the capabilities they provide.

When a system is composed of nodes that each provide some unknown number of capabilities (perhaps beyond that available to the overarching system), it is difficult at best to measure the health or availability of the capability per machine or in total. Instead, a heuristic method is required that essentially learns the current capability and capacity with respect to the variations of that availability. This form of analysis can then be deployed as a capability on each node that will accept such analytical components. This analytical component can then isolate the variations of availability to arrive at a continual metric of health. This form of distributed analysis should yield better results with larger networks and as such be a standard distributed mechanism for systems monitoring.

The management of the nodes providing a capability will in general not be solved via technologies (that is already done well) instead, node management will be a responsibility assigned through policy. In simple terms, the owner of each physical hardware resource will determine who will be permitted to manage that resource. Many organizations are not willing to permit management level access to personnel not affiliated with their organization.

VIII. ADDITIONAL CONSIDERATIONS

Due to the multi-organizational participation in SoS projects, there are additional considerations that must be addressed.

A. Non-Technical

The organizations that participate in an SoS project will generally each have their own vendors. It is also common that vendors may be involved in this class of project directly as well. In this case, there are competition issues that need to be addressed. Generally, the vendors may be reluctant to adopt and reuse systems and system components that were developed by another vendor. This may be due to a perceived loss of revenue from the development time, or due to a lack of trust toward the other vendor(s). In either case, these types of issues need to be addressed early on in the project. One simple way to address these concerns is via group adoption, where no single organization can make any decision without approval of an oversight group. This oversight “red tape” comes with its own costs.

Vendor biases can also be seen in other organizations through the “not-built-here” or “not-bought-here” syndromes. In these cases, an organization simply will not allow their developments to rely on components they did not build or purchase. This is a difficult concept to overcome, as this is an emotional issue rather than a technical one. It may be possible to address this type of consideration through group approvals as well.

Deadlines are a fact of life in all projects. In SoS engineering projects, these deadlines can make or break the project as a whole. Some simple guidelines that can help a project to be successful include the following.

- Small scope, for each item planned. If a deadline is missed, it is on a small unit of work with lesser impacts.
- Small teams, for each item planned. If a team is small that works on an item there is less likelihood of disagreement in the team.
- Under sell, never promise more in a single work unit than what is required.
- Plan ahead, the more future units of work that are planned, the fewer problems should arise.

As in any multi-organizational project, distribution of labor is a consideration. When varying levels of financial availability are also a factor, the distribution of labor can be additionally impacted. It is, therefore, beneficial to plan labor based upon unit of work assignment and available funding. When financially constrained organizations are the ones with a primary responsibility in a functional domain, concessions will have to be made to ensure these organizations can fulfill their responsibilities. This type of issue is a possibility; a plan should be devised in advance to handle such problems if they arise. This may be an external pool of funding such as “membership dues” for organizations or designations of “pay for service” where all prospective clients of a capability are required to pay a portion of the development costs.

B. Technical

In any SoS project, there will tend to be multiple platforms being used. This includes both hardware and software platforms for nodes and networks. As such, there are considerations that need to be made throughout the design of the overarching system to ensure that any reasonable set of technologies can be supported in the system. While there are simple decisions that can be made to support this goal universally, these decisions also have to take into consideration the performance and cost impacts.

Cross-platform and cross-language inter-operability is a major consideration for SoS architectures. If standardized interfaces are well designed, this may be alleviated in the overarching system, but may still be an issue within the component systems. Additionally, for performance reasons, some inter-node communications may be designed to operate “outside” of the standard interfaces. In these cases, all participating platforms may need a custom set of bindings or APIs to enable this capability. If this is done, then those specific bindings should be standardized in addition to the regular APIs in use.

Documentation is a key factor for technical success and longevity of a complex system. Given that most SoS projects will take many years to complete, the change in personnel and organizations involved necessitate the generation and maintenance of good documentation. This documentation must be produced throughout the course of design and development. Once produced, all documentation must be distributed such that it is available and discoverable for those who need it.

IX. CONCLUSION

An SoS is a form of distributed architecture that commonly leverages existing systems to provide functional capabilities that the overarching SoS will then expose externally. This form of architecture is often intangible and will tend to be
more constrained by political and organizational considerations than other forms of distributed systems. The development of these systems requires a full complement of systems engineers, business managers, and project management black belts to be truly successful.

As with any systems engineering project, SoS engineering requires comprehensive planning to form an adequate design that can be realized as an effective system. Additionally, an SoS can be of even more benefit than smaller systems through the use of parceled and well-planned standards.

Overall SoS architectures are about providing extended value through the leveraging of multiple organizations’ existing resources, often including their legacy systems to provide key-stone capabilities.

REFERENCES

Michael A. Corsello (M’08) has 13 years of experience in systems architecture, design, and development specializing in geographic information systems (GIS) and distributed computing. As a research biologist, he designed and developed ecosystem-level geospatial modeling solutions. He has since worked on national scale development efforts and various research projects. He currently serves as a Software Architect on a multi-service distributed DoD application project and is a lead modeler for the Spatial Data Standard for Facilities, Infrastructure, and Environment (SDSFIE).