Rich Services: Addressing Challenges of Ultra-Large-Scale Software-Intensive Systems

Barry Demchak, Vina Ermagan, Claudiu Farcas, Emilia Farcas, Ingolf H. Krüger, Massimiliano Menarini
University of California, San Diego
{bdemchak,vermagan,cfarcas,efarcas,ikrueger,mmenarini}@ucsd.edu

ABSTRACT
Ultra-Large-Scale (ULS) Software-Intensive Systems are the new frontier for Software Engineering research and development. The requirements space for ULS encompasses, in particular, the system-of-systems integration challenge. This challenge is characterized by the inherently distributed nature of the constituent systems and the integration solution, the typically interdisciplinary background from which these constituent systems emerge, the need to address a broad spectrum of crosscutting concerns as part of the integration fabric, and a significant amount of agility both in the integration architecture and the development process to accommodate changing requirements over time. In this position paper, we discuss some of the consequences we see emerging from this challenge, propose an architectural blueprint for addressing them, and present future research directions.

Categories and Subject Descriptors
D.2.11 [Software Architectures]: Domain-specific architectures. K.6.3 [Software Management]: Software Development

General Terms
Design, Economics, Theory

1. INTRODUCTION
The interesting systems we design and build today are, for the most part, Ultra-Large-Scale (ULS) Software-Intensive Systems (ULSSIS). The added value of a ULSSIS emerges from the interplay of a set of constituent systems, each element of which is typically a fully functioning system by itself. The Internet acts as the enabler for the composition of the subsystems into a coherent, integrated network of capabilities that, together, make up the ULSSIS. Establishing and managing the development practices by which these integrated systems come about is a fundamentally new challenge for Software Engineering; the focus shifts from the production of components (open or monolithic) to the choreography of components or the services they offer.

As one example of a ULSSIS, we can consider the recently launched Ocean Observatories Initiative (OOI) [6]. The OOI combines oceanographic instrument and sensor networks, data and computation grids, and a broad set of end-user applications with novel capabilities for data distribution, modeling, planning and control of oceanographic experiments. The resources of the OOI are distributed both physically and virtually among different organizations, each with their own policies for resource access and data delivery or consumption. Furthermore, the proposed OOI system will enable virtual organizations to be formed by combining existing and emerging resources; the products of these resources can then be consumed by yet other (virtual) organizations.

The set of stakeholders is vast, ranging from sponsors to ocean scientists to architects and implementers to operators and maintainers to the general public. Similarly vast is the number of ways users will want to interact with the system, based on their domain-specific backgrounds, needs, existing systems and applications.

This example already illustrates key challenges for ULSSIS design and development: How to capture and manage the requirements for the integration solution given the vast set of stakeholders from many different scientific and operational domains? How to identify and enact policies for access to and distribution of data and computation resources in the composite system? How to integrate policies of participating organizations with the policies of the integrated systems? How to ensure safety, security, availability, fail-safety, and other quality properties without access to or control over the constituent subsystems? How to respond to changes in requirements over time – both during design and at runtime? How to maintain a viable integrated system while subsystems and their capabilities may come and go over time?

We see these challenges across many domains: massively distributed, cyber-physical systems1 emerge in the public utility, safety, homeland security and defense domains – but also in traditional domains such as automotive, avionics and plant control – where sensor/actuator networks are coupled with a wide variety of information systems via the Internet to increase situational awareness and to support decision making processes. Such applications place tremendous demands on their underlying cyberinfrastructures in terms of security, trust, dependability, safety, flexibility, and other quality properties. In each case, these integrated systems are built by composing existing systems, which are developed and/or governed by different organizations. Logical and physical distribution lead to a high degree of complexity that requires new methods and tools for architecture design and implementation of the overall system integration.

In the past few years, service-oriented architecture (SOA) and design have emerged as techniques for facilitating system-of-systems (SoS) integration. It is our observation, however, that this trend has actually lead to a substantial fragmentation of the inte-

---

1 A cyber-physical system combines physical entities and processes with an underlying computational and data fabric (the cyber-infrastructure) that enables access to, observation and processing of, and control over the integrated system.
integration task [1]. Specifically, in the realm of Web services, all crosscutting concerns, such as authentication/authorization and failure management, are delegated to separate standards with little concern for their re-integration into a consistent whole.

Consequently, the development of integration solutions for ULSSIS becomes a new, challenging, interdisciplinary process. SoS integration requirements range from the well-known data-/application-/process-integration to the less understood integration of heterogeneous trust, security, and privacy zones.

The remainder of this position paper is structured as follows. In Section 2, we will discuss the requirements space for ULSSIS in more detail. In Section 3, we will introduce an architectural blueprint we have developed to tackle the integration challenge. In Section 4, we will discuss future research directions that we believe to be crucial for ULSSIS development.

2. THE INTEGRATION CHALLENGE

The integration challenge for ULSSIS is multi-faceted. For the purposes of this paper, we concentrate on the following three requirements dimensions: (1) Distribution and scattering of functionality, (2) Multi-disciplinary stakeholder communities, and (3) Need for agility and flexibility. Of course, even this set of dimensions is non-orthogonal; the difficulty of finding an orthogonal coordinate system for the requirements space of ULSSIS is one of their characteristics in the first place.

Distribution and Scattering of Functionality. ULSSIS emerge from the interplay of their constituent subsystems. Each of these subsystems contributes only a part to the functionality of the overall system. This induces the inherently distributed nature of the resulting ULSSIS – be it at the conceptual (logical) or physical (deployment) level. Functionality is literally scattered across the distributed subsystems. An obvious consequence is the need for explicitly specified and well-documented interfaces for each of the subsystems – and for the integrated system itself if it is meant to be an open system projecting its services to its environment.

The development of real-world systems (such as the OOI mentioned above) requires to establish many quality properties as part of the integration fabric rather than relying on the capabilities of the constituent systems. Even if one of the subsystems has capabilities that the integrated system should have – typically, those capabilities cannot be easily or economically injected directly into other subsystems. Rather, it is at the level of the interplay/ choreography of the subsystems that the scattered functionalities are composed and crosscutting concerns can be addressed.

Another consequence of the distributed nature of ULSSIS is the inherent need for scalability. By adding an increasing number of subsystems into the integration mix, the cost of addressing crosscutting concerns at the subsystem level is increasingly prohibitive.

Hence, one of the key challenges to building ULSSIS is to minimize the effort of integrating and reusing existing systems, while providing new functionality and still meeting the requirements of robustness, policy, performance, and maintainability. One example is failure management: When complex networked systems with thousands of scattered functions are assembled into an integrated ULS, failures become the norm [5] and, thus, need to be managed system-wide. This, however, requires building a comprehensive understanding of all of the different capabilities of the subsystems, their failure domains and mitigation strategies, as well as substituting for lack-thereof by introducing failure management capabilities into the integration fabric [3].

Multi-Disciplinary stakeholder communities. ULSSIS generally service large numbers of stakeholders, and each stakeholder brings its own business processes, capabilities, and requirements. As the number of stakeholders grows, so typically does the number and complexity of business concerns (e.g., governance, security, and failure management), as does their level of distribution across the overall system’s architecture. As each organization has its own boundaries and policies, the access to resources in these organizations must be governed. Of course, this argument also extends to the composite system – policies for the integration solution must be established and enacted at runtime. The challenge is how to do this in a uniform way so as to leverage policy monitoring and enactment across the overall architecture rather than by providing ad-hoc solutions for all end-points.

Moreover, driven by the needs of stakeholders to interact to accomplish their tasks, there is an increasing desire to enable the formation of virtual communities of interest (VCOI). These VCOIs define themselves by shared computational and data resources, common agenda, communication needs, and other collaboration characteristics, and also by common policies and workflows. Hence, a typical ULSSIS not only needs to provide the core functionalities for these VCOIs to perform their tasks, but also to manage the creation, existence, dissolution of and membership in these VCOIs. This is particularly important when there is not a single organization that has all the resources needed and all stakeholders in collocation. The OOI is again a good example: many investigators belonging to a wide range of organizations might share resources for specific scientific experiments, thus forming a VCOI over the integration fabric provided by the OOI.

Along with the diverse stakeholder set comes the need to cater to their existing applications, existing and emerging user interface needs, and coping with a wide spectrum of underlying technologies – ULSSIS are inherently heterogeneous in both stakeholder set and associated technology bases.

Need for agility and flexibility. ULSSIS, because they comprise a variety of subsystems, are inherently subject to the effects of requirements changing over time. Not only do the requirements for the integrated system change over time in both short-lived and long-lived ULSSIS, but the requirements for their constituent subsystems also evolve independently from the composite. This requires continually adapting the integrated system so as to avoid negative feature interactions [7] in the integration solution. In essence, this concern affects not only the requirements gathering and design process, but also the architecture for the integrated system. On the one hand, the pertinent requirements for all the subsystems and the integration solution must be understood, updated, and transitioned into the architecture. On the other hand, the more flexible the architecture is with respect to updating or substituting existing subsystems, the more reactive the ULSSIS will be to changes in the environment in which it operates. This is particularly important in the context of dynamic system reconfiguration, as such flexibility is needed for self-healing systems that recover gracefully from failures or attacks.

This extends to the integration fabric as well, because for long-lived ULSSIS, the technology base on which this fabric is built will change over time. Closely related to this is the challenge of disentangling logical architectures from deployment aspects and associated technologies. To manage problem complexity, a ULSSIS development approach needs to provide methods for decomposing complex problems according to separate concerns, provide flexible encapsulation for these concerns, and generate a conceptualization that can be easily leveraged into a deployment.

In fact, many emerging ULSSIS rely on the opportunistic, serendipitous integration of existing and emerging applications or
The evolution of the integrated system.

The challenge for ULSSIS lies: composition of the services projected only a sliver of the ULSSIS integration requirements found in practice. However, they point into the core direction of where the challenge for ULSSIS lies: composition of the services projected by the constituent, distributed subsystems while addressing crosscutting quality concerns and providing the necessary flexibility for the evolution of the integrated system.

### Summary

The three dimensions of distribution, stakeholder, and agility needs discussed in the preceding paragraphs address only a sliver of the ULSSIS integration requirements found in practice. Fortunately, they point into the core direction of where the challenge for ULSSIS lies: composition of the services projected by the constituent subsystems while addressing crosscutting quality concerns and providing the necessary flexibility for the evolution of the integrated system.

### 3. RICH SERVICES: A ULSSIS ARCHITECTURE BLUEPRINT

Traditional integration approaches involve time-consuming rework bearing major financial and technical risk. Service-Oriented Architectures (SOAs) have emerged as a widely accepted solution to this challenge; they use standards-based infrastructure to forge large-scale systems out of loosely coupled, interoperable services. New functionality can be created by either adding new services or modifying communication among existing services. However, most advances in the area of service-oriented development so far have been made at the deployment and implementation technology level. Recently, for instance, the notion of Enterprise Service Bus (ESB) has emerged. An ESB allows extremely flexible system integration at the deployment level. However, there is a need to exploit the flexibility of the service-oriented approach seamlessly throughout the entire development process. Hence, the challenge is to provide a new type of SOA that captures different system aspects and their interactions, and accounts for crosscutting concerns concisely and sensibly both at design time and later during the entire life-cycle of the system.

A combination of Service-Oriented Development and Model-Driven Architecture (MDA) and Design (MDD) techniques seems promising to address the integration and quality assurance challenges of ULSSIS. As MDA/MDD mature further, the vision of reflective, highly reconfigurable and adaptive systems is coming within reach. For concrete SoS integration solutions, this requires eliciting the appropriate models of the subsystems (at least to the degree necessary to obtain expressive interface specifications) as well as of the integration infrastructure to capture the entire spectrum of quality properties and the mechanisms to ensure them.

Our architectural vision for addressing the integration challenge for ULSSIS is the Rich Services architectural blueprint [1] shown in Figure 1. The architecture is organized around a message-based communication infrastructure and is hierarchically decomposed into Rich Services, which encapsulate various functionalities. The interaction between a Rich Service and its environment is accomplished via a Rich Service/Data Connector, which encapsulates the internal structure of the service. To manage service orchestration, the communication infrastructure has two main layers. The Messenger layer is responsible for transmitting messages between services. The Router/Interceptor layer is responsible for intercepting messages placed on the Messenger and then routing them among all services involved in providing a particular capability. This separation enables an efficient monitoring and enforcing of the quality-of-service properties required by the system at all levels of the hierarchy.

A Rich Service could be a simple functionality block or it could be hierarchically decomposed into further Rich Services. Rich Application Services provide core application functionality, mandating the business flow. Rich Infrastructure Services do not initiate any communication by themselves, but reroute or filter messages defined by the application services. Failure management, security, authorization, and policies are typical examples of infrastructure services. The communication infrastructure enables loose coupling and seamless communication between services. Such capability drives the evolution of the system by providing the underlying mechanisms to handle changes, dynamic reconfiguration, and policy enforcement. For example, the encryption concern can be addressed in a separate Rich Infrastructure Service rather than hard-coding it into every service in the system.

The use of a Router/Interceptor layer removes dependencies between services and their relative locations in the logical hierar-
chy. Thus, services from different levels of the hierarchy can interact with each other seamlessly with the help of appropriate infrastructure services and routing tables. This design choice enables massively distributed systems to be integrated and flexible system evolution to occur without requiring any particular deployment topology. For example, multiple security-related services can be designed and connected to the system at different levels in the Rich Service hierarchy to address the authentication, access control permissions, and privileges across multiple authority domains. Such adaptability enables the later addition of new features to an ultra-large system already built. The Rich Services architecture can be considered as both a logical architecture and a guide to a deployment architecture leading to an Enterprise Service Bus solution. The direct mapping between the logical architecture and its implementation simplifies the system evolution.

How does this blueprint respond to the challenges outlined above? First, it addresses the distribution and scattering of functionality by establishing a framework for re-integrating distributed services with their crosscutting concerns. This happens in a scalable way, and allows the option of introducing and managing policies at all levels of the hierarchy. Policies are associated with the interplay of the individual services, thereby establishing an end-to-end perspective. Second, each integration layer provides the opportunity to establish a common language [4] among the constituent subsystems; this facilitates integration of stakeholder concerns expressed in their individual domain languages while shielding the environment via the service/data connector. Third, the use of loose coupling of services by means of a messaging solution, as well as by lifting crosscutting concerns from the subsystems into the infrastructure paves the way for agile adaptation evolution of the resulting integration solution.

Demonstrating a concrete application of this blueprint is, of course, beyond the scope of this paper. However, we are successfully applying it both at the logical and deployment architecture level in a variety of projects – one of which is the previously mentioned OOI’s cyber-infrastructure.

4. CONCLUSIONS AND RESEARCH DIRECTIONS

In many ways, in his seminal paper on the THE operating system, E.W.Dijkstra laid the foundation for system-of-systems integration for a wide spectrum of system classes [2]; he reports specifically on the intellectual excitement and challenge this problem posed to him and his team. In essence, we face the same excitement and challenges again today in our efforts to build ULSSIS – in fact, one possible viewpoint on the integration fabric for ULSSIS (often termed cyber-infrastructure in today’s lingo) has all the characteristics of a domain-specific operating system layered on top of the enabling Internet infrastructure.

In the preceding sections, we outlined some of these key challenges, and discussed a first step towards creating an architecture blueprint that addresses these challenges. While the blueprint covers a significant part of the requirements spectrum for ULSSIS, much research is required for making progress towards a comprehensive engineering approach for this system class.

The lack of adequate modeling notations, methodologies, and seamless tool suites for precisely and adequately capturing the interplay between distributed components across all development phases is a critical problem with tremendous impact on quality assurance. Software and systems engineering research needs to deliver a comprehensive and flexible framework for the analysis, design, deployment, and evolution of ULSSIS. Such a framework should be based on a domain-driven, model-based approach to software and systems engineering.

Clearly, there is progress to be made along all dimensions of distribution, addressing the concerns of multi-disciplinary stakeholder communities, and the agility and flexibility of the resulting system and development process. Creating a comprehensive engineering approach to service-oriented ultra-large-scale systems requires a long-term research endeavor addressing theory to modeling to deployment.

Specifically, we see a need to establish a comprehensive service-oriented methodology for ULSSIS integration that (a) establishes a precise definition of services and their composition in the context of crosscutting concerns; (b) provides mechanisms for expressing and enforcing crosscutting concerns, including, but not limited to, failure management, authentication/authorization, and more general policies at the level of the interplay among a set of services; (c) captures the logical and deployment aspects of service composition and provides a mapping from the logical to the deployment concerns; (d) provides the necessary agility at runtime to adapt and dynamically re-configure the system based on changing requirements and environment situations such as recovering from failures in subsystems or from attacks on the overall system.

The underlying service notion naturally needs to be rich enough to also encompass stakeholder concerns and to support the creation of VCOIs as discussed in Section 2 – going far beyond what today’s integration approaches address. This will require an investment into the development of domain models for all aspects of the integration challenge, including failure management, security, policy and governance, as well as the user-centric aspects of user and application interfaces.

5. ACKNOWLEDGEMENTS

Our work was partially supported by NSF within the projects “ASOSA” (CCF #0702791) and “ITR: Collaborative Research: Looking Ahead: Designing the Next Generation Cyber-Infrastructure to Operate Interactive Ocean Observatories” (OCE/GEO #0427924), as well as by funds from the California Institute for Telecommunications and Information Technology (Caltit2).

6. REFERENCES