Rapid Systems of Systems Integration – Combining an Architecture-Centric Approach with Enterprise Service Bus Infrastructure

Ingolf H. Krueger, Michael Meisinger, Massimiliano Menarini, and Stephen Pasco
University of California, San Diego – Calit2
{ikrueger,mmeisinger,mmenarini,spasco}@ucsd.edu

Abstract

Rapid, yet methodical, systems of systems integration is in high demand. Application areas such as homeland security and disaster response add to the challenge because of a unique set of integration requirements; three examples are: (1) a high demand for flexibility with respect to the configuration and support of business processes to anticipate and cater to changing threat and mitigation scenarios, (2) high agility demands during both development and production to address legacy and emergent capabilities, processes, applications and technologies, (3) wide variety of trust relationships among and across stakeholders and their organizations. In this paper we report on an approach for balancing challenging integration requirements while rapidly delivering a high-quality, value added, integrated system architecture and service-based implementation infrastructure. In particular, we show how the choice of an Enterprise Service Bus as a deployment infrastructure helps discharge many of the obligations induced by the mentioned requirements – if it is combined with an agile, yet systematic approach for architecture discovery and design.

1. Introduction

Homeland security and disaster response applications today pose some of the most formidable challenges at software and systems architecture and integration. Besides the data-, process- and application-integration demands known from, say, enterprise application integration [5], these domains have a unique set of additional requirements. For instance, experience shows that threats and threat mitigation scenarios can change rapidly, and so must the business processes underlying corresponding decision support systems. Consequently, *flexibility* with respect to business process configuration and management is a key requirement in this domain.

In fact, the relevant requirements categories range from well-known and comprehensively researched data-/application-/process-integration to so far much less understood integration of heterogeneous trust, security and privacy zones in the face of a disaster.

We have encountered examples of such integration challenges within a wide range of projects conducted at the California Institute for Telecommunications and Information Technology (Calit2), with dozens to hundreds of stakeholders involved. Examples include the *BioNet* and *RESCUE/Responsesphere* projects. BioNet was a DHS/DTRA sponsored project, seeking to coordinate military and civilian capabilities to detect, characterize and manage the consequences of a biological attack. This involved, among others, establishing joint concepts of operations (business processes), as well as a software and systems integration architecture supporting these processes. A key goal was to provide common situational awareness to ensure timely, effective, and consistent response. Similarly, the NSF-funded RESCUE/Responsesphere project seeks to "radically transform the ability of responding organizations to gather, manage, use, and disseminate information within emergency response networks and to the general public. Depending upon the severity of the crisis, response may involve numerous organizations including multiple layers of government, public authorities, commercial entities, volunteer organizations, media organizations, and the public" [7].

Inherently, development of integration solutions in this space is an interdisciplinary, highly iterative and explorative effort. Success depends to a large extent on the ability of a developing organization to cater to the specific demands of this application domain while addressing the high demands at *agility* both during the development and implementation of the integration solution *and* during production of the resulting system.

In this paper, we argue that recent technological advances, such as service-oriented infrastructure technologies, together with a systematic, agile architecture development process are particularly well-suited to address these demanding requirements. In particular, we introduce the Enterprise Service Bus (ESB) we have configured as a means to discharge many of the flexibility and integration requirements in the projects mentioned above. This, in turn, enables the agility of the architec-

ture design effort, because less infrastructure details have to be explicitly modeled on the architecture level.

The remainder of this text is structured as follows. In Section 2 we discuss, in detail, the requirements that contribute significantly to the challenges of systems of systems integration in the homeland security and disaster response domains. In Section 3 we demonstrate the systematic development process we have successfully used across systems integration projects. In Section 4 we show how some of the obligations induced by the requirements identified in Section 2 can be discharged using an Enterprise Service Bus (ESB) as the systems integration infrastructure; the ESB, in particular, enables flexibility at all levels from architecture modeling and design to implementation, deployment and maintenance. In Section 5 we discuss our experiences with this approach. Section 6 contains our conclusions and an outlook.

2. Integration Requirements

In the following we take a closer look at the unique set of characteristics and requirements we find in systems of systems integration in the homeland security and disaster response domains. As most of the systems of interest are decision support systems (DSSs) we will focus our attention on this system class.

One key characteristic of DSSs is the large number of stakeholders, bringing individual capabilities and needs to the table. These stakeholders come from a wide variety of organizations, often including both civilian and military players, but also first responders, police, hazmat-teams, and intelligence providers. Furthermore, one typically finds providers of particular computational resources being members of an integration solution as well. Each of these organizations brings with it its own concepts of operations (business processes), capabilities and needs.

As we have argued above, threat scenarios change over time – certainly so over the long duration over which DSSs are in production; it is no stretch to assume that the DSSs put in place today will be operational over the coming few decades. Therefore, another characteristic of DSSs is the high demand for flexibility with respect to the configuration and support of business processes to anticipate and cater to changing threat and mitigation scenarios. This means that any integration solution has to enable rapid changes to the way the (sub)systems are connected, as well as to their protocols for information exchange. This leads to high agility demands during both development *and* production to address legacy and emergent capabilities, processes, applications and technologies,

At the highest level of abstraction we can group the remaining DSS requirements into the following areas: data-, process- and application-*integration*, *information management*, *infrastructure provisioning*, *trust and security management*, *presentation/UI/visualization*, as well as *quality and non-technical* requirements (such as standards compliance). In the following we refine these requirements categories further to demonstrate a relevant *subset* of the overall DSS requirements space. For reasons of space, we have limited our presentation to the requirements pertinent to the discussion in the remainder of this paper.

Integration: With respect to data- and application-integration, DSSs share requirements with traditional enterprise application integration [5] challenges. DSSs usually integrate numerous, vastly disparate data sources and corresponding interfaces, ranging from sensor networks to databases to individual applications. We can view each of the data sources and applications as "operational nodes" in the sense of [2][3] and interpret the capabilities they offer as "services" in the sense of a service-oriented architecture (SOA). Over the time the DSS is in production, many of the data sources (and their interfaces) will change. Consequently, an integration solution should enable flexible integration and configuration of legacy and emerging data sources and applications. In addition, as argued above, the need for flexible business process configuration and adaptation is paramount for DSSs. This already all but excludes traditional point-to-point integration solutions; they lack the required flexibility.

Information Management: Data and information extracted from it needs to be made available to eligible parties. This requires provisioning for information access and dissemination strategies, integrated with trust and security management (see below). In addition, pertinent information needs to be persisted such that it is available to all parties who need access at the appropriate time – including building and storing historical information. This is necessary to support the decision making process by providing access to *contextual* rather than purely *incidental* information.

Infrastructure Provisioning: DSSs in homeland security and disaster response encompass both fixed and mobile access. For instance, first responders arrive at the scene of a disaster with mobile devices including laptops, Personal Digital Assistants (PDAs), having access also to workstations in a virtual or physical command center. New networking capabilities need to be established in case of need. The diversity of access and presentation devices, together with their mobility requirements necessitates dealing with varying degrees of connectivity within distributed, heterogeneous networks. Another challenge is the inherent need for multi-modal interaction and its integration into a consistent situational DSS picture. For instance, many first responders cannot wield delicate input/output devices to participate in an integra-

tion solution; rather they can use sensors attached to them to provide pertinent data, and can communicate using voice and gestures. A viable integration solution needs to hide network complexity and the details of access methods as they would distract from the actual mission, say, of first responders. At the same time, the networking/communication infrastructure needs to support monitoring and management of the infrastructure itself, so that decision makers can include the state of the infrastructure into their situational picture and sphere of influence.

Trust and Security Management: Especially when the DSS spans multiple organizational boundaries, the integration solution needs to provide mechanisms for establishing and observing a wide variety of trust relationships among and across stakeholders and their organizations. This becomes obvious if we look at the many different parties who are involved in disaster response, for instance: first responders, haz-mat, police, possibly even the military, but also many corporate and private volunteers. Outside of a disaster situation, they all have their (legal) requirements for establishing trust relationships with other parties; how do these established relationships change in the face of devastation? How quickly can newly established rules for trust for the duration of a response effort be effectively implemented as part of the supporting IT infrastructure? Again, this characteristic of the domain appeals to the flexibility of the overall systems integration solution; statically coded trust relationships may hinder rather than support an effective response. Of course, a comprehensive integration solution also has to address security policies (who can access what under what circumstances; when and where does data need to be encrypted to what degree, etc.), access control and authentication including its monitoring, communication and (physical) infrastructure security, as well as data privacy (what part, say, of patient data can be legally transmitted across systems in light of a disaster situation.)

Presentation/UI/Visualization: To establish or increase situational awareness in crisis situations, user interfaces need to flexibly reflect the user role and their associated "need to know" and decision making capabilities. Furthermore, usability needs to be considered seriously and consciously built into the integrated system, catering to the wide range of end-users from data gathering in the field to decision makers. Many users will operate the system under stress, with zero to little time for climbing a learning-curve. Furthermore, the user interface needs to reflect the different types of (mobile and stationary) devices via which users will access the system.

Quality Aspects: Typically, DSSs have high demands at scalability, configurability, availability, survivability, reliability and other Quality-of-Service properties. Scalability and configurability are indispensable, because crises need not be contained in small geographic regions; this results in the need to provision for the interconnection of multiple DSSs, say, on a regional, national or international scale. The demands at availability and reliability are also inherent in the application domain.

3. Process and architecture design

In systems of systems integration one of the most challenging problems is dealing with the complexity of interactions between entities of the composite systems. We have identified a flexible lightweight iterative development process and an appropriate architecture design as key assets in providing an effective solution.

Dealing with complex integration projects, we often find that the requirements are emergent rather than stable and, because of the need to integrate disparate systems, often contradictory. The task of systems integration is to capture the business logic that deals with the interactions of different systems, and to ensure that requirements conflicts are resolved where they stand in the way of an effective solution. Clearly, IT is just one aspect of an overall software and systems architecture and integration effort; issues such as governance and policy management extend beyond mere IT. However, an appropriate IT integration architecture and infrastructure can support and, to some extent, even automate these aspects. This can be accomplished by simplifying, streamlining, automating, where possible, the identified business logic. Furthermore, it is usually the case that privacy and security requirements are of utmost importance; the elicited business processes should bring these requirements aspects out clearly.

To support the agility demands identified above, we therefore suggest to *co-iterate* business process and architecture elicitation. The resulting architecture needs to reflect and support the elicited business processes. In our experience, this can best be achieved if both evolve in parallel, such that the business processes inform the architecture, and the architecture validates the business processes.

Identification of business processes that capture all the interactions, procedures and security requirements the software needs to cater to is, therefore, a key ingredient to successful architecture development. In large integration efforts, *business process* documents (also referred to as ConOps) tend to become large and entangled, because they often exist just in prose. This can be balanced by extracting the key data and processing entities and their relationships from the business process documents and organizing them in the form of a *Domain Model* [4]. Domain Models capture the core "languages" of the subsystems in a way that is suitable for defining the interfaces among the subsystems. The usual way to create a *Domain*

Model in Object-Oriented Development (OOD) is to capture the entities and their relationships as a set of class diagrams. In our experience, the transition from textual description of business processes to class diagrams is painful and error prone, especially because the extraction effort requires frequent interactions with non-IT experts. To simplify our work, and to facilitate these interactions, we used a mind mapping tool to create an intermediate representation of the business processes (Figure 1).

The approach using *mind maps* has proven very valuable in our projects. We have based our architecture discovery process on an iterative refinement of the mind

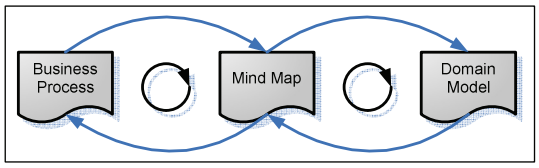

Figure 1. Iterative refinement of Business Process and Domain Model

maps elicited from the business process documents. Creating mind maps has helped us to find inconsistencies in the business process documents. Moreover, iterating between the Domain Model and the mind map allowed us to relay the improvements to the domain model we made during architecture design and implementation to the team eliciting the business processes and vice versa. The iterative refinement process shown in Figure 1 has been a significant asset in ensuring that the emerging business process documents and the software architecture and infrastructure are consistent and correct while carrying out a highly agile and iterative development process.

To guide our integration efforts we have adopted a formal development process based on Scrum [8]. This process prescribes quick iterations with constant adaptation of the goals to the changing (requirements) environment, facilitated by daily meetings. The use of such a flexible process allowed us to react quickly in incorporating frequent changes in *ConOps* and allowed us to provide rapid feedback to the business process development teams.

To ensure high quality of the products we develop, we adopted an incremental peer review strategy. The goal was to peer review every increment to every artifact daily. This keeps high standards throughout the development effort, while ensuring that there always is a consistent architectural basis. From agile development comes also the strategy to create monthly milestones. Every development group needs to deliver current versions of their artifacts. The artifacts need to progress and address an expanding set of requirements with each iteration – this has worked well for us within a ten to 15 person architecture team. Including a team of dozens to hundreds of stakeholders into an agile process requires establishing clear interfaces, for instance, by designating representatives of the other stakeholder groups as reviewers or authors of architecture artifacts.

To balance the agility of this process [1] we keep the architecture in the center of all our efforts. Keeping the architecture documents consistent with the implementation is fundamental to dealing with complex systems with changing requirements. Based on program requirements we have adopted the DoDAF standard [2][3] to create the architecture documents in one of our projects. The DoDAF keeps views of the system at two major levels of abstraction: operational and systems views. Operational views, intuitively, capture the "logical" architecture, whereas systems views focus on "deployment and implementation" aspects. We have found that the operational views can effectively be used to capture service-oriented architectures, despite their predominant use within component-oriented development approaches. In fact, from the domain model (which can be captured in OV6-7) the "operational nodes" and their capabilities/functions (OV 1-3) can easily be derived. From these representations we can then elicit the "services" provided and used by the various subsystems of the integration effort.

The mentioned co-iteration of business process and domain model development helps discharge, in particular, the flexibility and agility requirements mentioned *during* architecture development and supports system quality by closely aligning the architecture effort with stakeholder processes and needs.

The second key to the success of our system integration projects lies in the use of a deployment architecture that helps discharge the remaining requirements obligations. We have adopted a software and deployment architecture based on the message/service bus pattern [5], which we will describe in the next section.

4. Infrastructure

A suitable IT infrastructure needs to fulfill the described requirements, including application integration, security, workflow support, transactionality and reliable messaging. The flexibility to support changing requirements and thus to provide the ability to declaratively alter infrastructure capabilities or application configuration properties is essential. In this section we describe our integration architecture based on an Enterprise Service Bus (ESB), which addresses these issues.

An ESB can provide flexible and reusable *infrastructure* services that may be configured and augmented easily and quickly. In Table 1 we have collected the set of products and technologies we have analyzed and configured into an infrastructure offering to discharge the obligations from the remaining requirements discussed in Section 2. We now describe these infrastructure offerings in more detail. Note that for any one instance of an ESB

Table 1. ESB Infrastructure Offerings

Security:		**Orchestration/ Workflow**	WS-BPEL 1.1 and 2.x, Mule 1.8, PXE 1.0	**Containers**	Spring 1.2.8, Hivemind 1.1, ServiceMix 2.x, JBI, Mule 1.3, Celtix 1.0, iBatis 2.1.7
Authentication	Acegi 1.0	**Transaction support**	JDBC 3.x, JDBC 4.x, XA, JMS 1.1	**Provider Transports /Bindings**	EJB 2.x, EJB 3.x, File, FTP, IMAP, Quartz 1.5.2, RMI, SOAP, SSL, Stream, TCP, UDP, VFS, WSDL, JMS, VM (embedded), JDBC 3.x, JDBC 4.x, TCP, UDP, Multicast, HTTP, Servlet, SMTP, POP3, XMPP
Encryption	SSL, HTTPS	**Language neutral access**	REST, Tuscany		
Persistence	MySQL 5.x, Postgresql 8.1, Apache Derby 10.x	**Configuration Management**	JMX (J2SE 5.x)	**Topologies**	Peer to peer, ESB, Client/Server
Transactions	XA, JOTM 2.0.x, JTA 1.0.x	**Rules Engine**	WS-BPEL 1.1 and 2.x, Drools 2.0/3.0	**Scalability**	SEDA
Messaging	ActiveMQ 4.0, JBOSS Messaging 1.0, WebsphereMQ 6.0, SonicMQ 7.0	**Scripting Engine**	Groovy 1.0	**Routing**	Aggregate, Resequence, Forwarding consumer, Filtering outbound routing, Recipeint list, Multicasting, Chaining, Message splitting, Exception based, Response aggregator, Idempotent, Selective consumer
Object Brokers	Xbean, Spring 1.2.8, Mule 1.3	**WebServices**	Xfire 1.1, Axis 1.x, Axis 2.x, ActiveSOAP		

only a subset of these technologies need to be configured into an ESB.

Pluggable architecture: Our ESB offers a multitude of infrastructure choices. This includes, for instance, authentication protocols (such as Acegi), encryption standards (SSL, HTTPS), persistence providers (MySQL, Postgresql, Apache Derby), transaction support (XA, JOTM, JTA), messaging (ActiveMQ, JBOSS Messaging, WebsphereMQ, SonicMQ), and object broker middlewares (Xbean, Spring, Mule). Further choices include workflow support, configuration management, rule and scripting engines, WebService support, execution containers, transport protocols and adapters, network topologies, and routing.

Setup: The ESB provides standard interfaces to integrate services. Multiple providers may be called upon for key integration requirements, such as security, transactionality, workflow and reliable messaging. The pluggable architecture of an ESB allows organizations to use their preferred service solutions for their SOA. For instance, an integration solution can choose among BPEL-compatible rules engines, XSLT-based transformation engines or other messaging processors [6].

Agility: Services and components can be added flexibly to provide desired functionality. The ESB enables loosely coupled services to be assembled in composite applications with reliability and manageability. The integration solution can standardize on a connectivity policy that is independent of location and underlying network infrastructure. The ESB offers a breadth of connectivity options that scale with project needs.

Web Services Support: The open, extensible architecture of an ESB supports integration through web services; either in stand-alone services, or as part of a complex, distributed composite applications spanning multiple transport and message formats. The ESB can be used with any preferred SOAP solution, or support SOAP stacks as necessary.

Messaging: Publish/subscribe and point-to-point messaging enables message broadcasting and/or direct messaging. The messaging system supports durable, fault tolerant and highly available connections, and can be configured so that multiple applications receive important information simultaneously.

Deployment: The deployed ESB typically differs depending on the project. Figure 2 depicts one scenario we have configured for one of our DSS projects Sensors and disparate systems send messages to an ActiveMQ topic which runs as a separate instance on a server. ActiveMQ uses a local instance of Apache Derby for guaranteed message delivery.

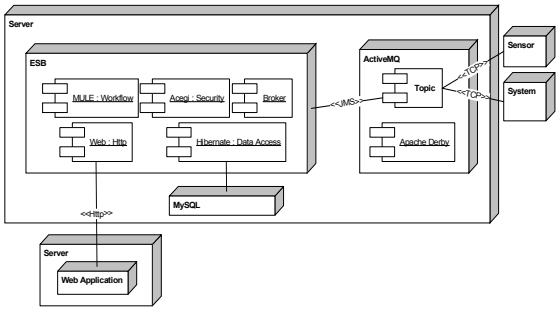

Figure 2. Deployment infrastructure

Once a message arrives on the ActiveMQ topic it is then immediately retrieved by the ESB, which has been configured as a JMS subscriber to the topic. Once retrieved, the message is then passed to the next configured ESB component; in this scenario the Broker is the first component to receive the message. From there it goes to the security component responsible for authentication, encryption/decryption, etc. The level and type of security is completely configurable and dependent upon the project security requirements.

During a predefined stage in the workflow, the message is typically persisted in some predefined state for

record keeping, archiving or post processing. Once a message has completed the tasks assigned by the workflow component it may need to be rendered for a human to view. This deployment depicts the ESB interfacing with a web application, such as a portal representing, say, a virtual command center. An Http binding component configured within the ESB allows this connection to the web application.

5. Experiences and evaluation

From the experience we have gathered in the different integration efforts, we learned important lessons that we believe are useful in dealing with systems of systems integration in general.

First we have learned that, given its foremost importance, security should be treated upfront in every integration architecture. Security is an end-to-end property. A careful analysis of the specific characteristics of the information required by each application, of who will be able to access, modify and own it is an integral part of every security analysis; we have leveraged the Common Criteria [9] based on the domain model we had developed to determine assets, risks, and mitigation strategies. Security analysis should always be undertaken as part of a global risk assessment addressing multiple view points.

Another lesson learned is that the use of open standards simplifies integration, faster development and deployment to organizations of different size.

From our experiences we can argue that using ConOps as a foundation for the software architecture document is a winning choice. Especially when the business processes are co-iterated with the architecture (model) and both are being defined or updated together with the implementation of IT integration, a close interaction between the ConOps teams and the software architecture team is very helpful. On one of our projects we have successfully used both teams to validate each other's work. The use of an agile process allowed us to have stakeholders always involved in the development of the integration solution. This fact is exceptionally important in the context of large integration efforts, where the systems that need to be integrated are complex, and different "owning" organizations are involved. In this scenario a clear and complete understanding of all the parts that will be integrated is usually not possible in the inception phase of the project. Laws and various regulations can influence and further limit the choices that have to be made to provide a viable system. An iterative approach with monthly milestones allows detection of requirements conflicts, inadequate expectations and new constraints as soon as possible, and provides a mechanism for presenting them to the stakeholders.

Because of the power and flexibility of the ESB, configuring and implementing it is nontrivial. Picking the right components and technologies to work together requires careful analysis. Creating the components that provide specific services and their wiring to the ESB still is not for free – adapters need to be written and communication channels on top of the messaging platform must be configured. However, this level of complexity is a far cry from having to establish and evolve traditional point-to-point connections while supporting all the requirements outlined in Section 2.

6. Conclusions and Outlook

Rapid systems of systems integration is a challenging task, requiring the balancing of a vast spectrum of requirements, while maintaining high quality standards. In this text, we have presented an approach to iterative, agile development for complex systems of systems integration problems in the homeland security and disaster response fields. Our experience indicates that using an enterprise-service-bus-based architecture together with an agile development process with rapid iterations, monthly milestones and daily peer reviews can be an effective and winning approach to integration projects. Particular value emerges from the *co-iteration* of business process/ConOps development with architecture modeling and infrastructure development. We also found initial indication that this integrated approach supports management of security aspects throughout the development process; this we will investigate further in future work.

7. Acknowledgements

Our work was partially supported by project Rescue (under National Science Foundation ITR Grant IIS-0331690), and by Calit2 (http://www.calit2.net/). We are grateful to the anonymous reviewers for their insightful comments.

8. References

[1] Barry Boehm, and Richard Turner, *Balancing Agility and Discipline: A Guide for the Perplexed*, Addison-Wesley Professional, 2003.

[2] *DoD Architecture Framework Version 1.0 Volume 1*, http://www.aitcnet.org/dodfw/VolumeI.zip

[3] *DoD Architecture Framework Version 1.0 Volume 2*, http://www.aitcnet.org/dodfw/VolumeII.zip

[4] Eric Evans, *Domain-Driven Design: Tackling Complexity in the Heart of Software*, Addison-Wesley Professional, 2004.

[5] Martin Fowler, *Patterns of Enterprise Application Architecture*, Addison-Wesley Professional, 2002.

[6] Nodet, Guillaume, *Introduction to ESBl*, Online Posting, 11 May 2006 http://www.servicemix.org/site/

[7] Rescue/Responsephere, Website http://www.itr-rescue.org/

[8] Ken Schwaber, and Mike Beedle, *Agile Software Development with SCRUM*, Prentice Hall, 2001.

[9] The Common Criteria Evaluation and Validation Scheme, http://niap.nist.gov/cc-scheme/index.html