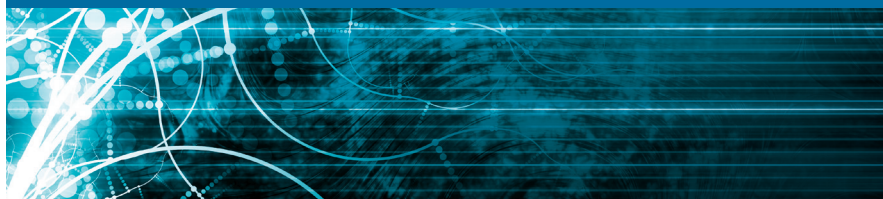


Service Orientation and Systems of Systems

Grace Lewis, Edwin Morris, Soumya Simanta, and Dennis Smith,
Software Engineering Institute

// Interconnected systems of systems provide capabilities that aren't available in any single system. Fundamental service-oriented principles can help in engineering them, regardless of the implementation technologies used. //



GIVEN THE NEED TO COLLECT REAL-TIME INFORMATION, integrate business partners to reduce end-to-end delivery times, and capitalize on globalization, organizations increasingly rely on interconnected systems of systems (SoS) that provide capabilities not found in single systems. Many approaches exist for engineering these SoS, ranging from point-to-point implementations to much more dynamic, ad hoc options in which configurations aren't known until runtime.

Service orientation is an approach to software systems development that

has become a popular way to implement distributed, loosely coupled systems because it offers such features as standardization, platform independence, well-defined interfaces, and tool support that enables legacy system integration. In fact, service-oriented architecture (SOA) has officially “crossed the chasm” between visionaries (the early adopters) and pragmatists (the early majority), according to a recent Software AG user survey in which 90 percent of the respondents claimed to have made some commitment to SOA adoption.¹ Gart-

ner's latest report on hype cycles for emerging technology shows SOA at the midpoint of the “slope of enlightenment,” meaning that methodologies and best practices are developing such that the technology is close to mainstream adoption.² As with the various approaches that came before it, we can expect newer technologies to replace or complement this approach eventually, but we believe service-orientation principles, regardless of implementation technologies, are beneficial to SoS engineering.

Service-Oriented SoS

In a service-orientation approach,

- services provide reusable business functionality via well-defined standardized interfaces to promote interoperability;
- service consumers exploit functionality in available services;
- service interface and implementation are clearly separated to promote platform independence;
- a SOA infrastructure enables discovery, composition, and invocation of services to promote loose coupling between consumers and services; and
- protocols are predominantly (but not exclusively) message-based document exchanges.

Figure 1 shows a high-level, notional view of a service-oriented system.

An SoS is “a set or arrangement of systems that results when independent and useful systems are integrated into a larger system that delivers unique capabilities.”³ We can therefore think of a software-reliant SoS as one that relies on software to accomplish its goals. Figure 2 shows some SoS examples, ranging from less complex to very.

Mark Maier identifies five SoS characteristics that are useful in distin-

guishing such systems from very large and complex but monolithic systems:⁴

- operational independence of the constituent systems;
- managerial independence of constituent systems;
- evolutionary development;
- emergent behavior; and
- geographic distribution.

Maier also defines four types of SoS based on their management structure:⁴

- *directed*, in which constituent systems are integrated and built to fulfill specific purposes;
- *acknowledged*, in which SoS have recognized objectives, a designated manager, and resources;
- *collaborative*, in which constituent systems voluntarily agree to fulfill central purposes; and
- *virtual*, which have no central authority or centrally agreed purpose.

In accordance with these characteristics, software-reliant SoS tend to be highly distributed and formed from constituent software systems operated and managed by different organizations.

Service orientation is becoming much more common for SoS implementation because its characteristics support generally common goals, such as cost-efficiency, adaptability, business agility, and leverage of legacy systems. Other current technologies for implementing SoS include cloud computing, grid technologies, and event-driven architecture. However, as software-reliant SoS move from directed to virtual paradigms, more complex technologies will be necessary to deal with the lack of a central authority or a centrally agreed purpose.

Leveraging Service-Oriented Principles

Several service-oriented principles that have contributed to wider SOA adop-

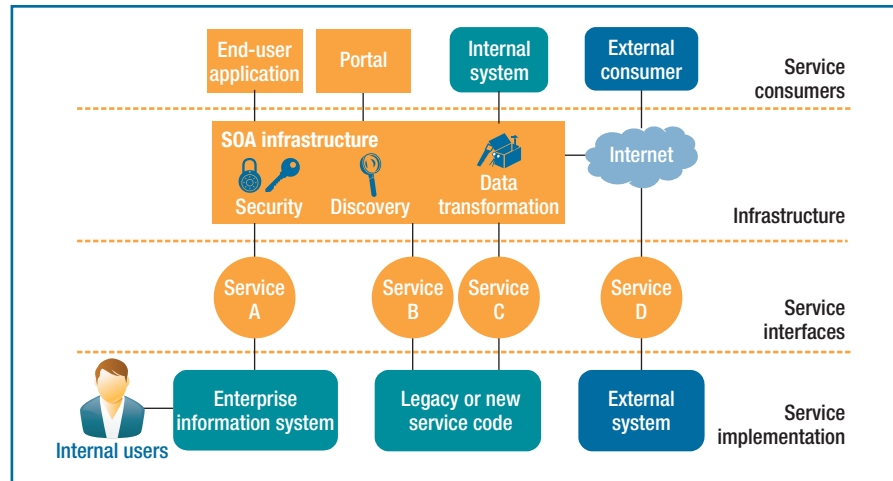


FIGURE 1. Notional view of a service-oriented system. Its main elements are services that provide reusable functionality with well-defined interfaces; a service-oriented architecture infrastructure that enables service discovery, composition, and invocation; and consumers using functionality in available services.

tion—standardization, loose coupling, strategic service identification, service discovery mechanisms, and governance—also work for SoS.

Standardization

Although options abound for implementing service-oriented systems, the most common is based on the WS-* set of standards for Web services.⁵ Standardization has many benefits for service-oriented systems, including

- interoperability (promoted by standard interfaces to heterogeneous technologies);
- ability to use third-party services;
- ability to use off-the-shelf tools based on a single set of standards;
- enablement of other aspects of service-oriented systems, such as service discovery and composition; and
- potential for shorter development times because all that a service consumer needs to know to use a service is contained in the service description.

Standardization is a controversial topic in many SoS settings, especially in virtual settings in which systems that have never come together will do so at runtime. We believe that some level of

standardization is necessary, especially in virtual SoS environments, to deal with the lack of a centrally agreed purpose.

SoS benefits from standardization because we can encapsulate the complexity and heterogeneity of constituent systems to promote operational independence, managerial independence, and geographic distribution of constituent systems. Standardization also enables other aspects of importance to SoS, such as interface-based testing, that are often difficult to achieve because of autonomy and independence of constituent systems.⁶

Loose Coupling

Different architectural patterns emphasize different forms of coupling, but there's always some form of it—for example, data-centric systems are coupled to data models, and event-driven systems are coupled to events and event mechanisms.

Service orientation has two forms of loose coupling:

- *Between the service provider and the service consumer.* In a service-oriented environment, providers and consumers know as little as possible about each other. In the case of WS-* -based Web services,

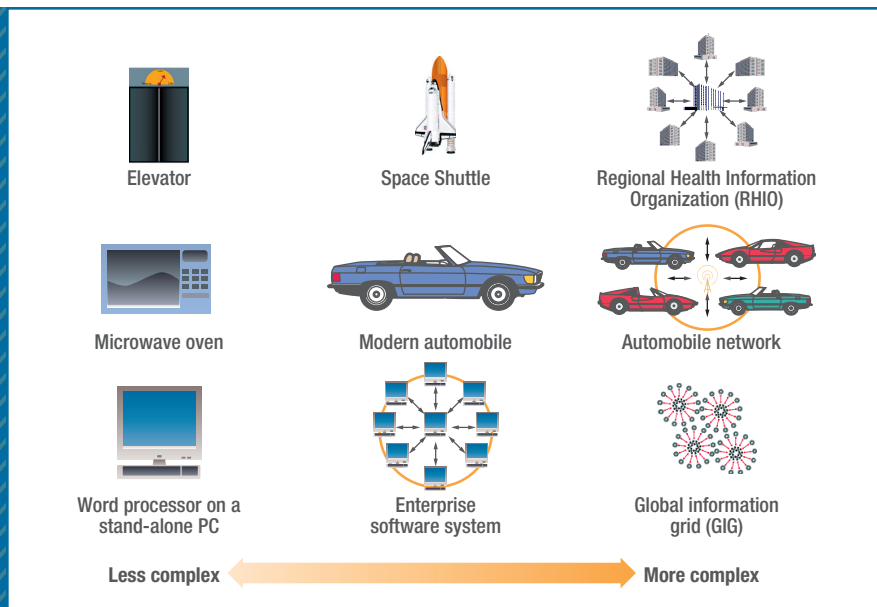


FIGURE 2. Systems of systems (SoS) examples. A wide spectrum of SoS ranges from the simple systems that we use on a daily basis such as PCs and microwave ovens to very complex systems such as automobile networks that involve complex interactions between vehicles and the environment.

potentially all that the consumer needs to know to use a service is contained in the service interface, that is, the Web Services Description Language (WSDL) document associated with the service. The SOA infrastructure mediates many of the differences between providers and consumers and also centralizes support for key quality attributes.

- *Between the service interface and the service implementation.* This clear separation of service interface and implementation is what supports platform independence in service-oriented systems. The standardized service interface hides the technology details of the service implementation, as well as its geographic location.

The main benefit of supporting these forms of loose coupling in SoS environments is the ability to hide technology and constituent system locations. In a service-oriented system, the SOA infrastructure mediates differences between service consumers and providers.⁷ This mediation concept can be extended to

SoS environments to deal with differences between constituent systems.

We can also apply other principles used in service-oriented systems to support loose coupling in SoS environments. For example, technology shouldn't bleed through system interfaces (error messages in the service implementation shouldn't pass on to the service consumer other than via standard error messages that are part of the service interface), and integration mechanisms should be technology-neutral.

Strategic Service Identification

In service-oriented environments, service identification occurs in the problem domain, based on the premise that the business changes, not the technology. The primary input is the set of business goals for SOA adoption. Top-down approaches identify business processes that support business goals, and common steps between these processes become candidate services. Bottom-up approaches identify legacy capabilities that support business goals as candidate services. A combination of these approaches leads to the identification of

services that represent reusable business capabilities.

This process for service identification in SOA environments works in SoS environments as well. In this case, business processes map to SoS usage scenarios, and services map to constituent system capabilities.

Service Discovery Mechanisms

In a service-oriented environment, services are created and published somewhere that's accessible to consumers (registry, webpage, directory, and so on) so that they can find the services they require. At a minimum, the metadata associated with a service is its specification or contract; any additional metadata is commonly stored in a repository and includes attributes such as description, classification, usage history, test cases, test results, quality metrics, and documentation. Metadata should include everything needed to discover and reuse a capability, which minimizes the interaction between developers and therefore promotes agility. It should also include information about its attributes once it's discovered, such as quality (performance limitations), assumptions (encryption mechanisms), and constraints (usage context).

SoS environments should consider making a capabilities repository accessible to constituent systems as well as to SoS developers searching for capabilities. Considerable tool support can be leveraged for registry and repository implementation.

Governance

SOA governance is the set of policies, rules, and enforcement mechanisms for developing, using, and evolving service-oriented systems and for analyzing the business value of those systems.⁸ SOA governance has three types:

- *Design-time governance* includes elements such as rules for strategic identification, reuse, development,

and deployment of services as well as policies to enforce consistency in aspects such as use of standards, reference architectures, and processes.

- *Runtime governance* applies to deployment and management of service-oriented systems and includes policies to enforce service execution according to policy and to ensure that important runtime data is logged and analyzed.
- *Change-time governance* applies to maintenance and evolution of service-oriented systems and includes policies for maintenance and evolution of service-oriented system elements as well as communication of changes to stakeholders.

Something similar to SOA governance is necessary in SoS environments to provide agreement on aspects such as characteristics of capabilities registry; design-time, runtime, and change-time policies; establishment and monitoring of service-level agreements (SLAs) between the SoS and constituent systems; and tool support. Because of the independent nature of constituent systems, change-time governance is also key to SoS. How are problems in constituent systems reported? What happens if a constituent system changes? How are changes and upgrades in constituent systems communicated? How will capabilities of constituent services be tested?

What Doesn't Yet Work Well

Even though multiple aspects of service orientation can be applied in SoS environments, several aspects still aren't mature enough to support SoS, especially as we move toward virtual SoS.

Multi-Organizational SOA Implementations

The top drivers for SOA adoption have primarily focused internally: application integration, data integration, and internal process improvement. Even though this is changing, the number of organi-

zations that use service orientation for external integration is still a minority.⁹ The execution of distributed development tasks such as system assurance is an ongoing challenge in SOA environments.¹⁰ Multi-organizational concerns, such as trust, federation, and security, are active research areas with few examples of large-scale implementations.

Standardization Quality for Attribute Specification

Despite multiple standardization efforts and research projects, there isn't yet a widespread standard for specification of quality attributes such as security, availability, and performance. Standardization in this area is key for enhanced service discovery and for automated SLA management and monitoring.

Support for Interoperability beyond the Syntactic Level

Systems interoperability can only be achieved with agreement at the syntactic, semantic, and operational levels. Interoperability exists at the syntactic level when there's agreement on data representation. Semantic interoperability requires agreement on the meaning of the exchanged data. Finally, organizational interoperability requires agreement on how to act on the exchanged data. Although mature standards for SOA implementation such as XML and WSDL support syntactic interoperability, achieving semantic and organizational interoperability remains a research challenge.¹¹

Dynamic Service Discovery and Binding

Dynamism has various degrees. At the lower end of the spectrum is late binding of a proxy service to a specific service instance that depends on user context or load-balancing policies. At the higher end is fully dynamic binding in which service consumers are capable of querying service registries at runtime, selecting the "best" service from the list, and invoking the selected service. Late bind-

ing is a common, out-of-the-box feature of many SOA infrastructure products. Fully dynamic binding, on the other hand, requires semantically described services that use an ontology shared between service consumers and service providers. This is an active area of research, as well as an unsolved problem.

SOA Governance Automation

Many aspects of SOA governance simply can't be automated, especially in design-time governance. In addition, the behavioral aspects of governance make it difficult to enforce without automation, especially in multi-organizational system implementations.

A Service-Oriented SoS Engineering Approach

Figure 3 presents a service-oriented SoS engineering approach. Even though we recognize that it corresponds to an ideal situation, it does embed the concepts and trade-offs that would be necessary for a service-oriented approach to SoS engineering. The main actors in Figure 3 include the following:

- *SoS end user* represents a user who requires new SoS capabilities.
- *SoS developer/integrator* represents the development/integration team responsible for providing capabilities to the SoS end user. In a collaborative and virtual SoS environment, the line between the SoS end user and the SoS developer/integrator is blurry. Especially in a virtual SoS environment, the SoS end user could perform the tasks of the SoS developer/integrator.
- *System developer* represents the constituent system's development team.
- *System end user* represents the constituent system's end users.

In the ideal process proposed in Figure 3, the cycle starts with an SoS end user who has a requirement for new

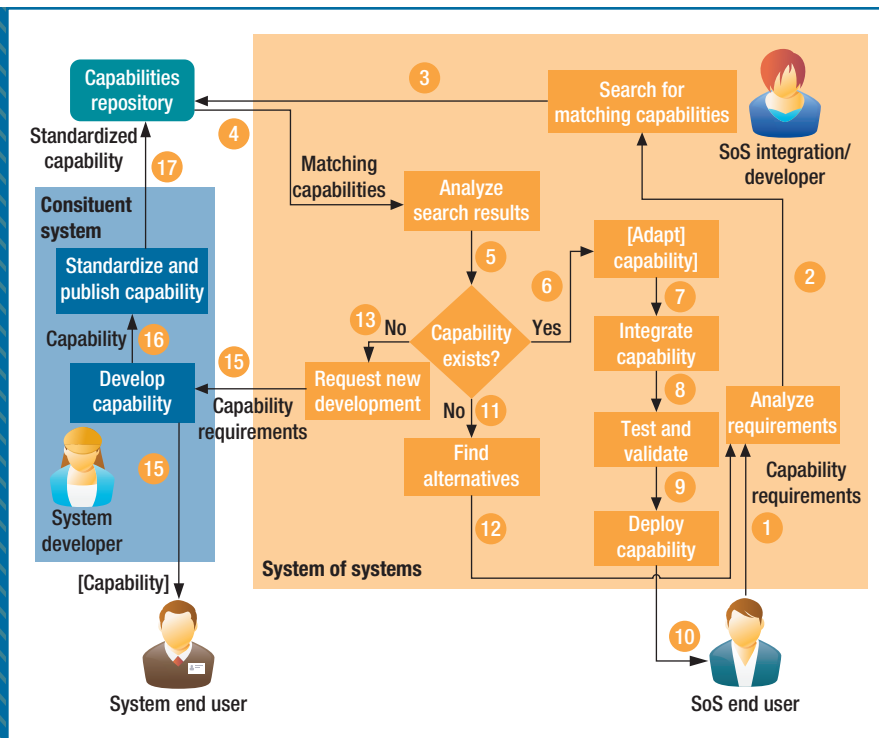


FIGURE 3. Service-oriented SoS engineering approach. In this approach, the SoS developer/integrator tries to match requirements to existing standardized capabilities that are available in a capabilities repository before committing to any type of development “from scratch.”

capabilities. Each step below corresponds to its matching number in the figure (although not explicit in the figure, SoS governance is in place to ensure that the process functions as described):

1. An SoS end user has a requirement for new capabilities.
2. The SoS developer/integrator analyzes the new capability requirements.
3. The SoS developer/integrator searches the capabilities repository for matching capabilities.
4. The SoS developer/integrator analyzes the returned matching capabilities.
5. The capability requirements might match an existing capability (to a certain extent) or not. If there isn't a match, the SoS developer/integrator might have the option of finding other alternatives or requesting new development.

If the capability exists in the capabilities repository:

6. If there isn't a 100 percent match between requirements and capabilities, the SoS developer/integrator might adapt the existing capability by developing code to deal with the mismatches.
7. The SoS developer/integrator integrates the new capability into the SoS.
8. The SoS developer/integrator tests and validates the new capability and its effect on the SoS.
9. The SoS developer/integrator deploys the new capability.
10. The new capability is delivered to the SoS end user.

If the capability doesn't exist in the capabilities repository and the SoS developer/integrator can't afford to wait for development of new capabilities:

11. The SoS developer/integrator looks for alternatives either by repeating the search with different criteria or

by presenting the SoS end user available capabilities and seeing if the end user is willing to relax capability requirements in exchange for faster delivery of capabilities.

12. Process starts again at Step 2.

If the capability doesn't exist in the capabilities repository, and the SoS developer/integrator can wait for development of new capabilities (this is possible for directed and acknowledged SoS environments but might not be possible in collaborative and virtual ones):


13. The SoS developer/integrator sends requirements to the system developer.
14. The system developer develops capabilities that match the requirements.
15. The system developer might decide to make the new capability available to system end users.
16. The system developer standardizes and publishes the new capability in the capabilities repository. Depending on the type of relationship between the SoS developer/integrator and the system developer, the SoS developer/integrator could require certain tests and compliance requirements before publishing the capability.
17. After the new capability is published, it's now available to the SoS developer/integrator who would restart the process at Step 3.

The time required to go through an iteration of this process highly depends on how long the SoS end user can wait for capabilities and how much he or she is willing to accept mismatches in exchange for quicker capability deployment.

In an ideal service-oriented SoS environment, constituent systems would register their capabilities in a service-oriented, standardized capabilities registry. SoS architects, integrators, and developers would query

this registry to find capabilities that meet desired functional and quality attribute requirements. Each capability in the registry would contain associated metadata to be able to use the capability.

Because of a lack of control over SoS constituents providing these capabilities, SoS architects, integrators, and developers would use practices that include mechanisms to deal with this challenge. One example is relying on runtime-monitoring mechanisms in addition to testing for system assurance. Another example is the use of defensive programming practices such as exception-handling techniques and fallback strategies to handle situations in which constituent systems are unavailable.

The concept of service orientation is here to stay, but its implementation technologies will change over time to meet new system requirements. For example, both cloud computing and software as a service are approaches largely based on service orientation. SoS engineering efforts could benefit from service-oriented principles that support operational independence, managerial independence, and geographic distribution of constituent systems. The challenge for service-oriented SoS, as well as for service-oriented systems in general, is behavioral and not technological: the incentives and enforcement mechanisms have to be in place for this approach to succeed. 



GRACE LEWIS is a senior member of the technical staff at the Software Engineering Institute (SEI), where she's currently the lead for the System of Systems Engineering team within the Systems of Systems Practice (SoSP) Initiative in the Research, Technology, and Systems Solutions (RTSS) program. Her current interests and projects are in service-oriented architecture (SOA), cloud computing, and technologies for systems interoperability. Her latest publications include multiple reports and articles on these subjects and a book in the SEI Software Engineering Series. Lewis has a B.Sc. in systems engineering and an executive MBA from Icesi University in Cali, Colombia, and an MS in software engineering from Carnegie Mellon University. Contact her at glewis@sei.cmu.edu.



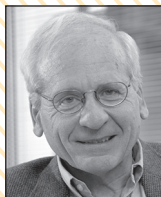
ejm@sei.cmu.edu.

EDWIN MORRIS is a senior member of the technical staff at the Software Engineering Institute (SEI) with more than 20 years' experience in the software field, including design and development of embedded real-time systems and support tools, management of technical staff, and support for a wide range of military, government, and corporate enterprise systems of systems initiatives. Morris is currently a member of the RTSS program at the SEI, where he's investigating strategies for employing smartphone technology in tactical systems. Contact him at



India's largest software companies. He has a B.E. in electronics engineering from Sambalpur University and an MS in software engineering from CMU. Contact him at ssimanta@sei.cmu.edu.

SOUMYA SIMANTA is a member of the technical staff in the System of Systems Practice (SoSP) Initiative of the Research, Technology, and Systems Solutions (RTSS) Program at Carnegie Mellon University's Software Engineering Institute (SEI). His current research is in mobile computing, service-oriented computing, and evaluation of current and emerging technologies for engineering systems of systems. Before coming to the SEI, Simanta performed software design and development in the finance and telecommunication domains, working for one of



Columbia University. Contact him at dbs@sei.cmu.edu.

DENNIS SMITH leads the System of Systems Practice (SoSP) Initiative in the Research, Technology, and Systems Solutions (RTSS) Program at Carnegie Mellon University's Software Engineering Institute (SEI). He co-organized the development of a SOA research agenda and has co-developed SMART, a method for migrating to SOA environments. Smith is a member of the executive committee of IEEE's Technical Council on Software Engineering (TCSE). He has an MA and a PhD from Princeton University in sociology, and a BA in sociology from

References

1. "SOA Governance User Survey: Best Practices for SOA Governance User Survey," Software AG, 2008; www.softwareag.com/Corporate/res/SOAGovernanceSurvey.asp.
2. "Hype Cycle for Emerging Technologies," Gartner Research, 2009; www.gartner.com/DisplayDocument?id=1085912.
3. "Systems Engineering Guide for Systems of Systems," Office of the Undersecretary of Defense for Acquisition, Technology, and Logistics, Aug. 2008; www.acq.osd.mil/sse/docs/SE-Guide-for-SoS.pdf.
4. M. Maier, "Architecting Principles for Systems-of-Systems," *Systems Eng.*, vol. 1, no. 4, 1998, pp. 267–284.
5. G. Lewis and L. Wrage, "Model Problems in Technologies for Interoperability: Web Services," *Software Eng. Inst.*, Carnegie Mellon Univ., 2006; www.sei.cmu.edu/library/abstracts/reports/06tn021.cfm.
6. S. Ghosh, "Testing Component-Based Distributed Applications," Purdue Univ., 2000; <http://docs.lib.purdue.edu/dissertations/AAI3018200/>.
7. D. Chappell, *Enterprise Service Bus*, O'Reilly, 2004.
8. S. Simanta et al., "A Scenario-Based Technique for Developing SOA Technical Governance," *Software Eng. Inst.*, Carnegie Mellon Univ., 2009; www.sei.cmu.edu/library/abstracts/reports/09tn009.cfm.
9. "Enterprise and SMB Software Survey, North America and Europe, Q4 2008," Forrester, 2009; www.forrester.com/ER/Research/Survey/Excerpt/1,5449,704,00.html.
10. E. Morris et al., "Testing in SOA Environments," tech. report CMU/SEI-2010-TR-011, Software Eng. Inst., Carnegie Mellon Univ., 2010; www.sei.cmu.edu/library/abstracts/reports/10tr011.cfm.
11. G. Lewis et al., "Why Standards Are Not Enough to Guarantee End-to-End Interoperability," *Proc. 7th IEEE Int'l Conf. Composition-Based Software Systems (ICCBSS 2008)*, IEEE Press, 2008; http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4464021.