SORASCS: A Case Study in SOA-based Platform Design for Socio-Cultural Analysis

Bradley Schmerl, David Garlan, Vishal Dwivedi, Michael W. Bigrigg, and Kathleen M. Carley
School of Computer Science, Carnegie Mellon University
5000 Forbes Ave
Pittsburgh, PA, 15213 USA
+1 412 268 1252

{schmerl,garlan,vdwivedi,bigrigg,carley}@cs.cmu.edu

ABSTRACT
An increasingly important class of software-based systems is platforms that permit integration of third-party components, services, and tools. Service-Oriented Architecture (SOA) is one such platform that has been successful in providing integration and distribution in the business domain, and could be effective in other domains (e.g., scientific computing, healthcare, and complex decision making). In this paper, we discuss our application of SOA to provide an integration platform for socio-cultural analysis, a domain that, through models, tries to understand, analyze and predict relationships in large complex social systems. In developing this platform, called SORASCS, we had to overcome issues we believe are generally applicable to any application of SOA within a domain that involves technically naïve users and seeks to establish a sustainable software ecosystem based on a common integration platform. We discuss these issues, the lessons learned about the kinds of problems that occur, and pathways toward a solution.

Categories and Subject Descriptors
D.2.11 [**Software Architectures**]: Domain-specific architectures.

General Terms
Design, Human Factors.

Keywords
Service Oriented Architectures, Platform Design, Socio-Cultural Analysis

1. INTRODUCTION
An increasingly important class of software-based systems is platforms that permit the integration of third-party components and tools. Like frameworks [13] platforms provide a way to integrate independently-developed elements that can take advantage of common services (such as communication infrastructure, user interface mechanisms, registry and look-up services, etc.), and that in turn conform to the requirements of the platform (such as observing platform protocols, user interface conventions, initialization and take-down procedures, etc.).

An example of a widely-used platform is the High Level Architecture (HLA) for distributed simulation. Initially developed by the US Department of Defense, and later standardized as IEEE 1516, HLA supports the integration of simulations built by different vendors, running on distributed hosts. HLA provides common services for inter-simulation communication, data management, and simulation management (e.g., registering new simulations). In turn, an HLA-compliant simulation must provide an interface that is called by the HLA runtime to carry out a joint simulation exercise. Other widely-used platforms include a large variety of service-oriented architecture (SOA) platforms (such as IBM WebSphere, Apache ServiceMix, Mule), smart phone platforms (such as iOS and Android), and grid computing platforms (such as ^{my}Grid).

Among the platforms in wide use today, service-oriented architectures are particularly prominent. SOA is an approach for developing large-scale distributed systems through the incorporation of loosely-coupled services and typically exploits the technology of the Internet. To migrate legacy systems into the SOA domain, they are typically modified (often through wrappers or adapters) to provide a service-based interface, which is registered with a SOA. The service then becomes available for invocation (using standard web protocols such as SOAP), or it can be combined with other services to produce more complex applications using an "orchestration" language (such as BPEL) that prescribes the order of service invocations and pathways of communication between them. SOAs have found widespread applicability in support of enterprise business management and e-commerce applications (such as Amazon and eBay).

SOAs are attractive not only for business and e-commerce; many other domains could in principle benefit from their use. Indeed, one might argue that SOAs are ideally suited for any family of applications that must be developed out of heterogeneous, independent components running in a distributed setting. Examples include scientific computing (e.g., MeDICi [11]) and healthcare (e.g., VistA [9]).

One such domain is the area of socio-cultural analysis (SCA). SCA is an increasingly important kind of application domain that attempts to build behavioral models of social systems (e.g., by extracting information from unstructured text such as web sites, blogs, email, etc.), and gain insight about these social systems through analysis, simulation, or observation of the models. It is used heavily in a variety of fields including anthropology, sociology, business planning, law enforcement, and national security. While there are many powerful tools to support these activities, there is currently no standard way to integrate tools from different developers or to selectively *compose* capabilities from different tools.

ICSE'11, May 21-28, 2011, Waikiki, Honolulu, HI, USA

But how easy is it to instantiate a SOA to create a platform for such a domain? If you believe the hype, it should be straightforward. One simply provides a web service description file for each service, registers those services with a registry, and voila! However, our experience is that this is not the case. In this paper we detail that experience, illustrating both the benefits and drawbacks of using SOA as a platform for SCA. Our primary insight is that although SOA provides an adequate *starting point* as a platform for domains like SCA, it is not by itself sufficient. In particular, the critical question that one must answer is how to create a platform that will support a sustainable software ecosystem [14].[1] To do this, we argue that one must go beyond the raw capabilities of SOA to address the real needs of the ecosystem's stakeholders: end users, tool integrators, and framework maintainers. As we illustrate, this requires substantial additions and adaptations to simple SOA in order to make a successful ecosystem.

2. SOCIO-CULTURAL ANALYSIS TODAY

Socio-cultural analysis involves understanding, analyzing and predicting the relationships in complex social systems. Such systems are typically represented as dynamic social networks that relate entities in the system (e.g., people, knowledge, places, actions) to each other. Dynamic network analysis (DNA) is centered on the collection, analysis, understanding and prediction of dynamic relations among multi-mode networks, and the impact of such dynamics on individual and group behavior [1][5]. DNA facilitates reasoning about real groups as complex dynamic systems that evolve over time. Within this field, computational techniques, such as machine learning and artificial intelligence, are combined with traditional graph and social network theory, in addition to empirical research on human behavior, groups, organizations, and societies to develop and test tools and theories actions that are enabled and constrained by relations in the network.

Socio-cultural analysis techniques typically entail a series of procedures. First, one needs to gather the relational data. One approach to this is to extract relations from a corpus of texts, such as public domain web pages, news articles, journal papers, stock holder reports, community rosters, etc., and various forms of human and signals intelligence. Second, the extracted networks need to be analyzed. That is, given the relational data, analysis identifies key actors and sub-groups, points of vulnerability, and so on. Third, given a set of vulnerabilities, we can ask what would happen to the networks were the vulnerabilities to be exploited. How might the networks change with and without strategic intervention? For example, military intelligence may use news reports, intelligence reports, etc., to build a network to understand the "human terrain" of the field of operations, and then use simulation to determine how best to communicate with a population; federal agencies may take a combination of news stories and crime reports to understand gang-related drug activities in a city and use simulation to plan the best courses of action to fight them; sociologists may use interviews and other sources to understand a population and then use simulation to work out the best strategies for educating them about policy changes.

[1] "A *software ecosystem* is a set of actors functioning as a unit and interacting with a shared market for software and services, together with the relationships among them. These relationships are frequently underpinned by a common technological platform or market and operate through the exchange of information, resources and artifacts."[18]

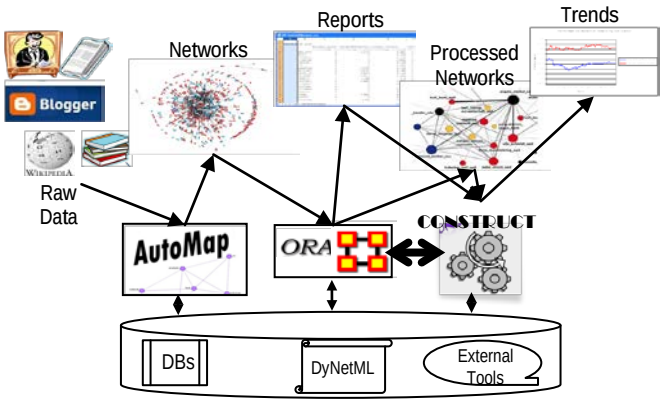

Figure 1. The Toolchain for socio-cultural analysis developed by the CASOS group at Carnegie Mellon.

A large variety of tools have been developed to help analysts with such tasks. For example, the Analyst's Notebook (I2)[2], ORA [7], and UCINET [2] provide tools to help conduct network analysis. But the tools in this area go well beyond network analysis to include tools for web scraping, text mining, data mining, statistical analysis, geo-spatial analysis, decision support, simulation, and gaming. The center for Computational Analysis of Social and Organizational Systems (CASOS) at Carnegie Mellon University has been engaged in developing methods and tools that provide support from the data collection phases through the analytic phase to the simulation or "what-if" forecasting phase. This toolset contains the following tools: AutoMap [7] for extracting networks from natural language texts, ORA for analyzing and visualizing networks [8], and Construct [3][6] for what-if reasoning about the networks. Analysts in this area always use sets of tools, in an iterative fashion, to address diverse questions with data that varies in type, quantity, and quality. Figure 1 provides an example of the way that these tools are integrated into a tool chain. Each of the tools is a standalone program. They may be loosely integrated through files written in an XML format called DyNetML, a standard interchange format for multi-mode social network data. These tools can be used in a linear process; however, typically there are a series of analytic spirals that may use multiple tools as the analyst cleans and refines the data and fine-tunes the analysis.

An SCA analyst will typically use tools in this domain to develop models of social systems, such as for a humanitarian relief effort like the Haiti earthquake in 2010. Such a model can be built from open source news data provided through a source such as Lexis-Nexis. This textual data needs to be preprocessed into a usable form, or "cleaned," to filter out headers, remove noise and normalize concepts. A multi mode network of associations between people, places, resources, knowledge, tasks, and events is then generated from the resulting concepts and analyzed.

Currently, an analyst will execute a particular sequence of operations using the available tools. In effect, an analyst is manually executing a mental workflow particular to this activity. In addition to having to choose the operations to perform, the analyst also needs to act as the glue between those operations. If different tools are involved, he must manually ensure that data formats match, or are properly translated, and that data is located in the right place.

[2] http://www.i2group.com/us/products-services/analysis-product-line/analysts-notebook

When a nearly identical problem needs to be analyzed, such as developing a model of humanitarian relief for the Chile earthquake also in 2010, a (different) analyst will have to manually follow similar steps as above, but may encounter differences in processing and tools. For example, the source data for Chile may have been scraped from the web instead of from a news source as for Haiti, and so additional cleaning steps to remove the HTML tags are necessary. In the Haiti example, the resulting network may have been analyzed using UCINET. In the modeling of Chile, the analyst may opt instead to use ORA (because of familiarity with ORA, or licensing fees of UCINET, or because she wants to make use of particular capabilities). However, she still wants to be able to use the knowledge in processing and analysis that the analyst for Haiti developed.

The problem for analysts is that a significant duplication of effort is required to conduct very similar analyses, and they need to manually interact with tool operations, and manage tool interoperability. They can share the *results* of their analyses (often as reports), but may only anecdotally share the *processes* for getting those results. The community as a whole requires a platform that supports sharing of knowledge about how to analyze certain data sets, fast turnaround of results when new data is made available or new situations arise, and the ability to work in an exploratory way to tease out and refine conclusions and data.

3. VISION

What is needed in this domain is a unifying platform that supports a number of key requirements:

Heterogeneity: Allows analysts to use a vast range of processing, analysis, data, and report generation services created by multiple tool creators, running on a variety of distributed hosts;

Compositionality: Supports composition of operations, enabling complex and domain-specific analyses and assessments to be constructed;

Reuse: Provides mechanisms to share analysis primitives and compositions, allowing different communities to contribute analyses, data, and tools, without completely rewriting legacy tools;

Usability: Analysis services should be accessible through the web using common interface standards and procedures for discovering available services, and for using them to perform analyses.

Model Discovery: Services should be available for helping analysts identify what models can be applied to what data.

To achieve this vision, over the past three years we have been developing SORASCS (Service Oriented Architecture for Socio Cultural Systems) [10]. Our goal has been to meet these requirements with an integration platform that supports a sustainable ecosystem involving at least three key groups of stakeholders:

Analysts: These are the end users of the system – the people who develop and execute analyses. They range over a spectrum of subcommunities. At one end are users whose primary role is to execute prepackaged analyses. This community has little understanding about the internal analytical working of the tools they use, and are simply interested in the results of some analysis. At the other end are researchers who need to experiment with alternative compositions to develop new forms of analysis that can be use to produce novel insights into some phenomenon, support particular analytical needs of their funders, or package up new analyses to be used by less-skilled analysts. Analysts' primary concerns are those of usability, reuse, and compositionality.

Tool Developers/Integrators. These are the people who create the tools (and in some cases the data sets) that analysts employ to perform analyses. As noted earlier, currently there is a large investment in legacy tools that cannot be practically rewritten. It is expected that tool developers will integrate tool functionality into SORASCS. Furthermore, the body of tools will continue to grow over time. For these reasons, it must be relatively easy to integrate new tools as they become available.

Platform Developers/Maintainers: These are the people who create and maintain the platform over time. While our team was expected to do the initial development, long term, our expectation is to transfer maintenance of the platform to an open-source consortium supported by the community at large. Ease of platform development and cost of maintenance is a central concern for these stakeholders.

In the remainder of this paper we describe how we achieved this vision, starting with a simple open-source SOA solution, and based on lessons learned from that experience, eventually developing a platform that addresses the needs of these stakeholders.

4. FIRST VERSION

4.1 SOA Based Design for SORASCS

On the surface of it, service-oriented architectures (SOA) [15][16] are ideally suited to address the requirements listed above. In particular, standards exist for defining service compositions (termed *orchestrations*), policies for governance, and facilities for service discovery. With respect to the requirements stated above, heterogeneity is supported for both data (through data ontologies and transformations) and analysis mechanisms (through approaches and technologies for migrating legacy code to web services that are defined in common Web Service Definition Language (WSDL) or REST standards). Composition is supported through standard techniques for orchestration, and service registries allow one to discover services that can be used. Reuse is promoted through the distributed deployment of services and orchestrations that are accessible over the internet. Furthermore, there are well-designed standards for security and privacy of data, and high performance SOAs that are designed to work on large data sets are emerging (e.g., Mule).

Given this kind of support from SOA platforms, it seemed natural to us (and to our funders) that basing the SORASCS platform on standard SOA technologies would be an ideal match to the problem at hand.

4.2 SORASCS v1 Design and Implementation

The first design of SORASCS focused on two key aspects: (a) provisioning existing tools as services, and (b) defining example workflows using these services. In providing services, we focused on two kinds of tools, both developed by the CASOS center: Automap and ORA. Automap was chosen because the individual text processing functions were based on batch processing, and so we anticipated that they would be easy to wrap as web services. ORA is a full-featured interactive graphical tool that contains functionality for visualizing networks in addition to providing numerous kinds of analyses. It was chosen because it was representative of other types of analysis tools that exist in the domain, and its integration would provide general lessons that could be used in integrating other tools.

The first version of SORASCS was based on the design of SOA platforms available in industry [12]. Figure 2 depicts the system organization for this version. Each tool was inspected to identify

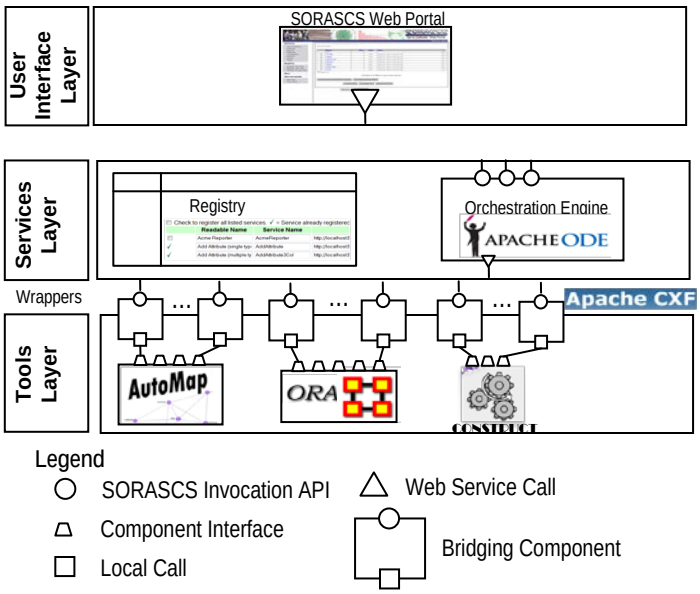

Figure 2. SORASCS v1 System Organization.

distinct functionality to be provided by SORASCS. A bridging component was written to map each function of the tool to a web service API that can be accessed through SORASCS. An Orchestration Engine provided orchestrations as web services in SORASCS, and executed them by controlling how and when constituent services within the orchestration were invoked. A registry allowed tools and clients to find services within SORASCS. Finally, a web portal was developed to allow a user to interact with the services provided by SORASCS via a browser.

4.2.1 Technology Selection

As much as possible open source frameworks that implemented SOA standards were used in this first version. The reason for this was primarily that we desired SORASCS to be an evolvable platform that would actively be worked on and augmented easily by other parties. This excluded commercial SOA technologies. We also desired SORASCS to be built on mature technologies that had an active development community so that problems in the underlying technology were more likely to be addressed.

An important additional selection criterion was support for dynamism, both in service deployment and orchestration. This requirement exists because analysts need to experiment with alternative service configurations, and because integrators would be adding their own services. At the time that we were evaluating technologies (2007), many claimed to support this kind of dynamism, but in fact required rebooting of servers to recognize new orchestrations or services. For these reasons we chose Apache technologies as the basis for our implementation. Specifically, we used:

- Apache CXF, upon which we built service implementations. Apache CXF allows us to present the APIs of existing code as web services. Additionally, it supports mapping service calls in SOAP (or REST) to Java API calls on backend code. It also supports various web standards including those for security, reliable messaging, etc.

- Apache ODE and WS BPEL as the orchestration engine and orchestration language, respectively.

4.2.2 Service Implementation

Once the implementation technology was chosen, we set about designing how to migrate existing tool functionality to services. In an ideal SOA world, services are autonomous: they define explicit boundaries, and communicate via a contract. Because the legacy tools we were integrating were standalone tools, merely wrapping each tool as a web services was insufficient – we had to expose individual functions of the tools. Our implementation effort therefore followed the following process:

Componentization: In this phase, we identified key business functions that needed to be provided by SORASCS. To do this we had to make some changes to the tools in order to access their functionalities (cf., Section 4.3.2). This phase was concerned not only with writing code, but also identifying dependencies on libraries, configuration files, etc.

Service Identification: In this phase we classified services via domain decomposition, which was done in a top-down fashion. First, we looked at network analysis as a domain and the key functions provided by the tools, followed by an inspection of the tool code to determine how the functionality was implemented. This led to a service catalog, where we identified the candidate services, their input and output signatures, and produced a set of high-level categories.

Service Implementation: In this phase, we implemented the catalogs that were previously identified. Our service implementation depended heavily on Apache CXF, which determines how service endpoints are realized.

Identifying Granularity and Categories: In the domain of network analysis, services often require many parameters. One challenge was that many services in certain categories have similar APIs in terms of input and output, but may be configured differently. Had we treated these as different *service types*, we could have ended up with a proliferation of service types that would make composition or interaction difficult, because there would be many special cases. We decided to combine such service types and provide an extensible parameterization API for these configuration parameters.

Identifying Business Processes: Analysts use multiple operations to perform analysis tasks. We can think of these as the business processes of the domain. In this phase, we identified a set of orchestrations to help automate common processes using BPEL, and made them available as services with the SOA.

The above process is similar to the activities that are defined by IBM's SOMA, a method for developing service-oriented architectures [1]. In this version of SORASCS, we categorized the services as follows:

Simple Text Transformation: Operations to process text that require no other information (e.g., removing extra white space);

List-Based Text Transformation: Operations that process text using an auxiliary list, for example, to remove a specified list of words from the text, or normalizing words based on a thesaurus.

Generators: Operations that take text and generate information from them, such as lists of concepts or multi-mode networks.

Reports: Operations that take a network, perform a number of network analysis algorithms on them, and produce reports. For example, operations that run various centrality measures on the network and produce a report identifying key people in the network.

Visualization: Operations that produce documents containing images to visualize the networks graphically.

In this version of SORASCS, we side-stepped the issue of how to treat data in this environment. The operations from the tools were all file based – a fact that we could not change without great effort in reengineering the tools – and so in this version we assumed the existence of a distributed file system to access data.

4.2.3 User Interfaces

Once services and orchestrations were made available through SORASCS, we needed to give users access to them via a user interface. This user interface would (a) allow users to know what services and orchestrations are available in SORASCS, and (b) invoke them and see their results. The SORASCS Web Portal was developed to provide such access. The portal was form-based, defining a form for each service. Users uploaded files that they wanted to use to SORASCS and then located a desired service by browsing through the categories (presented in the left of the figure) and the services in those categories. Once the service completed execution, the user could examine the resulting files, and download them to a local machine. Orchestrations were presented to portal users in the same way as other services (i.e., they appeared in the *Tasks* category on the left, and users filled out a form to interact with them). Orchestrations were written entirely in BPEL.

4.3 Results and Initial Lessons

To evaluate our design we carried out an exercise to integrate a variety of tools, and created some representative analysis scenario use cases involving those services. In addition we held a community meeting where we demonstrated this version of SORASCS to about 50 participants. While the integration of existing tools as services using the platform was generally viewed as being successful we met considerable resistance that caused us to refocus our approach for the second version. In this section, we discuss what we learned from this meeting, and in Section 5 we discuss how our focus changed in the second version. The lessons we learned can be broken into three categories, corresponding to the different SORASCS stakeholders: user lessons, tool developer/integrator lessons, and platform designer/maintainer lessons.

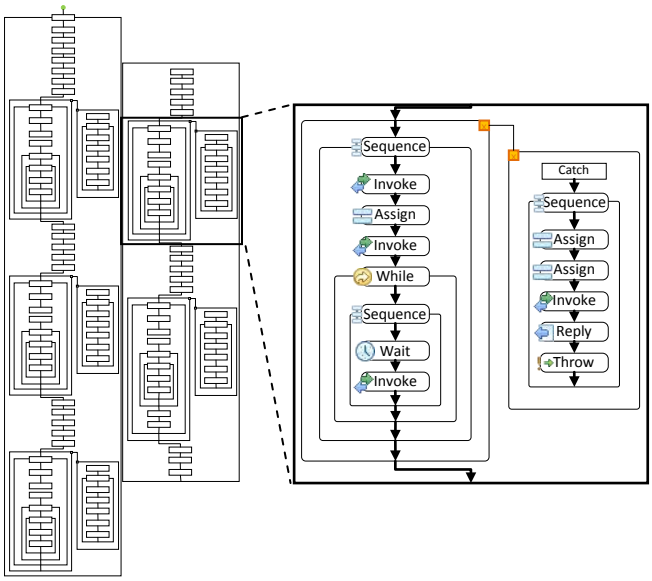

Figure 3. BPEL Orchestrations for HSCB Tasks.

4.3.1 User Lessons

Perhaps not surprisingly, resistance from potential users stemmed from the fact that they were not interested in the underlying architecture but rather in what SORASCS could do for them. In particular, they had the following concerns:

Compositionality. The prepackaged orchestrations that we provided were fine for novices, who wanted to execute turnkey analyses, but expert analysts wanted more configurability of the analyses, and also to be able to construct their own. Having to construct service compositions using notations like BPEL, or even BPMN, require a technical level of knowledge that would distract them from the actual analyses they wanted to do. Figure 3 shows a representation of the BPEL for a simple workflow containing five logical processing steps. As is evident from the sheer number of BPEL operations, even a relatively simple workflow requires a huge number of low-level steps. (These involve assigning the right variables to prepare a service invocation, invoking the service using the service APIs correctly, and dealing with errors.) This was the wrong level at which non-technical analysts wanted to compose their analyses. Additional complications in the BPEL orchestrations include dealing with user interaction, and the iterative nature in which they should be used.

Dynamism. The orchestrations that we defined in this version lacked the dynamism that was needed to accomplish the kind of tasks outlined in Section 2. Specifically, there was no easy way to tailor orchestrations to new, slightly different situations (other than rewriting the BPEL). Also, there was no support for specifying alternative services if particular service instances were not available.

Existing Tools. While some of the functionality of existing tools was a natural fit to services, other functionality was not. For example, it makes sense to develop a service for generating networks from processed source data, but the visualization of those networks is usually done as an interactive process that is not translatable to the stateless nature of services in a SOA. Furthermore, analysts expressed concern that the training they had already received on existing tools did not translate to SORASCS: they wanted to use the tools that they knew, with the interfaces they had invested time in learning, and which for some already had good built-in support for some tasks.

File and Data Management. The location and organization of files both used as sources and generated as part of the analysis was confusing even to some expert analysts. SORASCS exacerbated this problem because users now had to ensure that files were uploaded to SORASCS before they could be processed, and downloaded when they were finished.

4.3.2 Tool Developer/Integrator Lessons

The two tools we chose to initially integrate as services, Automap and ORA, represented two points on a spectrum of the tools encountered in this domain: batch processing tools that perform manipulations on data, and highly interactive tools with rich user interfaces. Our integration efforts with Automap and ORA taught us lessons about the challenges to be addressed when integrating legacy tools. These lessons were:

Separation of UI and Business Logic. Many tools with rich interfaces are designed using a model-view-controller pattern, which serves to decouple the user interface from the application logic. However, in our experience, for many tools the view and

the model are intertwined, making it difficult to create a strictly procedural interface to those tools. This is a challenge that integrators will have to work around. To be able to easily integrate into a web-based system, this separation between the UI and the rest of the system needs to be explicit and consistent.

Deployment Generality. Many legacy systems make assumptions about where libraries and configuration files are located relative to their installation. When deployed as part of a SOA, which has its own conventions about how services should be deployed, these assumptions no longer hold. Integrators and tool developers need to provide a way to configure the location of libraries and configuration files so that they can be deployed within a SOA, for example as part of an application server.

Protected Invocation. The platform needs to be robust when problems and errors arise in invoking tool functionality. For example, if parts of a tool are running in the same process as the platform, the platform must trap exits or crashes to avoid total system failures. Also, when invoking tools in different processes, the platform must take care to manage memory and process limits to handle scalability.

Thread Safety. When developing standalone applications, developers typically only need to be concerned about thread safety of those parts of their system for which it is explicitly required. In a SOA, where multiple requests are likely to be issued simultaneously, parts of the system that were not originally intended to be thread safe may have concurrent use when that service is being invoked by multiple users.

Consequences of Distribution. Providing parts of standalone client applications as distributed services means that integrators must be concerned with issues such as whether a particular user has permission to invoke a service and how to access data. Furthermore, while we provided a parameterization API to configure options for the services, the SOA did not have a way to communicate these options to clients. This led to clients (like the Portal) having to hardcode UIs to allow users to specify these options. Consequently, the UIs had to statically know the services they were interacting with. If a new service was added to SORASCS, any UI tools would need additional code for dealing with this new service.

Our experience with integrating Automap and ORA showed that it was indeed possible to integrate functionality from different kinds of tools as services, using standard SOA technologies. But it also highlighted that some functions of tools (especially the highly interactive and graphical functions) were not as well-suited for provisioning as services. For those kinds of functions, it was more natural to allow the use of the existing tools in some kind of semi-autonomous fashion.

4.3.3 Platform Designer/Maintainer Lessons

We also learned a number of lessons about designing platforms using standard SOA technologies:

Architectural Lock-in. Choosing between different SOA technologies is difficult. Many technologies claim to support web standards, but often the actual features that they support vary between technologies. Partly, this is a function of the maturity of the technologies, and partly this is because the standards themselves are evolving. However, once a technology has been chosen, architectural decisions are forced on the platform that may run counter to the overall platform requirements. For example, when we examined Mule, it had excellent support for large data and automated data transformation, but lacked support for dynamic workflow creation and deployment; Apache technologies provided great support integration of legacy systems, but had limited support for data transformation. Choosing one technology over another means that the platform developers must hand craft the features not well-supported by the underlying technology. Also, the technologies themselves are continually evolving. But, once the choice has been made, switching to a different technology would involve significant platform reengineering. In fact, we were advised numerous times that we should even lock the particular version of the technology that we chose, as upgrading to newer versions might also incur significant cost. (In practice, this has not turned out to be much of an issue.)

SOA Idiosyncrasies. Particular technologies support standards in different ways, which may impact the performance of the platform or even whether particular requirements can be satisfied at all. Unfortunately, discovering these idiosyncrasies requires thorough testing and evaluation of the different candidate technologies. For example, support for dynamic orchestrations differs between technologies: as noted earlier, while most orchestration engines claim to support dynamic deployment of orchestrations, some require the system to be rebooted before the orchestration can be made available to users. Some technologies have limitations on the size of data that can be passed to servers, limitations that are not explicitly stated and in fact depend on timeouts in the protocol, and the bandwidth between the invoker and the service, not on the actual size of the data (i.e., the amount of data that can be transmitted depended on how much of the data could be sent within a particular time, not on any size specification of the data).

Integration Incentives. In addition to the technical integration challenges we encountered, there were also some business concerns that we had not anticipated but that had to be addressed. These issues reduced to the question of why a tool provider would want to integrate a tool into SORASCS. While the CASOS lab provided great support for tool integration, some other tool providers viewed SORASCS as a competing platform to their own, and so were reluctant to participate. Other tool developers feared that if users were able to recreate their own analyses by composing services, funding would no longer be made available for tool developers to hardcode custom workflows in their tools. Furthermore, some tool developers saw the models and theories that they use, as well as the way they are assembled for their user base, as their intellectual property, and so are reluctant to make these available more generally to others.

Ease of Tool Integration. To integrate Automap and ORA we aimed to minimize the reengineering that a potential integrator would have to perform before integrating their tool with SORASCS. The lesson that we learned at this stage was to have the platform perform as much of the mundane functions of integration as possible, such as managing invocation and thread safety, before invoking the functionality of the integrated tools, e.g., the platform can provide thread management facilities, such as queues, to manage the degree of concurrency allowed.

Platform as Domain-Specific Architecture. Most SOA technologies aim to be applicable to general business domains, and have limitations and idiosyncrasies as noted above. Taking these into consideration, it is necessary to augment SOA technology and concepts to particular domains. For SCA this is particularly relevant because it is important that services be assembled by non-technical analysts who have expertise in the domain they are trying to analyze, but little expertise in programming. Thus, one of

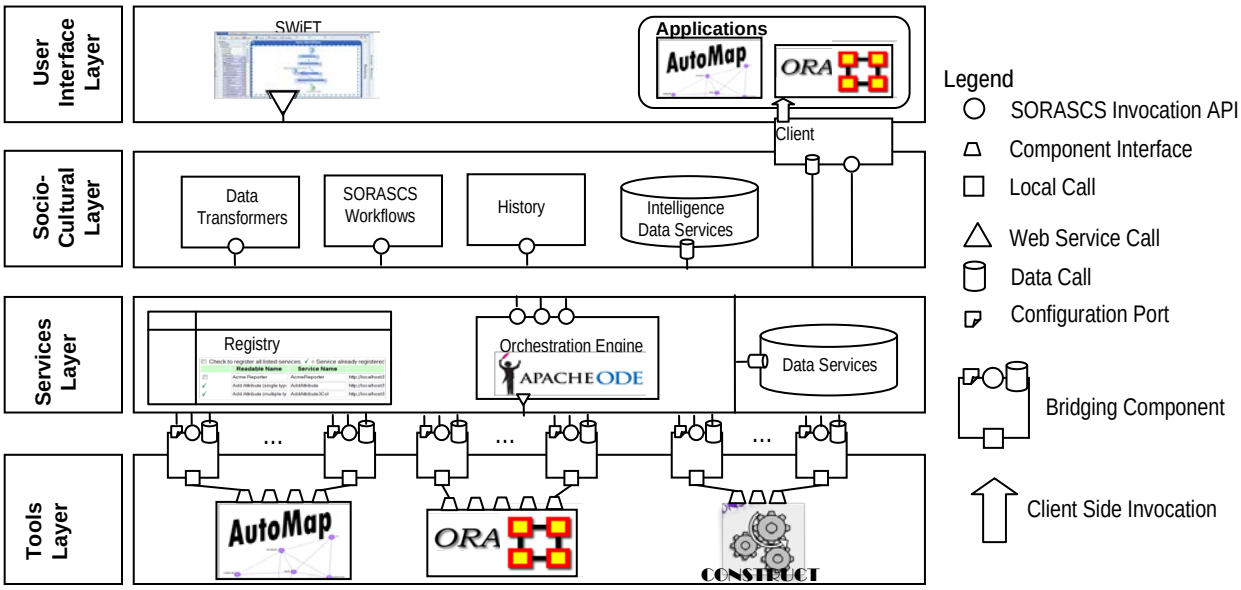

Figure 4. SORASCS v2 System Organization.

the challenges is identifying the abstractions, protocols, and supporting services that should be built on top of SOAs, but that are tailored to the needs of the SCA domain.

5. SECOND VERSION

Building on the lessons learned from the initial version, it was apparent that standard SOA technologies were not sufficient to provide the necessary functionality, usability, and flexibility that was required in this domain, both from a user's perspective and an integrator's perspective. We therefore concentrated on providing abstractions specific to SCA for both of these communities.

5.1 Augmenting with SCA-Specific Layers

As mentioned previously, SOAs provide a great deal of support for putting together distributed heterogeneous systems in a general way. Rather than abandoning SOA technologies and developing our own integration infrastructure, we decided to specialize SOA concepts and abstractions to the SCA domain. Our aim here was twofold: (a) make the abstractions and tools that we presented to users more targeted toward analysts; and (b) make the abstractions for tool integrators and developers better matched to the SCA domain.

To provide better support for users, we needed to add domain-specific layer (the Socio-Cultural Layer) that provides platform services for constructing, executing, and running analyses in a way that did not require analysts to write programs, or their equivalents (e.g., in BPEL). We therefore needed to find the appropriate abstractions to allow analysts to focus on composing their workflows, while still allowing these compositions to be executable on top of a SOA.

An essential requirement that is not met at all by SOAs is the need to use existing standalone tools, not just the functionality of their parts, in concert with finer grained services. Indeed, as mentioned in Section 4.3.1, some tool functionality is not amenable to providing as services. We therefore had to provide support for existing, standalone tools to (a) be used in compositions with other services; and (b) seamlessly use data stored and managed by SORASCS

The fine-grained services that we provided in the first version of SORASCS were still necessary to enable useful workflow composition. However, we improved the integration framework to manage as much of the SORASCS-specific functionality as possible, as we describe below.

We also augmented the services layer to provide additional SORASCS support for managing data, specifying service configuration parameters, and analyzing workflows. In the following sections, we discuss each of these improvements in more detail.

5.2 SORASCS v2 Design and Implementation

Figure 4 shows the system organization for version 2 of SORASCS. In addition to the components in version 1, we have (a) added a *Socio-Cultural Analysis* layer that provides functionality specific to the domain, and (b) augmented the service wrapper layer and the user interface layer with additional functionality.

5.2.1 User Interface Layer

In addition to making the web portal more user-friendly, we concentrated on providing mechanisms to enable the use of existing applications that users are familiar with and tool support for the construction of analysis workflows.

Existing Tools. The ability to use existing tools in a standalone fashion in conjunction with other web services in a SOA is a unique characteristic of SORASCS, and one that is critical to its success. SORASCS provides support for this by including a client program that manages launching already-installed applications on a user's machine. This part of SORASCS also seamlessly manages the flow of data between the client and SORASCS. Applications appear as web services in SORASCS that allow them to be invoked as part of a composition, or from the SORASCS portal. Users can easily add existing tools that can then be used in their compositions. Tool developers can further ease this integration by providing a specification of the command line parameters that the tool can accept to SORASCS as part of the tool's deployment. Furthermore, if a particular application specified in a composition is not available, the user can indicate an alternative tool to use.

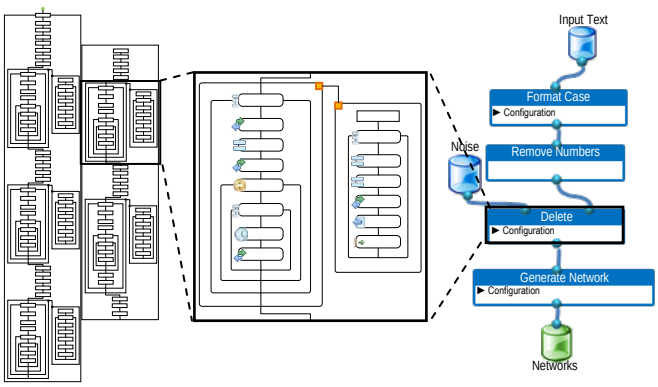

Figure 5. SWiFT Workflow Mapping.

Workflows. In contrast to version 1, where service compositions were defined using orchestrations, in version 2 we introduced the notion of *workflows* as an explicit form of abstraction and representation in SORASCS. Workflows in SORASCS typically follow the pattern of data acquisition, followed by data processing to make it amenable to generating and augmenting networks. Once networks are prepared, services for analyzing, visualizing, or passing to simulations can be invoked. From an analyst's perspective, we think of these workflows as a data flow style. The underlying abstractions of SORASCS workflows are similar to those used in scientific workflows, but do not include control flow abstractions. Users do not have to be concerned with data transformation, or location, because SORASCS takes advantage of services in the new Socio-Cultural layer to automatically locate and apply services to help with this (described in Section 5.2.2). Analysts can define templates, as reusable parameterized workflows that can be used within other workflows to build more complex workflows. To support these activities, we have constructed SWiFT, a web-based tool for rapid construction and execution of these workflows.

Figure 5 shows the SWiFT workflow on the right, and how it is mapped to BPEL on the left. The workflow is a strongly-typed data flow, where the data is matched at the ports. Each service invocation is compiled to a pattern in BPEL that manages the asynchronous invocation of services in SORASCS (using the SORASCS service API), and handles any errors that are returned. Workflow construction is a matter of specifying the ordering of the operations desired and the data that they will use, rather than the programming intricacies of BPEL that was necessary in version 1.

5.2.2 Socio-Cultural Analysis Layer

The SCA layer provides SCA-specific services to support constructing, analyzing, and locating workflows, as well as to provide data management abstractions and types that are specific to SCA activities.

SORASCS Workflow Services. The workflows constructed in SWiFT are translated to component and connector architectural models in a data flow architectural style, where they can be formally analyzed and transformed. Examples of workflow analyses range from performance analysis to a machine-learning based analysis that can advise analysts about service ordering. Examples of transformations include automatically inserting data transformation services, or parallelizing and reordering the sequence to make a workflow more efficient. Finally, workflows are compiled into BPEL orchestrations that are executed on the SOA. The SORASCS Workflow platform service is also responsible for managing the execution of workflows, and the location of appropriate Data Transformer services that can automatically convert data when there are mismatches between the output of one workflow step and the inputs of the next.

History. To facilitate analyses, and for users to examine and repeat activities they have done before, we provide a way to record a series of SORASCS service invocations using a new capability called the History service. Whenever a service or workflow is invoked, the History Service records the inputs and configuration parameters of the call, as well as any results. It also records metadata about the calls, such as the amount of time the service takes to complete, that can be used in workflow analysis. Additional uses of this history information include the ability for users to develop workflows based on their interactions with services, machine learning algorithms to learn and advise about typical service use and ordering, and to enable users to examine how data and reports were generated.

Data Services. To raise the level of abstraction for handing data, version 2 provides new abstractions for: (a) organizing data into projects and categories to better match the way users think about data; (b) typing the data in ways that are informative to analysts, while maintaining the storage of the data itself as files; and (c) tracking the origin of data, whether it was uploaded by a user or created by a particular service.

5.2.3 Service Integration Layer

The service integration layer was augmented to provide built-in support for the following SORASCS housekeeping functions, allowing integrators to focus on wrapping applications [17]:

Web Service Definition. For each category of service, SORASCS provides a standard web service API that is used for invoking the service. For each category, the platform provides classes that map the web service call to a small group of APIs that the integrator must implement in order to integrate their tool. Tool integrators therefore do not need knowledge of the web service standards that are used for calling the tools, or the protocols that are used for invoking the service. They can concentrate on providing the tool functionality as services.

User Authentication. Tool integrators do not need to be concerned with whether SORASCS users have permission to invoke the operation they are wrapping or manipulate the data. The framework takes care of this, and when or if the operation is invoked, they can assume that this has been taken care of.

Data Management. When a service is invoked, any data is transferred to the location of the service using the SORASCS data service, and any changes are automatically synchronized with SORASCS. Tool integrators are therefore free to refer to files, rather than write code to integrate data from SORASCS.

Thread Management and Safe Invocation. Instead of reengineering tools to be thread safe, the platform provides thread management and safe invocation facilities: it provides a queue that manages the requests and feeds them to the tool in sequence. The code that integrators write is called by the SORASCS.

Parameterization. SORASCS provides an XML schema that allows integrators to specify the configuration parameters that can be passed into the service. This specification is stored in the SORASCS registry, and is used by UI tools to build forms for getting the configuration parameters from the user. For example, the portal uses it to generate a web form of the kind in Figure 4;

SWiFT can use it for the same specification, to configure service calls statically, or to get any information required by the user dynamically as the workflow executes.

5.3 Results and Lessons

Once again, after developing this version of SORASCS we migrated existing services, added more services and applications, and held a community meeting to demonstrate the new user interfaces and discuss our progress. To target this community meeting better toward users, we demonstrated the following capabilities:

Integration with Existing Specialized Tools. We identified partners within the community who had existing specialized HSCB tools that could use SORASCS as a back end. One such example is the Visualization of Belief Systems (VIBES)[3] tool, which uses some of the CASOS tools and provides a rich user interface for analyzing beliefs in networks. Originally, the tool was deployed with a version of the CASOS tools, which were invoked by VIBES via a CASOS scripting interface. We were able to use this point of integration, quickly reengineering the part of VIBES that called the script to instead call an equivalent workflow execution in SORASCS. This provided a compelling example of development of a tool targeted for a particular subset of the community, interested in belief analysis.

Workflow User Interfaces. We demonstrated a number of alternative workflow construction tools to give users an idea of how they could use SORASCS to write, run, and share workflows.

The feedback from this meeting was extremely positive.

5.3.1 User Lessons

The new user interfaces more successfully illustrated the possibilities of using SORASCS with existing tools, as well as the value added in having parts of the functionality of the tools implemented as services. However, applications are currently integrated in a coarse-grained fashion: SORASCS manages the data and invokes the tool, but SORASCS does not track what the user does in the tool. Users would like SORASCS to direct the tools (e.g., to bring up the appropriate part of the tool to interact with), and be able to remember what was done in the tool (e.g., how a user customized a certain network to make it visually appealing). While these issues are highly desired by users, they have a significant impact on the integration of tools into SORASCS. This functionality would require tool reengineering to make them record all of their actions, and also to be able to play those recordings *inside* the tool. This is beyond the original scope of SORASCS, but if tools provided this functionality, it could be used by SORASCS to record and construct more interactive workflows.

5.3.2 Tool Developer/Integrator Lessons

Minimizing Integration Code. In our experience with this framework integrators need only write around 50 lines of code to integrate a tool's function, much of which is mapping from the configuration parameters passed in from SORASCS to the data structures required by the tool [17]. This was a significant improvement over the previous version which forced integrators to consider all SORASCS-related issues discussed in Section 5.2.3.

5.3.3 Platform Designer/Maintainer Lessons

Extensibility. Building additional abstractions on top of standard SOA technologies is generally good practice. In SORASCS, we were able to develop a platform that specifically addresses the needs of the SCA community. In doing this, we needed to restrict, in those layers, the way in which services are integrated into SORASCS, and the representation of workflows. However, there is still an opportunity for the underlying SOA technologies to be used directly by developers and maintainers wanting to create different kinds of services or more complicated workflows. The platform is therefore extensible, and over time these new features could be supported directly by SORASCS.

6. STATUS AND ONGOING WORK

SORASCS currently has over 120 services integrated into it. In addition, we are actively developing workflows and workflow templates that use these services. We have successfully integrated applications from various universities, in addition to standard tools like the Microsoft Office Suite.

Currently SORASCS is evolving to strengthen the support for workflows, to provide full support for dynamism as described in Section 2, to add workflow analyses and transformations to help analysts understand workflows and make them more efficient, and better share and tailor their workflows. In addition to workflow-related activities, we are addressing the following issues:

Appropriate User Tools. There is still a gap between what users want, and what is provided, when it comes to constructing and using workflows. Analysts need to be focused on analyzing vast amounts of data and obtaining results quickly. They need help to quickly locate existing tasks that they need for their analysis, or to construct new ones. We anticipate the further work on SWiFT will help to address these issues.

Certification. One important target audience for SORASCS is the intelligence community, and SORASC must be deployable in agencies supporting intelligence, state activities, humanitarian response, and diplomatic missions. To do this, all the code must be certified by those agencies so that it is cleared for field use. This is a long and detailed process, with different concerns and requirements for each kind of deployment. The details of what is required are not specified in a common, easy-to-attain format. The process is eased considerably by using elements already approved. For this reason we attempted to incorporate technologies (such as Apache Tomcat) that have been successfully certified in other contexts. However, we anticipate that full certification will be an issue when the time comes for deployment. Furthermore, the tools and models that could be integrated into SORASCS will vary in what they are certified for, if any. This is another compelling reason for SORASCS to support both web and non-web applications.

Security. Related to certification is the issue of security. We can use standard WS-Security tools in our existing framework to provide authentication, authorization, and data security. One remaining issue of concern is the ability of SORASCS to launch a program on other machines. While this is crucial to the usability of SORASCS, as noted earlier, it is a potential point of vulnerability. As much as possible, we have provided transparency so that a user knows exactly what SORASCS is doing on their machine. However, we are also providing a variety of protocols that can be used between the client and SORASCS, in addition to the current one that allows SORASCS to push instructions to the client machine. For example, we could replace it with a pull mode whereby the client polls SORASCS for any commands that it wants the client to do. However, in this mode we have to consider increased network traffic and decreased response time of tool invocation.

[3] http://www.maad.com/index.pl/visualization_of_belief_systems

Existing web services. In the time that it has taken to develop SORASCS, other related functionality has been provided as web services. For example, some analysts use ArcGIS web services for geospatial analysis. We are currently investigating a lightweight way to wrap these existing web services as SORASCS services.

Simulation. Currently, we have integrated tools that perform text processing and dynamic network analysis. Our next step is to be able to interface with existing agent-based and system dynamic simulation tools in this domain. By extending SORASCS to provide support for running virtual experiments, a host of new requirements are likely to be uncovered, e.g., ability to link to data farming environments, distributed process management, data fusion, and rapid process and movement of large amounts of simulated data. We plan to provide bridges between SORASCS and faster-than-real-time gaming simulation environments like the C2 Wind Tunnel that is used for simulating battle conditions.

Developing a platform that supports a sustainable software ecosystem means satisfying the requirements of all the stakeholders for that platform. We originally thought that using standard SOA technologies would enable SORASCS' sustainability, but instead we had to augment SOA with additional infrastructure that more directly addressed the concerns and requirements of the SCA communities. In particular, we needed to make tool integration easy, and create higher level representations of workflows to support analysis. Not all issues that we encountered were technical; there is a synergy between addressing the needs of the user base and encouraging tool developers to integrate tools – if users are not keen to use a platform, then there is little incentive for developers to use the platform. Furthermore, platform developers need to consider the environment (both technical and business) in which the platform will be used.

While the details of our journey were specific to SCA, many of the lessons we learned likely have general applicability in other domains where dynamic composition of heterogeneous tools and data is required. First, providing a platform for non-technical users will almost certainly likely require a bridging layer to support higher-level abstractions of compositions, and to provide intuitive ways to compose and execute new configurations. Second, for communities that currently rely on the use of interactive tools, it is likely that some scheme will be needed to combine both standalone and service-oriented elements. Third, effective data management is likely to be a critical aspect of any solution. Fourth, tool integration can be dramatically improved by providing a vocabulary of domain-specific adapters that shield the tool integrator from low-level details such as thread safety, error handling, user interface management, and history tracking. Finally, taking an ecosystem view early in the design process will likely lead to a more successful system by clarifying the incentive systems that will support a sustainable and evolvable platform.

ACKNOWLEDGEMENTS

This work was supported in part by the Office of Naval Research - ONR-N000140811223, and the center for Computational Analysis of Social and Organizational Systems (CASOS). The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Office of Naval Research, or the U.S. government. We thank the following people for their contributions: Aparup Banerjee, Jeff Barnes, Dan Chieffallo, Laura Glendenning, Minseong Kim, Frank Kunkel, Yue Lu, Mai Nakayama, Nina Patel, Jeff Reminga, and Hector Rosas.

REFERENCES

[1] Arsanjani, A., Ghosh, S., Allam, A., Abdollah, T., Ganapathy, S., and Holley, K. SOMA: A method for developing service-oriented solutions. IBM Systems Journal, 47(3), 2008.

[2] Borgatti, S., Evrett, M. and Freeman, L., UCINET 6 for Windows: Software for Social Network Analysis User's Guide. Analytic Technologies. 2002.

[3] Carley, K. A Theory of Group Stability. American sociological Review, 56, 331-354, 1991.

[4] Carley, K.M., Dynamic Network Analysis. Dynamic Social Network Modeling and Analysis: Workshop Summary and Papers. Breiger, R., Carley, K., Pattison, P. (eds). Committee on Human Factors, National Research Council, 2003.

[5] Carley, K.M., A Dynamic Network Approach to the Assessment of Terrorist Groups and the Impact of Alternative Courses of Action. In Visualizing Network Information Meeting Proceedings RTO-MP-IST-063, France, 2006.

[6] Carley, K.M., Martin, M.K., and Hirschman, B. The Etiology of Social Change, Topics in Cognitive Science, 1(4), 2009.

[7] Carley, K.M., Reminga, J., Storrick, J., and Columbus, D., ORA User's Guide 2010. Carnegie Mellon University School of Computer Science Institute for Software Research Technical Report CMU-ISR-10-120, 2010.

[8] Carley, K.M., Columbus, D., DeReno, Bigrigg, M. and Kunkel, F. AutoMap User's Guide 2010. Carnegie Mellon University School of Computer Science Institute for Software Research Technical Report CMU-ISR-07-121, 2010.

[9] Department of Veterans Affairs. VistA – HealtheVet Monograph. 2008. http://www4.va.gov/VISTA_MONOGRAPH/docs/2008_2009_VistAHealtheVet_Monograph_FC_0309.pdf

[10] Garlan, D., Carley, K.C., Schmerl, B., Bigrigg, M., and Celiku, O. Using Service-Oriented Architectures for Socio-Cultural Analysis. In *Proceedings of the 21st International Conference on Software Engineering and Knowledge Engineering (SEKE2009)*, Boston, USA, 2009.

[11] Gorton, I., Wynne, A.S., Almquist, J.P., Chatterton, J. The MeDICi Integration Framework: A Platform for High Performance Data Streaming Applications. In *Proc the 7th IEEE/IFIP Working Conference on Software Architecture, 2008.*

[12] Heffner, R. Real World SOA: SOA Platform Case Studies - How Seven Firms Are Building Their Software Infrastructures for SOA, Forrester Research Report, Sept 15, 2005.

[13] Johnson, R.E. Frameworks = (components + patterns). Communications of the ACM, 40(10), 1997.

[14] Messerschmidt, D.G. and Szyperski, C. Software Ecosystems: Understanding an Indispensible Technology and Industry. Cambridge, MA, USA: MIT Press. 2003.

[15] Newcomer, E., Lomov G. *Understanding SOA with Web Services.* Addison Wesley, 2005.

[16] OASIS. OASIS Reference Model for Service Oriented Architecture 1.0. http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn014.pdf. 2006.

[17] Schmerl, B., Bigrigg, M., Garlan, D., and Carley, K.M. Integrating Components into SORASCS. Carnegie Mellon University School of Computer Science Institute for Software Research Technical Report CMU-ISR-10-122, 2010.

[18] Workshop on Software Ecosystems. http://iwesco.wordpress.com