A DESIGN PATTERN FOR AUTOMATIC GENERATION OF WEB SERVICES FROM DOMAIN ONTOLOGIES

Robert Dourandish[1], Nina Zumel[1] and Michael Manno[2]
[1]*Quimba Software, San Mateo, CA, USA*

[2]*Air Force Research Labs, Rome, NY, USA*

Keywords: Distributed Systems, Automatic Systems, Ontology, Web Services, Service-Oriented Architecture, Emergency Response, Syndromic Bio-Surveillance.

Abstract: Web Services and Service-Oriented Architecture have become ubiquitous and are increasingly embedded in every aspect of systems architecture. At the same time, advances in workflow tools now enable us to compose complex new applications by dynamically orchestrating existing web services in new and previously unanticipated execution sequences. The combination of the two is slowly transforming software engineering to a service-centric discipline, with the focus shifting from creating expansive systems to building small, specialized services that can be sequenced, on demand, to support previously unanticipated missions. Implementing and deploying specialized services in this way presents significant challenges in design and programming, as well as long-term maintenance. A fundamental challenge is to maintain the underlying program code long after it has been released and, potentially, incorporated in numerous other processes. This paper presents a methodology and a design pattern to automatically generate web services based on domain ontology. Our approach promises to significantly reduce the programming and maintenance burden of creating and deploying web services, particularly in mission-critical, collaborative, and distributed operations such as emergency response, supply-chain, or healthcare.

1 INTRODUCTION

For the past several years we have been researching massively scaled, multi-organizational, automated information sharing infrastructures, with a focus on emergency response (Dourandish et al., 2006). Emergency response is a uniquely challenging example of a complex, distributed, and networked eco-system because of the extremely broad range of systems, technologies and resources of network participants.

Specifically, we are focused on bio-surveillance as a persistent information exchange during everyday, emergency, or disaster healthcare support: that is, whenever emergency help is summoned, or when a patient seeks emergency or non-emergency care from a health care provider. This type of surveillance is known as syndromic surveillance: surveillance using health-related data to signal that there is sufficient probability of a disease outbreak to warrant further public health response. Though historically syndromic surveillance has been utilized to target investigation of potential epidemics, its utility for detecting outbreaks associated with bioterrorism is also increasingly being explored by public health officials (Buehler and Hopkins, 2004).

2 DOMAIN CHALLENGES

As a discipline, Emergency Response has a number of unique properties that make software development for the domain a challenging task. Chief amongst these attributes is the fact that emergency response is a collaborative operation that often includes multiple organizations and disciplines (Bruinsma and Hoog, 2006) such as a fire department, an ambulance company, and one or more hospitals. In addition, larger response operations often require a coalition of responders from multiple jurisdictions. As a result, each response operation could be subject to multiple legal regulations, local policies, or court-mandated considerations (Bui et al., 2006). Furthermore, each

Dourandish R., Zumel N. and Manno M. (2007).
A DESIGN PATTERN FOR AUTOMATIC GENERATION OF WEB SERVICES FROM DOMAIN ONTOLOGIES.
In *Proceedings of the Third International Conference on Web Information Systems and Technologies - Internet Technology*, pages 341-348
DOI: 10.5220/0001265703410348
Copyright © SciTePress

participating response organizations, indeed each responder, could have different operational protocols, resources, or capabilities. It is clear that enumerating all possible combination of responder, legal or policy umbrellas, and operating protocols is not feasible.

This domain therefore presents significant programming and automation challenges. The complexity is due to the heterogeneity of the overall eco-system, and to the number of participants in a typical emergency response network, who may be called upon to handle a wide variety of possible events. Such scenarios can range from common events such as vehicle accidents, heart attacks, gunshot wounds, or childbirth, to more catastrophic events such as hazardous material release, hurricanes, or earthquakes.

The key research question is how to implement the domain knowledge in such a way that implementations can easily be localized and be contextually responsive (as much as possible) to the broad nature and scope of emergency calls. While these questions are universal to most domains, they take on an added level of complexity for emergency response:

- The domain is protocol driven. While these protocols, such as how to treat a potential heart attack patient, are reasonably standard, they can differ slightly from location to location. These variations are typically due to medical direction, proximity to a major hospital, or legal issues. Nonetheless, these local protocols are important and any programming in this domain must facilitate possibly many minor variations, even within a small geographic area.
- The domain is governed by a large number of local to global regulations dealing with a wide range of policy issues such as privacy, legal casework, and accepted local customs or practices.
- Multi-jurisdictional collaboration during emergency situations is often abruptly initiated, staffed on an as-available, ad-hoc basis, and without any guarantees that the joint coalition response personnel have previously trained together. In the case of international response, the responder coalition may not share the same protocols or the situational and operational context in order to quickly adapt their own.
- The domain lacks a standard lexicon that is shared by all participants.

The above attributes highlight the complexity of programming for this domain. To address these challenges our research focuses on knowledge-based strategies, as opposed to requirements-based engineering. Specifically we focus on methods of encapsulating and utilizing expert knowledge as the foundation of software specification and implementation. To accomplish this objective, we have further subdivided the domain into a set of *response protocols* and a series of *core operations* to support those protocols. We were able to take advantage of this distinction to implement the system as two components: (1) a foundational platform, implementing core services, and (2) a series of domain-level services that run on the platform, e.g. the response protocols. The former included basic messaging and transactional[1] support while the latter implemented expert knowledge.

3 FOUNDATIONAL PLATFORM

The foundational platform is based on a variation of the portal architecture pattern coupled with a logical specialized P2P network overlaid on a Service-Oriented Architecture (SOA). This architectural approach effectively creates distributed collaborative P2P portals with specialized applications, e.g. emergency response. Because data exchange is a native component of the portal pattern, the architecture can also enhance distributed data mining in support of secondary applications such as bio-surveillance. The Service-Oriented approach also means that portals are able to contribute data to other applications on as needed or ad-hoc basis.

Finally, *the* key issue in emergency response, particularly multi-jurisdiction response to large incidents, is Command and Control (Veelen et al., 2006). We use a hierarchical network topology that allows participants to assume roles of a "parent node", or a "child node" within a given context. This approach implements local Command and Control (C2) of resources assigned to each node while enabling creation of dynamic C2 as warranted, such as in response to large-scale emergencies. Figure 1 shows the technology stack of the foundational platform. The stack *generically* provides a platform for messaging, data transfer, data privacy and security, and computing (e.g. cluster) services.

A Service Oriented Architecture (SOA) was the

[1] A detailed discussion of the transaction processing core of the platform is beyond the scope of this paper. However, it should be noted that the transaction code conforms to a typical transaction design pattern.

only logical choice to create a common operational environment that did not require a priori technical or policy agreement between all participants. Furthermore, the SOA was the only approach that would allow all responders to use their existing infrastructures, particularly the wide-area connectivity afforded via the Internet. Using IP-based networks inside the firewall, also a common practice, allowed us to implement the services without the need to distinguish where they were running (e.g. inside or outside an organization's network) and secure the communication using standard strategies such as VPN.

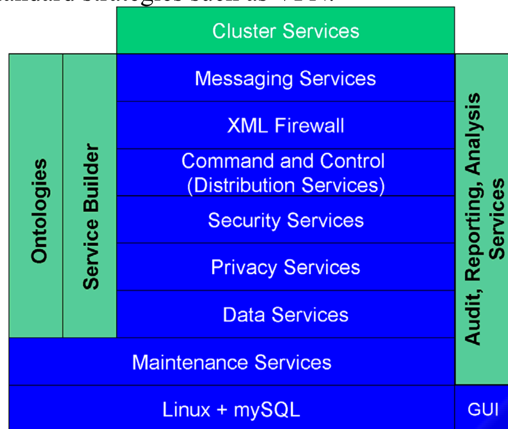

Figure 1: Foundational Platform Technology Stack.

Employing a Service-Oriented Architecture also solves the challenge of duplicating the infrastructure underpinnings, particularly with respect to ad-hoc coalitions that are formed in response to specific incidents. To become a part of a responder network all an organization needs to do is to operate the stack shown in Figure 1.

4 DOMAIN LEVEL SERVICES

Once system-level and deployment issues are addressed through the Foundational Platform, what remains is the domain-specific programming of services that will be delivered over this network.

In our approach, a basic model of the domain is implemented using ontologies, and various strategies are used to create the context for an event that occurs within the domain, and to determine the appropriate way of handling that event. While this approach has proven successful (Stojanovic et al., 2004) our approach does not implement a single ontology. Instead we chose to model the world using numerous small models, each represented by a different ontology. In doing so, we sought to extract and express Subject Matter Expert (SME) knowledge, in very specific and concisely-defined areas, and use the resulting ontology in lieu of engineering requirements for Web Service code.

5 DESIGN PATTERN

After researching various approaches to tightly integrate ontologies with software engineering, we created a logical design pattern as depicted in figure 2. The key attribute of this pattern is using XML as the bridge between expert-designed ontologies and engineer-implemented systems. This pattern is designed for the domain layer (Fowler et al., 2003) of the system and assumes an ontology editor tool that is used by the Subject Matter Expert (SME) to specify the Ontology. In our research we used Protégé, an open source ontology editor and knowledge acquisition system developed by Stanford University.

The pattern illustrated in Figure 2 reflects a core operational doctrine that domain experts should be in full control of a service's behavior. There are two implications resulting from this choice. First, various domain operations must be described in small enough steps that are both manageable in terms of ontology description and meaningful in terms of operational deployment. Second, it is possible to create a broad foundation that can support SME-described ontologies. The emergency response domain exhibits both these attributes: Fundamentally, the domain is protocol-driven and individual protocols are combined to compose more complex protocols in response to particular scenarios or authorized practice level of responders. Within this domain, there is also a set of core messaging and communications practices, commonly referred to as the "10 codes" that largely systemize the communication paradigm and provide some level of lexical consistency amongst responders[2]. In our research, we implemented the "10 codes" using the foundational platform previously described. Higher level response operations are defined and executed in accordance with the design pattern as follows.

[2] This is an extremely broad statement. There are significant inconsistencies in use of 10 codes from jurisdiction to jurisdiction.

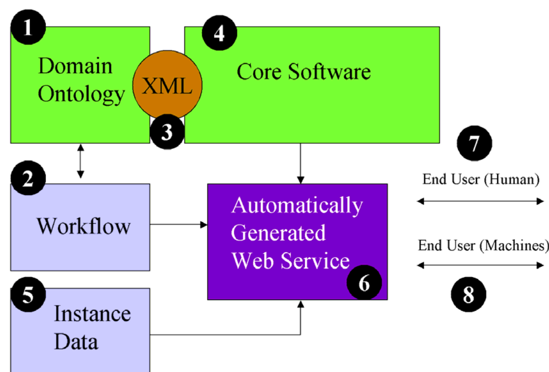

Figure 2: Design Pattern.

The first step, **(1)**, is for an expert to use an ontology editor to define an ontology. This ontology will eventually be transformed to an executable Web Service, invoked by humans or machines, **(7, 8)** through a normal Web Service interface, for example via SOAP messages. The ontology editor we used, Protégé, is capable of generating RDF-based XML, which we further processed to achieve a simpler syntax, **(3)**. The domain ontology will include the explicit components of workflow, **(2)**, for example protocol steps. The implicit workflow elements, such as acknowledging command and control messages, are implemented as part of the foundational platform.

The XML representing domain knowledge and protocol workflow steps are processed by our core software, **(4)**, which augments the XML with references to implicit workflow steps, and incorporates database or other references to include explicit instances of domain components, such as individuals, equipment, or locations. In computer science terms, the core software packages the XML into a series of appropriate objects (classes, methods, etc.) organized using a transaction object. The result of this step is a Web Service, **(6)**, that is executable over the solution platform. Invoking the service will, in effect, execute the SME-defined domain ontology.

6 AN EXAMPLE

The design pattern is exemplified through the following "call for help" scenario. This example demonstrates a call to a central location (911 in the United States) that is typically contacted when help is needed. The response protocol is used here to illustrate the underlying principles of our research.

6.1 The Domain Ontology

Figure 3 above shows the *911 Call* ontology. The definition includes classes, such as *Event* and subclasses, such as *Medical Event,* as well as other relevant components for an emergency medical response, such as personnel, the required level of training, and what class of responders (e.g. Fire or Ambulance or both) may respond to this type of event.

6.2 The XML

As stated previously, the next step in our methodology is to process the ontology into XML. While the Protégé ontology development environment is capable of producing XML description of the ontology, we found it easier to process it for clarity – the version that is discussed here.

```
(1) <quimbaService name="911 Call">
(2)  <metadata name="Event">
(3)       <attribute name="EventID" />
    <attribute name="EventDateTime" />
          <attribute name="EventLocation" />
          <attribute name="ResponderType" />
          <attribute name="ResponseMode" />
(4)       <datamap index="0" value= "this.EventID" />
(3a)      <datamap index="1" value="Emergency" />
        <datamap index="2" value="this.EventLocation" />
        <datamap index="3" value="this.EventDateTime" />
</metadata>
(5) <metadata name="MedicalEvent" typeof="Event">
<attribute name="CaseID" />
<attribute name="PersonnelType" />
          <attribute name="CaregiverType" />
          <attribute name="EquimentType" />
          <attribute name="EndCode" >
            <default type="C2" code="Qxx" />
          </attribute>
          <datamap index="1" value="MedicalEvent" />
</metadata>
(6) <metadata name = "CareGiver">
(7)       <attribute name="isPerson" />
(8)       <attribute name="hasCertification" />

          <attribute name="hasAuthorizedPracticeArea" />
          <attribute name="hasSkills" />
          <attribute name="hasTraining" />
</metadata>
```

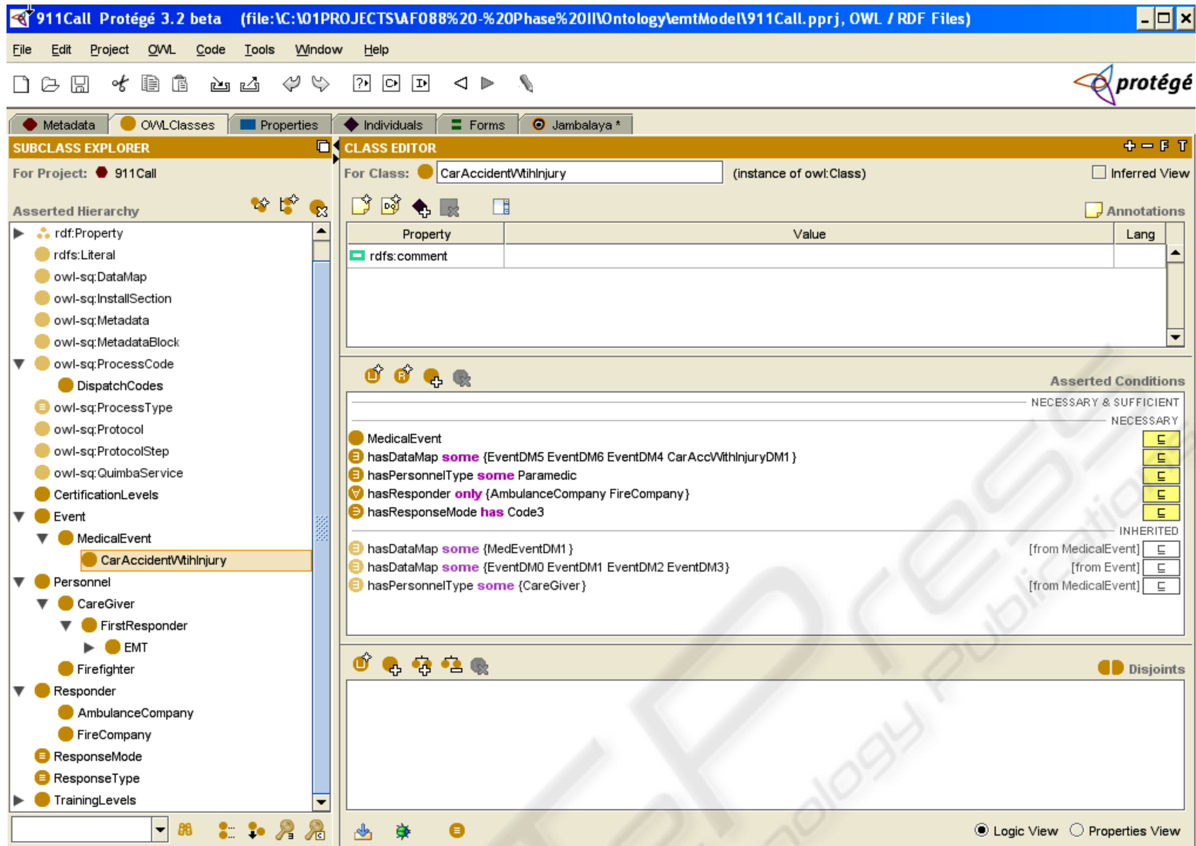

Figure 3: Example Ontology Developed in Protégé.

```
<metadata name="Responder">
          <attribute name="hasName" />
          <attribute name="hasEquipment" />
          <attribute name="hasResponseType" />
          <attribute name="hasResponseMode" />
</metadata>
(9a) <metadata name="EMT" typeof="CareGiver">
          <attribute name="hasCertification" value="EMT-
B" />
</metadata>

(9b) <metadata name="Paramedic" typeof="EMT">
   <attribute name="hasCertification" value="EMT-P" />
   <attribute name="hasTraining" value="EMT-B" />
</metadata>

(10) <metadata name = "FireCompany"
typeof="Responder">
    <attribute name="hasFirefighter" />
   <attribute name="hasCaregiver" typeof="CareGiver" />
          <attribute name="hasResponseArea" />
</metadata>
<metadata name = "AmbulanceCompany"
typeof="Responder">
   <attribute name="hasCaregiver" typeof="CareGiver" />
   <attribute name="ResponseType" value="Medical" />
</metadata>
(11) <metadata name ="CarAccidentWithInjury"
typeof="MedicalEvent">
          <ResponderType value="FireCompany" />
          <ResponderType value="AmbulanceCompany" />
          <PersonnelType value="Paramedic" />
          <ResponseMode value="Code3" />
(11a) <datamap index="1"
value="CarAccidentWithInjury" />
          <datamap index="4"
value="set(this.ResponderType)" />
          <datemap index="5"
value="set(this.PersonnelType)" />
          <datamap index="6"
value="set(this.EquipmentType)" />
</metadata>
(12) <protocol name="dispatch">
(13)      <step num="1" id="s1" name="Get Information"
prompt="What is your Emergency" options="Event"/>
(15)      <step num="2" id="s2" name="Dispatch
          Responder"
(15a)Type="C2"
(15b) Code="Q09"
(15c) input="this.datamap" />
</protocol>
</quimbaService> <!-- //911 dispatch -->
```

The XML is enclosed in a tag, quimbaService **(1)** that names the Web Service, e.g. *911_Call*, as it will be accessed. By convention any space in the service name will be replaced with an underscore.

Next is a series of metadata **(2)** tags that represent the classes defined by the ontology, as well as their hierarchy and attributes **(3)**. As per Object Oriented Design standards, child classes inherit their parents' attributes. Parent attributes are overridden if they also appear in the child's definition, as shown in **(3a)** and **(11a)**. Each class may have multiple parents. A subclass relationship is signaled using the *typeof* directive, as shown between *Event* **(2)**, *Medical Event* **(5)**, and *Car Accident with Injury* **(11)**. Some metadata elements such as *Care Giver* **(6)** include higher-level knowledge representation concepts such as a *person* **(7)**. It is important to note that the ontology is viable and operable without such high level definitions. However, since a responder can become a caregiver and a patient as the event progresses, reasoning about generic terms is useful. The implication here of course is that the Subject Matter Expert designing the ontology may need a bit of training or, alternatively, may be teamed up with a knowledge engineer. As with any domain, emergency response includes several taxonomical concepts, such as the hierarchy and certification levels for the Emergency Medical Technicians (EMTs). The fact that a certification is a requirement for an individual to be considered a caregiver is expressed by a *has* attribute **(8)**, and the different skill levels are expressed by combining *hasCertificate* and *hasTraining* attributes in the metadata, as shown in **(9a)** and **(9b)**. The execution has two dimensions – data and process, or steps. The data generated from instantiating the ontology (i.e. applying it to a particular 911 call) is collected using the *datamap* tag **(4)**. This tag mirrors the data filed in the underlying transaction object that is implemented as part of the foundational platform -- the data that is contained in the transaction are, in effect, the parameters that are passed from process to process in order to affect the execution of the Web Service. The second execution component, the process steps, are enumerated using the *step* tag **(13)** which, as shown, may include actual prompts that are presented to the user or calls to the components of the underlying platform **(15)**. In this specific case, the ontology designer is calling the underlying Command and Control ("C2") element **(15a)** of the platform, invoking the dispatch protocol **(15b)**, and causing the instantiated data to be passed to it **(15c)**. When the service is executed, the underlying platform will broadcast the dispatch request ("Q09") to all responders by creating a transaction and passing it to a service that represents each responder. Each responder service will evaluate the data and determine whether or not it can respond to this request ("transaction") and report back accordingly. The net result is that a responder is dispatched as the result of the 911 call.

6.3 Executing the Ontology

To execute the ontology as represented by the final XML, we currently deploy the XML to a web service that implements a "service runner". This service runner is an interpreter and executes the directives, much the same way a PHP interpreter would execute PHP syntax. Each directive, such as metadata, has a specific handler that is domain independent. This interpreter does not understand the emergency response domain, any more than the PHP interpreter understands a particular application. It simply collects, routes, and otherwise manages data and message traffic amongst participating nodes. These activities change the state of the network – thereby producing actions. Because the service-runner itself is a web service and since the XML drives the entire execution process, the entire system is accessible to any program that complies to Service Oriented Architecture.

7 LESSONS LEARNED AND FUTURE DIRECTION

An early phase of our research utilized ontologies as requirements engineering instrument. In that mode, the ontology was developed by a subject matter expert, validated through the Protégé knowledge base query mechanism, and supplied to a programming team to generate the equivalent web service. While this process initially worked well, we quickly discovered that the two – the core Web Services and the domain ontology – rapidly diverged.

Fundamentally, the reason behind the divergence was the fact that the two efforts, the ontology development and the system engineering, were two separate tasks with no physical relationship[3]. Although a relatively small effort, the initial phase

[3] We use the word "physical" to call attention to the fact that there was a logical relationship between the two.

of our research proved that if ontologies are to significantly and seriously contribute to production-grade, system-scale engineering effort, they must become an integral part of the system, in the same way as data models are. In the case of a data model, a physical component, i.e. the database, binds the data model with the system that uses it. There is no such equivalence in knowledge engineering.

Our research has proven that it is possible to generate Web Services directly from domain ontologies designed by Subject Matter Experts. While still in its infancy, this research has the potential to significantly impact distributed computing, at least in some domains, where collaborating Service-Oriented components can be generated based on domain ontologies. If proven scalable, the work may lead to ushering of a new paradigm in software development with a significant shift to knowledge engineering as the foundation of software design, with the potential of replacing requirements engineering.

While we were extremely successful in demonstrating automatic generation of web services based on domain ontologies, the research also revealed a number of challenges.

In our opinion the foundational challenge is the lack of standard knowledge engineering methodologies. We believe knowledge engineering, as a discipline, needs a level of practicable systemization similar to Object Oriented Design. Lack of such accepted methodologies will not only lead to repeat of the "islands of technology" phenomenon that the community experienced with databases and operating systems, but also will create interoperability and utility issues. A key dictum of any knowledge engineering methodology should in our opinion be componentization and reuse – two objectives the software engineering community is now actively pursuing. Finally, knowledge engineering, in our opinion, requires tools that can be used by Subject Matter Experts with little or no knowledge of programming.

Our research also revealed a number of challenges that are unique to using knowledge as the foundational driver for software: because services are generated automatically, the ontology will have a multiplying effect in either the utility or dysfunction of the resulting code. As such, tools that simulate ontologies and help "debug" the ontology are extremely important (Easterbrook, 1991). Furthermore, because Services are not collocated, the distributed aspect of the system becomes a key issue in operations that orchestrate existing web services for a new purpose. As such, versioning, audit, validation, and identity management take an entirely new and complex role. Finally, because different experts can have different views of the world, an ontology conflict resolution mechanism must be incorporated in the underlying platform. Because we use numerous small ontolgies to model the world, we also need to research conflict resolution strategies. While the automation adds a new level of complexity, this however is not a new concept in computing (Easterbrook, 1991); (Klein, 1991); (Sycara, 1993); (Fang et al., 1993). The key research challenge here is the real-time, mission-critical aspect of conflict resolution.

Our immediate future direction is two-fold:

Our first goal is to study scalability issues in both deployment and maintainability. There are, for example, active questions in dealing with the "weakest link" in an environment where the operation being executed is orchestrating web services that are not collocated. The key question here is whether or not a mechanism can be created to negotiate and maintain a level of service between collaborating web services. If so, what does that mechanism look like in practical terms? Can it be automatically negotiated or do we need humans in the loop, at least in parts of the process?

Secondly, we plan to study the next generation of our service-runner that may generate Java classes that are directly executable. To be a feasible operational solution, the research would have to address a number of additional challenges, ranging from Java Virtual Machine (JVM) compatibility to dealing with updates of statically linked code.

ACKNOWLEDGEMENTS

This work was supported in part by a Small Business Innovation Research award from the US Air Force, under contract number FA8750-05-C-0085. The views and opinions presented in this paper represent the views of the authors and do not, necessarily, represent the views of the DoD.

REFERENCES

Dourandish, R., Zumel, N., Manno, M., "Automated Military-Civilian Information Sharing", Military Communications Conference (MILCOM), 2nd IEEE Workshop on Situation Management (SIMA), Washington, D.C., 2006.

J. Buehler, W., Hopkins, R.S., Overhage, J.M, Sosin, D.M., and Tong, V. "Framework for Evaluating Public Health Surveillance Systems for Early Detection of Outbreaks," *MMWR, 53(RR5):1*, 2004.

Bruinsma, G., de Hoog, R., "Exploring Protocols for Multidisciplinary Disaster Response Using Adaptive Workflow Simulations", Information System for Crisis Response and Management, ISCRAM, Newark, NJ, 2006.

Bui, Tung, Sankaran, S., "Foundations fro Designing Global Emergency Response Systems", Information System for Crisis Response and Management, ISCRAM, Newark, NJ, 2006.

Van Veelen, J.B., et. al, "Effective and Efficient Coordination Strategies for Agile Crisis Response Organizations", Information System for Crisis Response and Management, ISCRAM, Newark, NJ, 2006.

Easterbrook, S., "Handling Conflict Between Domain Descriptions With Computer-Supported Negotiation", *Knowledge Acquisition: An International Journal, 3*:255-289, 1991.

Klein, M., Supporting Conflict Resolution in Cooperative Design Systems, *IEEE Transactions on Systems, Man and Cybernetics, 21(6)*, 1991.

Sycara, K.P., Machine learning for intelligent support of conflict resolution, *Decision Support Systems, 10(2)*:121-136, 1993.

Fang, L., Hipel, K.W., and Kilgour, D.M., *Interactive Decision Making: The Graph Model for Conflict Resolution*, Wiley, 1993.

Fowler, M., et. al, "Patterns of Enterprise Application Architecture", Addison-Wesley, Boston, MA, 2003.

Stojanovic, L., et. al, "The role of ontologies in automatic computing systems", IBM Systems Journal, Vol 43, No 3:598-616, 2004.