OTTER PROJECT
Ontology Technology that Executes Real-time: Project Status

Thomas A. Tinsley
Tinsley Innovations, 7337 Otter Creek Drive, New Port Richey, Florida, U.S.A.

Keywords: OWL, Protégé, Service Component Architecture, Service Data Object, Data Access Service.

Abstract: Enterprise Architecture models are often overlooked or bypassed during information systems development. This usually results in complicating application integration and data sharing which can increase cost and cause problems. The OTTER project solves this problem by combining Enterprise Architecture principles, ontology reasoning, and Service Component Architecture. Together, this makes the Enterprise Architecture the foundation for component service development and execution.
Protégé is used to define the layers of the Enterprise Architecture. These layers are mapped to Service Component Architecture standards to provide real-time execution of processes. Information access and service component access are both provided by OTTER using OWL data expressions. This use of OWL data expressions is an alternative to using XML web services for service access and SQL for relational database access.

1 INTRODUCTION

The OTTER prototype integrates the proven application of Enterprise Architecture (EA) principles, OWL business definitions, and Service Component Architecture (SCA) as shown in "Figure 1: Executable Enterprise Architecture." This integration turns the otherwise static models of EA into executable patterns. This is done by bridging the class expression language of OWL to the Service Data Object standard of Service Component Architecture (SCA).

Applying Enterprise Architecture as a foundation for development and execution was not found in other publications after months of research. The Otter project was then initiated to prove the validity of this approach.

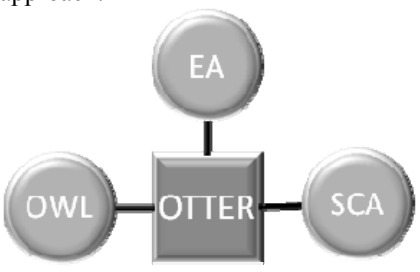

Figure 1: Executable Enterprise Architecture.

The real value of OTTER is in the layered extensibility of the executable EA patterns. Through extensions, business principles can be captured and very specific business applications can be defined. The "include ontology" capability of OWL provides the support for layering the architecture with each layer defined as a separate OWL ontology.

OWL provides an outstanding language for defining things and axioms that can be tested through reasoning. This capability has been proven by its application in multiple industries. This also makes it an excellent choice for defining an Enterprise Architecture.

Many major providers of infrastructure software have adopted SCA as their component model. These providers include IBM, Oracle, and SAP. They have adopted this standard across all of their infrastructure products.

The OTTER prototype creates a bridge between OWL and SCA by mapping OWL class expressions to the SCA standard of Service Data Objects (SDO). By bridging at this technology level, business service messages and component access to data is accomplished using class expressions. This replaces the need for XML to define Web Services and SQL to access data repositories.

The executable capability of the EA pattern and the patterns that extend it change these static models

A. Tinsley T..
OTTER PROJECT - Ontology Technology that Executes Real-time: Project Status.
DOI: 10.5220/0003719305210527
In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development* (SSEO-2011), pages 521-527
ISBN: 978-989-8425-80-5

to implemented services. These implementations can be applied and reused by multiple applications.

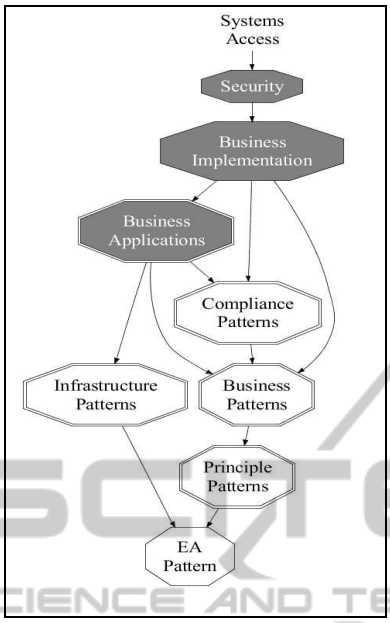

Figure 2: Layered Ontologies.

The EA Pattern provides the classifications and high-level property definitions for all of the components within the business ontology. An overview of how the models are layered is shown in "Figure 2: Layered Ontologies." The shaded patterns only contain individuals. The double-lined octagons indicate multiple ontologies while the single-lined octagon indicates only a single ontology.

The principles and rules for operating specific vertical lines of business would be defined by extending the EA Pattern. Principle patterns such as Accounting, Marketing, and Finance could be leveraged by multiple vertical businesses. Multiple vertical lines of business models can be combined for automatic integration.

Access to the systems would be given based upon the defined authentication and authorization defined by the security pattern.

2 GOALS

The OTTER project has a primary goal of transforming static EA models into executable patterns in a secure environment. This primary goal is accomplished by applying the following:

- **Provide the Infrastructure for Real-time Execution.** Components defined in ontologies that extend business ontologies should be executable. For each business model, there could be multiple execution model options.
- **Use Industry Standards for Information Technology and Ontology.** The industry standard of Service Component Architecture will be used to define components and the standard for Service Data Objects will be used to define information access.
- **Security should be provided for Online Access, Process Access, and Information Access.** Security is a primary concern and has been included at the earliest stage of the project.

3 DELIVERABLES

The OTTER project has six primary delivery objectives:

- A core Enterprise Architecture (EA) pattern.
- Examples verifying the use of the EA pattern.
- Protégé plugins for viewing from the EA perspective and initiating an HTTP server.
- Prototype executable components using OWL Protégé, Service Data Objects, and Service Component Architecture.
- Security for browser access, process, and information.

3.1 A Core Enterprise Architecture (EA) Pattern

The EA Pattern currently used in the prototype has the high-level class structure defined in Protégé as shown in "Figure 3: Enterprise Architecture Pattern."

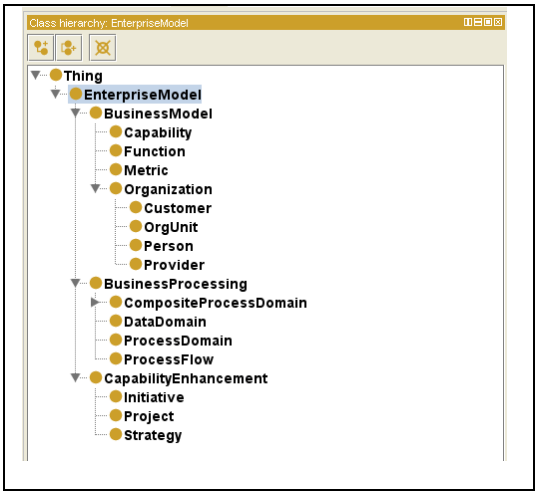

Figure 3: Enterprise Architecture Pattern.

The Enterprise Model is divided into three major categories. The first one, Business Model, contains the information about the business capabilities, functions, metrics, and organization. The second category, Business Processing, describes the processes used by the business, the data used by the processes, the flow of information, and the composite processes that may or may not be automated. The third category, Capability Enhancement, includes the strategies, initiatives, and projects to enhance the capabilities of the organization.

3.2 Example Verifying the Use of the EA Pattern

As a test to validate the EA Pattern, the published form of the Association of Retail Technology Standards (ARTS) model was used as the base model to create multiple OWL files.

- This model includes the high-level processes of Store, Distribution, and Home. Within these processes are ten sub-processes and forty-two information flows.
- The data model includes nineteen subject areas and the subject areas include forty-seven sub-subject areas.
- Additionally, twenty composite components were defined that referenced the processes, flows, and data.

3.3 Protégé Plugins

There are four plugins implemented for EA viewing. They are included in Protégé under the tab "EA Pattern".

3.3.1 Data Domain

All data properties associated with the selected class are shown in this plugin. This includes the data and object properties of super classes. There are tabs for keys, data properties, and range object properties as shown below in the examples in "Figure 4: Data Domain Keys Tab.", "Figure 5: Data Domain Data Tab.", and "Figure 6: Data Domain Range Tab."

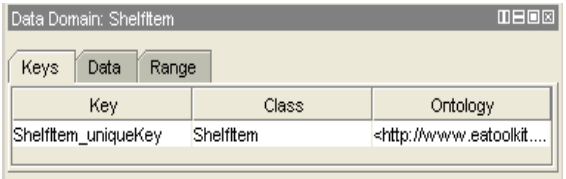

Figure 4: Data Domain Keys Tab.

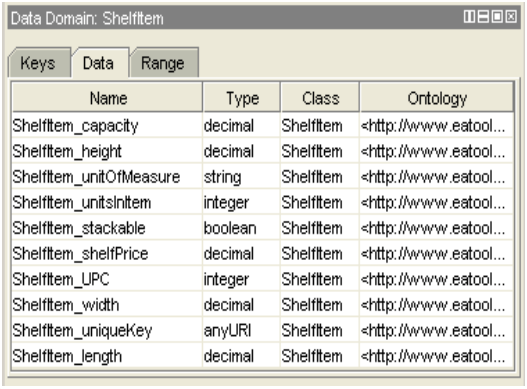

Figure 5: Data Domain Data Tab.

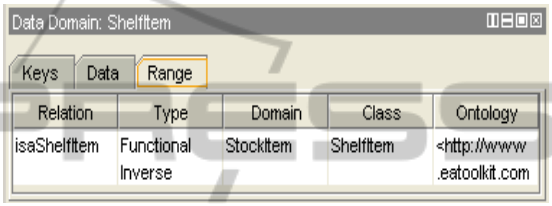

Figure 6: Data Domain Range Tab.

3.3.2 Process Domain

This view shows the processes using two tabs. One shows the sources of information for a process as shown in "Figure 7: Process Domain Sources Tab." and the other tab shows the consumers of information from the process shown in "Figure 8: Process Domain Consumers Tab."

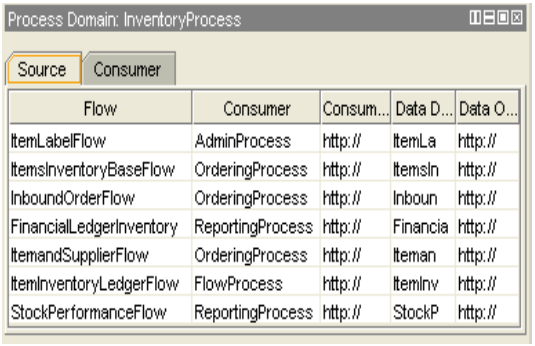

Figure 7: Process Domain Sources Tab.

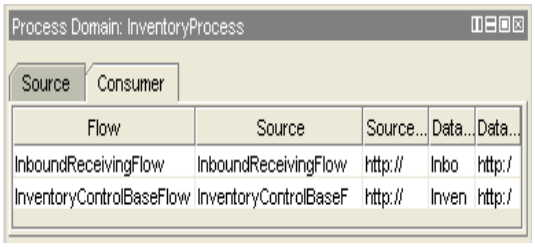

Figure 8: Process Domain Consumers Tab.

3.3.3 Process Flow

This view only has one tab that shows the sources and consumers of an information flow. This is demonstrated in "Figure 9: Process Flow."

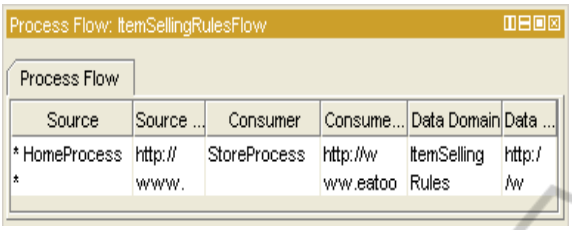

Figure 9: Process Flow.

3.3.4 EA Server

This view includes a log of alerts for the HTTP server shown in the example in "Figure 10: Server View."

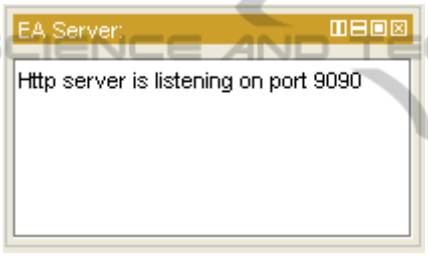

Figure 10: Server View.

3.3.5 Prototype Implementation

The Prototype uses OWL Protégé, Service Data Objects, and Service Component Architecture. It demonstrates the functional integration of OWL with the standards of Service Data Objects and Service Component Architecture.

Service Data Objects

Service Data Objects (SDO) is a specification for an application interface for accessing data to be used by programmers. The specification was included as part of the Service Component Architecture specification. These specifications were developed in coordination with experts from BEA Systems, IBM Corporation, Rogue Wave Software, SAP AG, and Xcalia.

The SDO specification was created to describe complex business data structures. The intent is to describe the data in a format that is not dependent upon how the data is stored. Whether the information is stored in a relational database, an XML document, or other types of structures should not make any difference to the programmer.

In order to read and write different data formats, the programmer uses a Data Access Service (DAS) component that supports the specific format being processed. For example, the programmer would use a relational database DAS to access data in a relational database. To access an XML file, the programmer would use an XML DAS. In OTTER, a DAS is used to access the individuals in the business ontologies.

Standards for Programmatic Data Graph Manipulation

In the SDO specification, the API for Java is shown in "Figure 11: Service Data Object Structure."

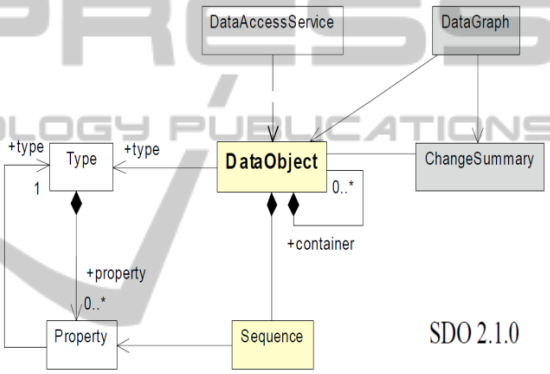

Figure 11: Service Data Object Structure.

This diagram from the specification shows how a Data Object is defined by properties and types. It also shows how a Data Object can be a container for multiple Data Objects.

By definition, an SDO is a data graph. Since all structures in OWL are data graphs, the use of SDO to view and change the content of an OWL ontology can be accomplished in a direct manner.

SCO Serialization

The metadata and content of an SDO is serialized in OTTER by using the JSON (Javascript Object Notation) standard. This provides the serialization needed to make HTTP service requests and to return an SDO to a browser.

Ontology Data Access Object

The prototype includes an OWL Data Access Service (DAS) with the following capabilities:

- **Get_DataGraph:** Creates an SDO data graph from a class expression and populates it with the

selected individuals.

- **Create_DataGraph:** Creates an SDO data graph from a class expression without any individuals.
- **Load_JSON:** Populate the individuals in a selected data graph from the JSON content.
- **Get_JSON:** Returns a JSON string of the data within a specified data graph.

These capabilities are accomplished by using Protégé APIs to describe the metadata for each data object. This is accomplished using an OWL class expression and following a stepwise process.

1. For each class in the expression, identify both the data properties and the object properties for that class. These properties must also be included in the expression.
2. The root classes of the data graph are identified using the reasoner.

This process produces a metadata tree structure of the data graph. Using this metadata, the individuals from the ontologies or from a JSON string can be loaded into the data graph.

The programmer can then access the structure using the property names as prescribed by the SDO standard.

Example of SDO from a Class Expression

In Protégé, a class expression is used to identify a subset of individuals. In OTTER, the class expression is used to identify all of the individuals within the object properties in the class expression. This results in a data graph of sets rather than a single set.

In the EA Pattern, "Person" is a class that has a data property of "hasUserID" and an object property of "inOrganization". The "Organization" class has the data property of "hasOrgUnitID." Using these classes and properties, the following can be passed to the OWL DAS as a query:

```
query=Person and (hasUserID some
string) and (inOrganization some
(OrgUnit and hasOrgUnitID some
string))
```

The process extracted the following properties for the query:

```
Data Property List
  hasOrgUnitID
  hasUserID

Object Property List
  inOrganization
```

Using the individuals returned from the query and the property lists, the OWL DAS yields the following tree structure:

```
Datagraph content:
  Individual= "person1"
    property:inOrganization
      Individual= "Org1"
        property:hasOrgUnitID
          value="Organization1"
      Individual= "Org2"
        property:hasOrgUnitID
          value="Organization2"
    property:hasUserID
      value="admin"
  Individual= "person2"
    property:inOrganization
      Individual= "Org3"
        property:hasOrgUnitID
          value="Organization3"
    property:hasUserID
      value="anonymous"
```

As the results show, "person1" is in two organizations, "Organization1" and "Organization2" and has a user ID of "admin". The individual, "person2", is in one organization, "Organization3".

Currently, the prototype will not add properties when the "inverse" operation is used in the query. The workaround is to always provide an inverse property in the ontology and use it in the query. In the example given, the individuals in the "Organization" class have an object property of "hasPeople" with an inverse of "inOrganization". The individuals of "person1" and "person2" do not actually have the property of "inOrganization" defined directly, but are rather determined by the reasoner as inferred properties. Inferred properties are handled in the same way by the OWL DAS as direct properties.

3.3.6 Security for Browser Access, Process, and Information

Security functions to restrict access of individuals to processes and information. This makes security straight-forward, since OWL ontologies are primarily about providing restrictions to defined information.

In the prototype, the security information is in a separate OWL ontology. It includes the business model security information and may exist as multiple included ontologies. These ontologies can be secured separately to provide the privacy to protect the security information.

In the prototype, the browser access is supported by HTTP basic authentication to get the user id and password. At the time of logon, the OWL DAS is used to retrieve the individual in the ontology with the user id and password provided. If they are found, access is approved.

In the EA Pattern, security access is centered upon the "Organization" class. The OntoGraf in

”Figure 12: Security Classes.” shows the property links.

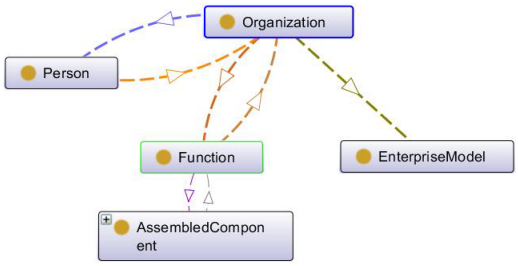

Figure 12: Security Classes.

To provide security access for a process, a “Person” can only access an “Assembled Component” when it is used by a “Function” provided by an “Organization” that has people that includes this “Person”.

Information access is provided by defining Access Control Lists (ACL). This is accomplished by using the property “hasSecurityAccess” within the “EnterpriseModel”. The “hasSecurityAccess” property has four sub-properties: “canCreate”, “canDelete”, “canRead”, and “canUpdate”. Each of these sub-properties is defined as a list of ontologies.

When a query is made by the OWL DAS, only the authorized read ontologies are included as the ACL.

Following the standards for processing an SDO, when a changed or new data graph is applied back to the ontology, the “canCreate”, “canDelete”, and “canUpdate” ontology lists will be used as the ACL.

3.3.7 Service Component Architecture

The Service Component Architecture specification defines the APIs for accessing and constructing service components. The SCA specification was developed in coordination with experts from BEA Systems, IONA Corporation, IBM Corporation, SAP AG, Sun, and Tibco Software.

In SCA, components are defined in the specification as shown “Figure 13: SCA Component.” Each component has properties, services, and references to other services.

Components can also be assembled from other components. These components can communicate within the same technical domain. This is shown in the diagram from the specification in “Figure 14: SCA Domains and Assembly.”

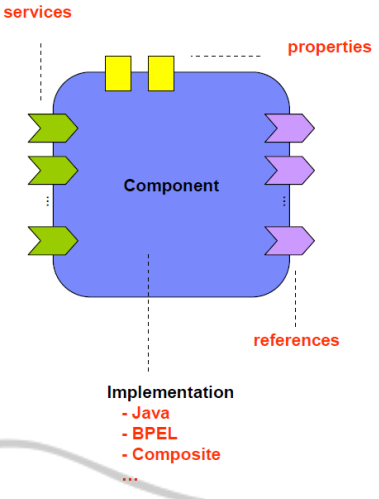

Figure 13: SCA Component.

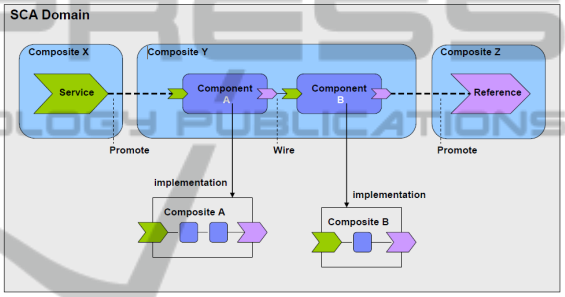

Figure 14: SCA Domains and Assembly.

The OTTER implementation of SCA supports the loading of java components and initializing their service interfaces according to their ontology definition. Initialization includes creating the data graphs used for the request and response of each service.

When a person logs on to OTTER, a session is created that maintains the person’s authorized service and ontology access.

An HTTP “get” to a service will respond with a JSON containing the request metadata. This information can be used to construct a request to the service. The HTTP “post” is used to send the request and receive the response.

The response to a “post” returns both the metadata of the data graph response and the data within the data graph.

3.4 Work Outstanding

The OTTER project is an active project with continuing work as listed below:

- **Patterns**
  - Include patterns to demonstrate principle

patterns.
    - Convert some aspects of the SmartGrid standards model to show the application to the utilities industry.
- **Prototype Enhancements**
    - Support all OWL data values in SDO. (The current support includes string, integer, and double.)
    - Extend the SDO to write changes back to the ontology.
    - Add browser-based model visualizations utilizing 2d and 3d graphics.
- **Coordination**
    - Publish the prototype for review.
    - Discussion with existing Enterprise Architecture organizations to establish a process for pattern certification.

4 CONCLUSIONS

Mapping OWL class expressions to SDO is possible. The OTTER prototype implementation proves that the concept is sound. The metadata for an SDO can be defined from an OWL class expression. Individuals can then be loaded into the SDO data graph structure directly from the ontology for data access or from JSON for service access.

Using class expressions to define data graphs can replace the use of XML for service definition and SQL for data access. This has been demonstrated in the prototype through the implementation of a Data Access Service (DAS) for ontologies in Protégé. DAS is used in SCA to create and access SDO data graphs.

The next step of writing individuals and their properties back into Protégé will provide the proof that OWL and SDO are compatible.

The value of the EA Pattern as a base ontology proved to be a requirement for implementing access to the ontologies. The value of defining the components of SCA using OWL provided a simplified interface for services.

The prototype is intended for concept demonstration only. If all aspects of the prototype are successful, another project will be initiated to provide an industrial strength implementation. This implementation will require a real-time reasoner that can evaluate each change made to the ontology for accuracy. The reasoner will update only the existing inferred properties that have changed.

With this mapping of OWL to SCA, the Enterprise Architecture pattern can be the foundation for service component development and execution. This will result in reduced expenses, less project time, and fewer errors. This could result in a paradigm shift in the quality of information systems.

REFERENCES

Adams, Matthew et al., 2006. Service Data Objects for Java Specification. *http://www.osoa.org/display/Main/Service+Data+Objects+Specifications,* Version 2.1.0.

Allemang, Dean et al., 2005. FEA Reference Model (FEA RMO) *GSA OSERA Deliverable Version 1.0*, Last Update: January 2005.

ARTS Data Model. *http://www.nrf-arts.org/content/data-model*, n.d., National Retail Federation.

Baclawski, Ken et al., 2010. Finding a Good Ontology: The Open Ontology Repository Initiative, Semantic Technology Conference, June 24, 2010.

Beisiegel, Michael et al., 2007. SCA Service Component Architecture Assembly Model. *http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications,* SCA Version 1.00.

Chandrasekaran, B. et al., Ontology of Tasks and Methods. *http://www.cis.ohio-state.edu/lair & http://www.swi.psy.uva.nl/usr/richard/home.html*.

Guizzardi, Giancarlo et al., 2006. Grounding Software Domain Ontologies in the Unified Foundational Ontology (UFO): The case of the ODE Software Process Ontology. *Federal University of Espirito Santo (UFES), Vitoria, Brazil.*

Kleb, Joachim et al., A Protégé 4 Backend for Native OWL Persistence. *Fraunhofer IITB & FZI Research Center for Information Technologies at the University of Karlsruhe*.

Knublauch, Holger, 2010. Protégé OWL Programmers Guide. *http://protegewiki.stanford.edu/wiki/Protégé OWL_API_Programmers_Guide*, Last Update: June 30, 2010.

Knublauch, Holger et al., Weaving the Biomedical Semantic Web with the Protégé OWL Plugin. *http://protege.stanford.edu*.

Krotzsch,Markus et al., 2009. Semantic Web Modeling Languages Lecture III: More OWL. *ESSLLI 2009 Bordeaux*.

MacGregor, William et al., 2006. Information Security: An Ontology of Identity Credentials Part1: Background and Formulation. *NIST National Institute of Standards and Technology, NIST Special Publication 800-103 Draft*, October 2006.

Mrohs, Bernd, J.Sc. (CS) et al., OWL-SF – A Distributed Semantic Service Framework, *Fraunhofer Institute for Open Communication Systems.*

Semy, Salim et al., 2004. Toward the Use of an Upper Ontology for U.S. Government and U.S. Military Domains: An Evaluation. *MITRE TECHNICAL REPORT*, September 2004.

The Open Group, 2010. Technical Standard, Service-Oriented Architecture Ontology. *Published by The Open Group*, October 2010.

Tinsley, Thomas A.,2009. *Enterprise Architects Masters of the Unseen City,* Tinsley Innovations.