A Framework for Dynamic Semantic Web Services Management

Randy Howard and Larry Kerschberg

choward@gmu.edu, kersch@gmu.edu
E-Center for E-Business, http://eceb.gmu.edu/
Department of Information and Software Engineering
MSN4A4, George Mason University, 4400 University Drive,
Fairfax, VA, 22030-4444

Abstract. The concept of Web services as a means of dynamically discovering, negotiating, composing, executing and managing services to materialize enterprise-scale workflow is an active research topic. However its realization has thus far been elusive. Existing approaches involve many disparate concepts, frameworks and technologies. What is needed is a comprehensive and overarching framework that handles the processing and workflow requirements of Virtual Enterprises, maps them to a collection of service-oriented tasks, dynamically configures these tasks from available services, and manages the choreography and execution of these services. The goal is to add semantics to Web services to endow them with capabilities currently lacking in the literature, but necessary for their successful deployment in future systems.

This paper introduces such a framework, called the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework, that addresses in an integrated end-to-end manner, the life-cycle of activities involved in preparing, publishing, requesting, discovering, selecting, configuring, deploying, and delivering Semantic Web Services. In particular, the following issues are addressed: 1) semantic specification of services capabilities including quality of service, trust, and security; 2) transaction control and workflow management; and 3) resource management, interoperation and evolution of the Virtual Enterprise.

Two models and their associated languages are introduced to specify the features for these enhanced services: 1) the KDSWS Meta-Model, and 2) the KDSWS Process Language. Both are based on the Knowledge/Data Model. This integrated and comprehensive approach provides a unified end-to-end approach to enable dynamically-composable, semantically-rich, service-oriented systems of Web services.

Key Words: Semantic Web Services, Virtual Enterprise, Ontology, Agent-Based Systems

1 Introduction

The relatively new concept of *Web services* [15] is important to both e-Business and e-Government in that it changes the traditional *client-server* model of accessing web-based information into a *peer-to-peer* model in which computers may exchange information over the Internet. Web services standards provide XML-based protocols to find publicly-registered services, to understand their purpose and operation, to negotiate and agree upon usage charges and Quality-of-Service commitments, and to invoke the services within the context of Internet-based workflow coordination of these services.

Web services provide a service-oriented approach to system specification, enable the componentization, wrapping and reuse of traditional applications, thereby allowing them to participate as an integrated component to an e-Business activity [9]. Web services are also maturing from the original generation of

static brochure-style information display to more dynamic and personalized user interactions [56]. They offer benefits similar to traditional outsourcing services in that software functionality is "leased" versus being "bought", making it easier to stay current. Standards to support the interoperability of the services include Universal Description, Discovery and Integration (UDDI) [55], Simple Object Access Protocol (SOAP) [24], and Web Services Description Language (WSDL) [15].

An interesting question is: "How do Web services relate to the Semantic Web?" The Semantic Web is "data integration across application, organizational boundaries", and Web services are "program integration across application and organizational boundaries" [6]. Tim Berners-Lee stated that Web services are an actualization of the Semantic Web vision because the semantic markup of Web services makes them computer-interpretable, use-apparent, and agent-ready [14, 38].

This paper introduces a framework, called the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework, that addresses in an integrated end-to-end manner, the life-cycle of activities involved in preparing, publishing, requesting, discovering, selecting, configuring, deploying, and delivering Semantic Web Services. In particular, the following issues are addressed: 1) semantic specification of services capabilities including quality of service, trust, and security; 2) transaction control and workflow management; and 3) resource management, interoperation and evolution of the Virtual Enterprise (VE).

"Semantic Web Services" (SWS) [13] is the term that describes our research approach. We view "Web services" as services that use little or no semantic markup, and have little of the enhanced capability described in this paper. Semantic Web technology, on the other hand adds semantic and process oriented information, together with heuristics and constraints that can be used to coordinate the activities of the VE. SWS allows the Web information to be structured not only for human consumption, but for machine processing as well [7]. This base of Semantic Web technologies involves Resource Description Framework (RDF) [37], DARPA Agent Markup Language (DAML) [27], Ontology Inference Layer (OIL) [21, 46], DAML+OIL [17], and Web Ontology Language (OWL) [5].

1.1 Problem Statement

Enterprise Application Integration (EAI) across a VE is difficult due to: a multitude of semantics and protocols; the need to propagate and synchronize rules, constraints and elements; and the temporal nature of a VE [49]. The handling of these issues needs to be streamlined in order to deal with the heterogeneous and constantly changing environments within an enterprise. Many integration solutions are rigid because they are developed for ad-hoc and individual interfaces. Also, many vendor solutions are proprietary, and tend to make changes for a broad base of customers versus the unique needs of a specific customer.

In order for Web services to operate more effectively within a VE, teaming arrangements need to be structured to maintain trust and contractual relations on which to conduct business. A challenge for companies wishing to create Inter-Enterprise Interoperation (IEI) solutions via Web services is to dynamically wrap and compose service-oriented functionality for the VE. Present solutions require the hand-crafting of Web services and their interfaces, although recent literature [36] has begun to address this issue.

Articles of Federation and Service Level Agreements should be incorporated into the automation process in order to manage the execution cycle of the Web services. In order for Web services to address business needs, they need to facilitate such issues as pricing, resource management, quality of service, scalability and delivery schedule.

The ultimate vision of SWS is the dynamic discovery, configuration, and deployment of a VE from services distributed over heterogeneous systems, thereby creating a VE from collections of services. At present, however, this vision is far from reality, in which companies configure services by hand, using the telephone to coordinate interfaces, etc. In order to enable the vision, this paper presents a unified framework to address the specification of service requirements, map those requirements to composable services, and coordinate the execution of services according to enterprise workflow requirements.

1.2 Issues with Existing Approaches

Semantic Web Services attempt to address the shortcomings of Web services [38]. However, they do not address fully the issues related to the VE. For example, OWL-S, formerly DAML-S [47], and a leading specification for the automation of Web services) considers the service and the process aspects as the primary elements, but does not show how to specify federation, agent and data store aspects. Some of the core developers of OWL-S have applied it to work in conjunction with Artificial Intelligence (AI) languages like SHOP2 and ConGolog [38, 64]. These efforts are using an action metaphor and tend to focus on the technical mechanics versus addressing many of the business issues such as pricing and resource management.

Approaches such as Business Process Execution Language for Web Services (BPEL4WS) [28], Web Services Choreography Interface (WSCI) [2]and World Wide Web Consortium (W3C) WS-Choreography group [3] do not address the full life-cycle of Web services. To achieve an end-to-end solution involves linking disparate protocols and technologies, despite the fact that they are all still interoperable on the XML foundation. The nuances of these various technologies still need to be mediated to some level to be truly interoperable.

Web services technologies are currently facing the same types of problems in implementing enterprise integration that 'traditional' technologies have already addressed in large scale deployments. The major problems [13] are:

- **Semantic Unification**. Data exchanged between application systems or trading partners (endpoints) are defined based on different schemas. When data are exchanged in the form of messages, a data mediation problem arises that requires resolution. A minor and related issue is that different application systems or trading partners use different forms of syntax, in addition to different schemas for messages. Even if endpoints describe their data in the form of ontologies, the semantic unification problem remains to be solved.
- **Service Behavior**. Different endpoints expect specific messages in a specific order and with specific sequencing. Communicating endpoints have to guarantee and to enforce the exchange behavior as agreed to establish interoperability.
- **Endpoint Discovery**. The manual establishment of trading relationships is considered error prone, slow and inflexible. Discovery mechanisms are put in place (e.g. UDDI) that promise to make the automatic discovery process easier and more reliable.
- **Message Security and Trust Relationships**. Communicating endpoints require assurance of message confidentiality and non-repudiation. Various security schemes (e.g. SAML [44], WS-Security [29]) are being developed that attempt to address these requirements. Furthermore, endpoints need to establish sufficient trust to engage in a trading relationship.
- **Process Management**. Supply-chain processes are very complex and highly dynamic. Attempts have been made to enable dynamic supply-chain reconfiguration with agent technology and dynamic workflow technology. A large body of work (e.g. ARIS [51], BPEL4WS, Business Process Modeling Language (BPML) [1]) exists that has not yet found its way into industry and real applications.
- **Integration Standards**. A plethora of standards exists in the area of Enterprise Integration. All of these have to be dealt with to some extent by the various enterprises.
- **Legacy Application Connectivity**. Most data that are communicated are managed by existing application systems that are not necessarily designed to be integrated. Adapter technology exists that allows connecting easily to application systems.

1.3 Paper Organization

Section 1 has introduced the problem and has discussed some of the problems associated with existing approaches. Section 2 deals with Web services in virtual enterprises and shows how Grid technologies encounter and solve similar problems. Service-Oriented Architectures and their relationship to Web services are discussed. Next, the basic functions and management needs of Web services within a VE are presented.

The Knowledge-based Dynamic Semantic Web Services Framework is presented in Section 3 with a detailed discussion of its components. The framework is decomposed into three hierarchical layers

consisting of Virtual Enterprise process specification, design specification and execution services. The mappings from one layer to another are also explained. A meta-model for the KDSWS classes, properties, relationships and constraints is developed using the Knowledge/Data Model constructs. The KDSWS Process Language is presented and discussed as a specification language for enhanced SWS.

Section 4 presents a case study in the form of a scenario that illustrates the major aspects of the KDSWS framework and the KDSWS Process Language. It highlights the enhanced semantic services supported by the KDSWS. Section 5 presents our conclusions and suggests areas for future research.

2 Web Services in a Virtual Enterprise

2.1 Virtual Enterprise Issues

> *A virtual organization, or enterprise, is one whose members are geographically apart, usually working by computer e-mail and groupware while appearing to others to be a single, unified organization with a real physical location [61].*
>
> *The virtual enterprise is a temporary relationship with two or more participants which is formed, operated, and dissolved to accomplish specific short term goals. It differs from existing inter-organizational models by the degree of shared accountability and responsibility of the participants and the structure by which companies contribute their competencies [49].*

The next steps in the evolution of EAI and IEI are towards that of a VE. For Web services to perform a significant role in a VE environment, many issues need to be addressed to accommodate the disjointed, distributed, and temporal nature of the VE. A VE is a dynamic collection of individuals, institutions, and resources [22].

As stated by [49], "The activities of the virtual enterprise cycle are accomplished by processes that are owned and operated by individual members of the VE or shared processes that are 'owned' jointly by the enterprise as a corporate entity. Whether the processes are individually or jointly owned is largely predicated on the objectives of the enterprise and how they are to be accomplished."

Operating web-service-centric systems effectively within a VE offers unique challenges because factions, herein referred to as partners, often compete and conflict with each other. Additionally, there oftentimes is no single dominant "partner-in-charge", nor is there an overarching policy to guide the process past roadblocks and exceptions.

2.2 Grid Technologies

> *Grid computing appears to be a promising trend for three reasons: (1) its ability to make more cost-effective use of a given amount of computer resources, (2) as a way to solve problems that can't be approached without an enormous amount of computing power, and (3) because it suggests that the resources of many computers can be cooperatively and perhaps synergistically harnessed and managed as a collaboration toward a common objective. In some grid computing systems, the computers may collaborate rather than being directed by one managing computer. One likely area for the use of grid computing will be pervasive computing applications - those in which computers pervade our environment without our necessary awareness [52].*

The problem that underlies the Grid concept is *coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations*. The sharing to be dealt with is primarily direct access to computers, software, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering in additional to file sharing. This sharing is, necessarily, highly controlled, with resource providers and consumers defining

what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules form what we call a *virtual organization* [22].

2.3 Service-Oriented Architecture

> *With the introduction of Web Services over the last year or so, there has been a renewed interest in service-oriented architecture (SOA). An SOA is an architecture that has special properties. It is an architecture made up of components and interconnections that stress interoperability and location transparency. The term service has been used for more than two decades. For example, leading transaction monitoring software has used the term 'service' in the early 1990s. Many client-server development efforts in the 90s used the term 'service' to indicate the ability to make a remote method call. Web Services has given the term service more prominence in the last few months. Services and service-oriented architectures are really about designing and building systems using heterogeneous network addressable software components [54].*

The architectural elements of a SOA provide functionality that is incorporated into the design although the actual implementation across the enterprise may vary. As shown in Figure 1 [12], Application Implementation Layers, and technology layers are applied to application architecture to provide more coarse-grained implementations, or the "application edge", to expose an external interface of a system. The internal reuse and composition using traditional component design is accomplished behind the interface. This design's approach is to define structures to pass between the layers to empower back-end processes to better fulfill the request.

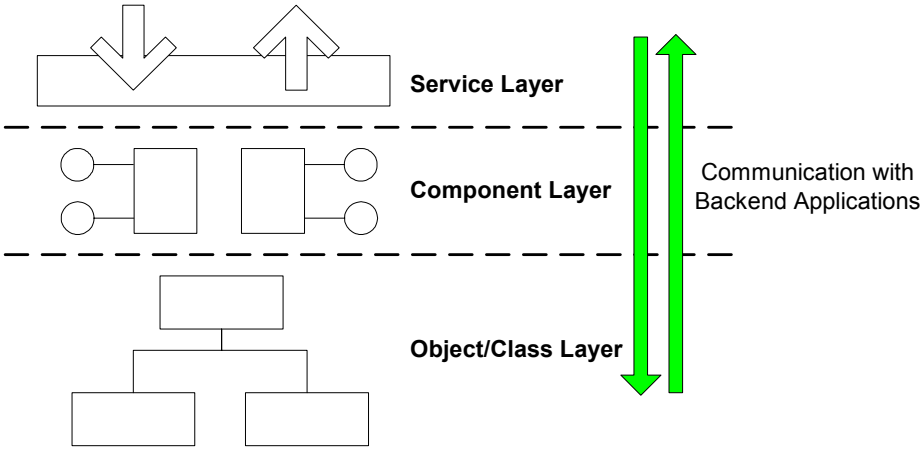

Figure 1. Application Implementation Layers

2.4 Web Services Functions

> *To realize our vision of Semantic Web Services, we are creating semantic markup of Web services that makes them machine-understandable and use-apparent. We are also developing agent technology that exploits this semantic markup to support automated Web service composition and interoperability. Driving the development of our markup and agent technology are the automation tasks that semantic markup of Web services will enable—in particular, service discovery, execution, and composition and interoperation. Automatic Web service discovery involves automatically locating Web services that provide a particular service and that adhere to requested properties [38].*

Users make requests to systems that require many layers of behind-the-scenes processing to service users' needs. Web services often provide functionality that is very similar to other Web services, and selecting which web service will best fulfill a given request is very difficult. Effectively differentiating between

similar Web services requires additional information about the request, requestor, organization, scenario, and suppliers that are involved in providing the solution.

The Web services architecture, or triangle, is shown in Figure 2 [53]. It consists of three main elements: Service Provider, Service Registry, and Service Consumer. The Provider creates the web service and publishes it to the Registry. The Consumer inquires the Registry to retrieve information about using the web service [14].

Web services provide a common and standard interface to all consumers; thus, the problems with integrating heterogeneous applications are greatly minimized if not eliminated. Web services also provide "implementation independence" which means that it provides encapsulation to the actual code, the details of components, protocols, and architectures, etc. Consumers only need worry about the inputs and outputs of the service.

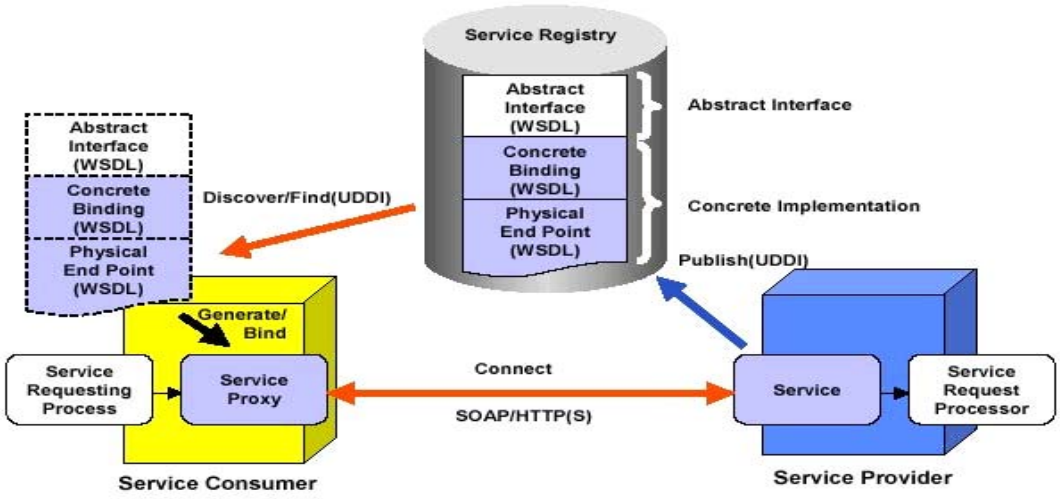

Figure 2. Web Services Architecture

Interoperability is the central issue for a VE, which means common protocols are needed for users and resources to negotiate, establish, manage, and exploit sharing relationships without doing harm (e.g., compromising security) [22]. Web services are loosely-coupled because they work with highly standardized interfaces such as Extensible Markup Language (XML), Universal Description and Discovery and Integration (UDDI), Simple Object Access Protocol (SOAP), and Web Services Description Language (WSDL).

XML is a simple, very flexible text format derived from Standard Generalized Markup Language (SGML) (ISO 8879) [57], and is designed to improve the functionality of the Web by providing more flexible and adaptable information tagging and description. One example of XML-based process language is ebXML which allows companies to have a standard method to exchange business messages, conduct trading relationships, communicate data in common agreed-upon terms, as well as define and register business processes [19]."

SOAP is an XML-based object invocation protocol for sending XML messages between endpoints, and may be used for remote procedure calls (RPC) or plain document transfer. It was originally developed for distributed applications to communicate over HTTP and through corporate firewalls. SOAP defines the use of XML and HTTP to access services, objects and servers in a platform-independent manner.

WSDL, the XML equivalent of a resume, and describes what a web service can do, where it resides, and how to invoke it. It is the equivalent of interface definition language (IDL) and type libraries found in CORBA or COM. WSDL defines a service's abstract description in terms of messages exchanged in a

service interaction. WSDL consists of three main components: the vocabulary, the message, and the interaction.

UDDI is a registry for connecting producers and consumers of Web services. A producer can use the UDDI publish API to register information about a web service, and a consumer can use the UDDI inquire API to locate one or more Web services that satisfy a particular set of criteria [58]. UDDI Versions 1 and 2 use identifiers and categories to describe businesses and the services they provide. Version 3 will be enhanced with multi-registry topologies, increased security features, improved WSDL support, a new subscription API and core information model advances. UDDI Version 3 introduces the notions of *root* and *affiliate* registries as part of its guidance on inter-registry associations [55].

To further realize visions of SWS, new AI-inspired content markup languages built on the Resource Description Framework (RDF) are being developed. RDF is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web. It emphasizes facilities to enable automated processing of Web resources. RDF can be used in a variety of application areas such as [37]:

- Resource discovery to provide better search engine capabilities,
- Cataloging for describing the content and content relationships available at a particular Web site, page, or digital library, by intelligent software agents to facilitate knowledge sharing and exchange
- Content rating,
- Describing collections of pages that represent a single logical "document",
- Describing intellectual property rights of Web pages,
- Expressing the privacy preferences of a user as well as the privacy policies of a Web site, and
- Creating digital signatures to build the "Web of Trust" for electronic commerce, collaboration, and other applications.

Some of the languages mentioned earlier are the Ontology Inference Layer (OIL), DAML+OIL, and DAML-S (the last two are members of the DARPA Agent Markup Language (DAML) family of languages) are being developed. These languages have a well-defined semantics and enable the markup and manipulation of complex taxonomic and logical relations between entities on the Web. A fundamental component of the Semantic Web will be the markup of Web services to make them computer-interpretable, use-apparent, and agent-ready [38].

These common interfaces facilitate interoperability because they separate the messaging from the internal components that are heterogeneous and dynamically changing across the VE [31]. While these mechanisms facilitate open communication, they also provide the capability to secure message contents by providing an infrastructure to support PKI and Kerberos [60]. Specifically, WS-Security provides protection through message integrity, message confidentiality, and single message authentication. It also provides a general-purpose mechanism for associating security tokens with messages, encodes X.509 certificates and Kerberos tickets, and opaque encrypted keys, along with extensibility mechanisms. These mechanisms can be used to accommodate a wide variety of security models and encryption technologies [10].

Since the use of Web services will continue to grow, the following issues need to be addressed in order for this relatively new technology to scale:

- Web services need to transition from being selected statically, and manually, to being discovered and composed dynamically to fulfill a request.
- In conjunction with the latter point, the dynamic selection needs to differentiate, based on an enhanced feature specification, among Web services offering similar functionality so as to determine which service best satisfies the request.

2.5 Web Services Management

The management levels within a VE are strategic (infrastructure), asset (resource management), and value-chain (processes) [42]. The issues that exist with Web services operating across a VE are set within the context of these layers:

- Strategic
  - Roles and responsibilities of the partners
  - Level of participation
  - Governing and enforcement policies
  - Resources made available within the organization and outside to customers
- Asset
  - Distribution and management of resources
  - Ownership and stewardship of resources
  - Knowledge repository architecture (central vs. distributed)
- Value-Chain
  - Process control
  - Handling anomalies
  - Workflow management
  - Coordination of constraints

Additionally, workflows need to sense the environment capture scenarios by using multi-level and specialized agents. Ontologies and taxonomies need to be capture and convey personalization to the workflow model. Workflows also need to work within a structure of enterprise and local constraints. Systems need to perceive the condition of the environment and act accordingly. Systems need to be more adaptive by providing a structure and introspection of functions and methods. Systems need to setup a teaching, as well as a learning, framework and address the latency of learning as well. Metrics need to be identified and developed to measure effectiveness for feedback mechanisms.

W3C's Management Model shown in Figure 3 focuses on those aspects of the Web services architecture that relate to the management of Web services; in particular with an emphasis on using the infrastructure offered by Web services to manage the resources needed to deliver that infrastructure [8]. This research conforms to the concepts presented in this model.

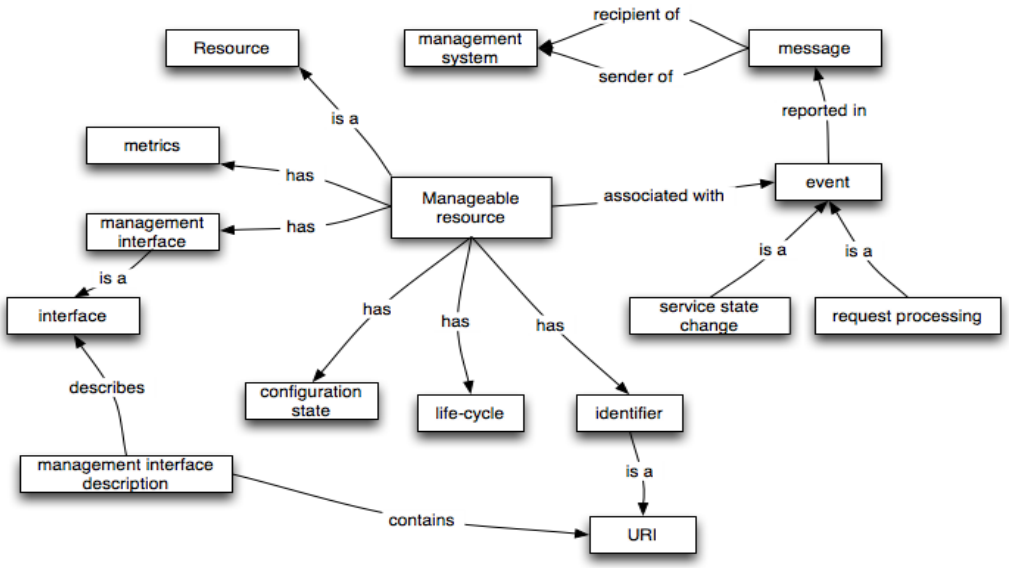

Figure 3. W3C's Web Services Management Model

The protocol marked for the mainline use for using Web services to manage distributed resources is the Web Services Distributed Management (WSDM) [43]. This protocol is still in Technical Committee status,

and has the additional purpose of developing a model of a web service as a manageable resource. A newer protocol, WS-Federation, focuses on security and trust aspects [4, 23], whereas the concepts of federation are more encompassing in this research than those found in WS-Federation.

3 Knowledge-based Dynamic Semantic Web Services Framework

This research proposes the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework to deliver a comprehensive and integrated end-to-end solution to dynamically prepare, publish, request, discover, select, configure, deploy and deliver SWS. The framework offers the following key contributions:

1. The term 'dynamic' refers to the automation of the Web Service life-cycle, and the term 'semantic' denotes the knowledge-based semantic specification of the relevant features and functions provided by the Web Service.
2. The framework is comprehensive in that it addresses the backend specification for federating enterprise resources, agents and knowledge storesrepositories.
3. It is integrated because the data and process specifications are created using the same foundation - the Knowledge/Data Model and Language [41, 48].
4. The framework is knowledge-based because it uses heuristics to associate the knowledge to objects and services, and captures usage context as well.
5. The framework is an end-to-end solution in that it addresses the entire Web services life-cycle, from creation to delivery. The retirement of services is weaved into the feedback cycle.

The framework provides components that facilitate management, coordination, and interoperability for the loosely-coupled, distributed, and diverse Web services. The framework presumes that the VE has more visibility into the partners' environments than with ordinary Web services, thus allowing the additional knowledge to optimize the interactions through Web services. Interoperability is facilitated by the integrated design space and mediation structures.

As shown in Figure 4, the framework is comprised of the KDSWS Processes, KDSWS Specification, and KDSWS Functional Architecture. The KDSWS Processes include the tasks in the life-cycle of Web services and the functional threads that run through the tasks to address the issues of the particular perspectives.

The KDSWS Specification captures the design specification in a manner that is usable by computers and humans. The specification is represented using the Knowledge/ Data Model and Language (KDM/KDL) and the KDSWS Process Model and Language (KDSPM/KDSPL), which is derived from meta-model and methodology, respectively. The specification also defines mappings to the elements in the KDSWS Functional Architecture, including SWS.

The KDSWS Functional Architecture is comprised of the Functional Federation Architecture (FFA), the Functional Agent Services Architecture (FASA), Functional Knowledge Architecture (FKA) and Web Services Protocols. The functional emphasis on these components originates from the need to embed the purpose, behavior and relations within the specification. The FFA coordinates the roles and responsibilities of the partners within an enterprise to assist with joint ownership issues and to compensate for the potential "no partner in charge" issue within a VE. The federation also facilitates use of Grid technologies and architectures. The FASA drives the framework, while the FKA maintains the knowledge for the framework.

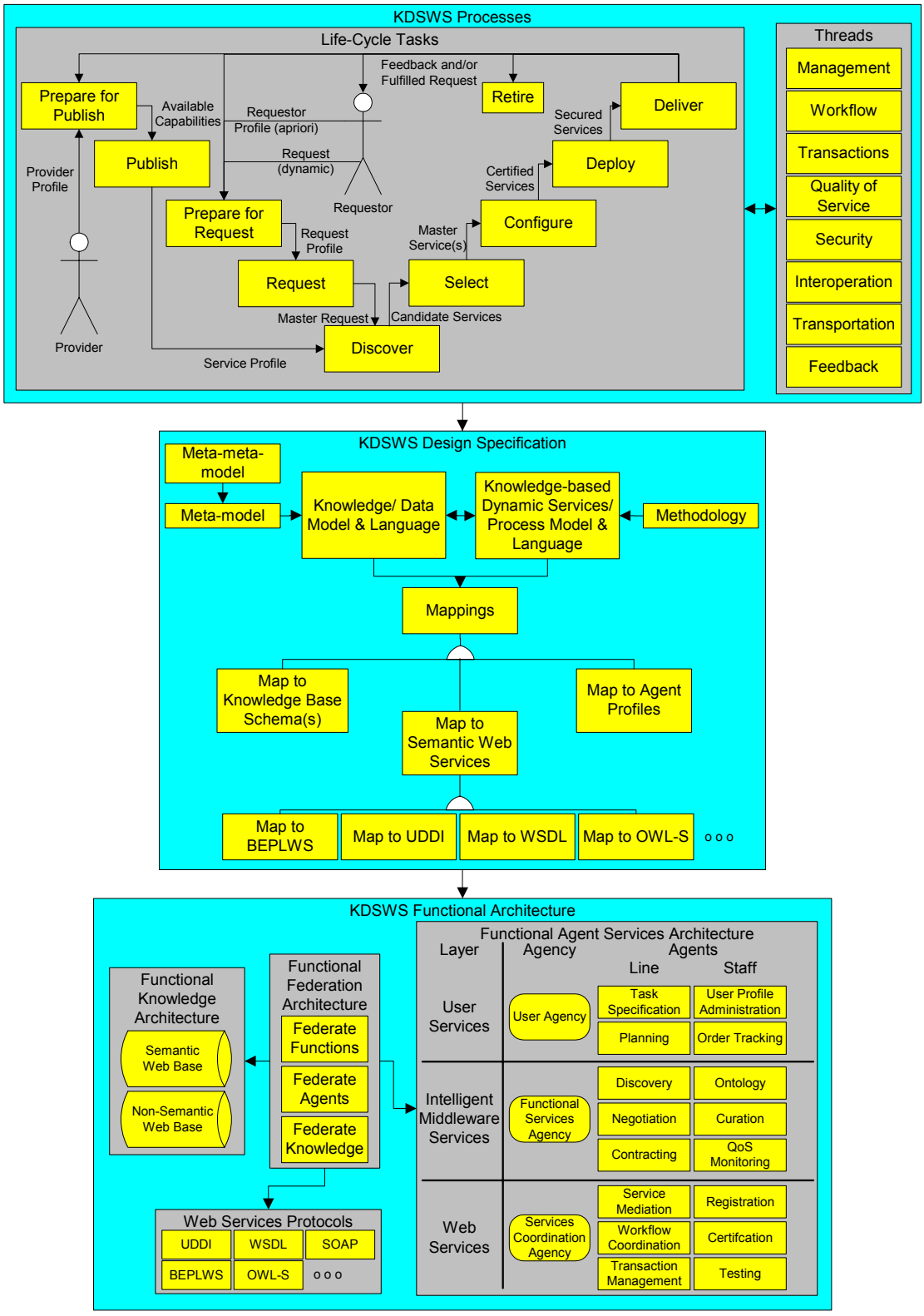

Figure 4. KDSWS Framework

Ontologies are a key enabler for the Semantic Web [20] technologies. The structures and components are designed in such a fashion as to map into an ontology or SWS framework. The framework also takes advantage of ontologies stored in such repositories as XML Databases.

3.1 KDSWS Processes

The KDSWS Processes layer is comprised of Tasks and Threads that are used to deliver functionality through Web services, and they (the Tasks and Threads) set the context of the processes. Some authors present Web services functionality as a stack [8, 26, 62], but this approach separates the activities within the life-cycle of Web services from the functional perspective.

Tasks are well-delineated steps to deliver functionality via Web services, and we propose the following collection of tasks for the Web services life-cycle: Prepare for Publish, Publish, Prepare for Request, Request, Discover, Select, Configure, Deploy, Deliver and Retire. The profile of the provider, and its services, is used in the Prepare for Publish (underlined to reflect an element in the diagram) task to establish the capabilities available to create service profiles for individual services in the Publish task. The Prepare for Request task uses the requestor's profile and the request to produce a request profile that the Request task uses to produce a "Master Request". The Master Request is used by the Discover task to create a list of candidate services for the Select task to choose from the selected services, some of which are "Master Services". The Master Services are either the complex services or the services involved in workflow steps. The Configure task uses the selected services to create a composed and certified plan for the Deploy task to secure the resources. The Deliver task executes the plan and provides feedback in the form of a fulfilled request or anomaly in the execution. The Retire task uses the feedback to determine if the service is viable or not.

Threads are layers of functionality that the tasks use to deliver the services; they address issues related to management, workflow, transactions, quality of service, security, interoperation and feedback. The management layer directs the activities of the other threads. Workflow involves the management of the steps to achieve the goals to the request, while transactions deal with the control of the lower-level actions to maintain the ACID (Atomicity, Consistency, Isolation and Durability) properties. Quality-of-Service (QoS) ensures the expected performance levels of availability, quality, security, response time and throughput [39] are maintained. Security provides for authentication, integrity, privacy and non-repudiation. Interoperation ensures that the service integrates with other services and protocols. Feedback involves keeping the requestor informed as to status and measuring performance results of the delivery.

3.2 KDSWS Design Specification

The KDSWS Specification has both a data-centric and a process-centric focus. The data requirements are captured in the meta-model, whereas the process requirements are captured in the methodology. Each element starts at the highest level and cascades down the level of detail to form a hierarchy of relations between the levels. The process elements integrate and reference elements in the data elements to form an integrated specification, or a hierarchical lattice to ensure the elements are consistent both vertically and horizontally. The specification has three levels of capabilities of Lite, Standard and Full that allow providers to specify the level of complexity of the features of their services and allows requestors to specify the level at which they which to participate.

The meta-model is specified via an adaptation of the KDM/KDL [41, 48], while the methodology is specified by the new (KDSPM/KDSPL), as depicted in Figure 4. The specifications correlate to the respective components of the KDSWS Functional Architecture. In addition the KDSWS specification provides mechanisms to map to Semantic Web and Web services components and protocols.[1]

[1] The specification is still under development and is scheduled to be completed by the time this article is published; however, it is mature enough to show its critical role in the DSWS Framework.

3.2.1 Meta-model

The Meta-Object-Facility (MOF) [45] establishes four levels of metadata architecture: the meta-meta-model, meta-model, model, and information. The meta-meta-model shown in Figure 5 displays the links relating the RDF/RDFS and OWL data types (because OWL uses the RDF Schema data structures) [5]. Some of the 'attributes' in the meta-model in Figure 6 are shown at an abstract level to provide context for the class. The term meta-model will herein refer to both the meta-meta-model and the meta-model together. The elements contained in the meta-model come from various sources such as the Web Services Modeling Framework [20], OWL-S [16] and Web Services Architecture Usage Scenarios [25].

The meta-model focuses on developing the 'building blocks' (i.e. constraints, preferences, profiles, capabilities, etc.) for the framework versus focusing on the higher-level and visible, finished products like a service, a profile, or request on which some approaches place their primary attention. This 'building block' approach makes the framework components composable, reusable and extensible. The lower level 'building blocks' can easily be aggregated to compose the higher level elements. By making entities like constraints and preferences a super-class, the benefits of generalization/specialization are achieved for those components. In the case of a complex service (one that calls other services), the provider is also a requestor in the same execution stream. The superclass level functions can handle the constraints or preferences that are common to requestor/provider, but then the specialized functions can be used as the service transitions roles. By creating the building blocks at the meta-model level, it enables new components to be incorporated much more easily.

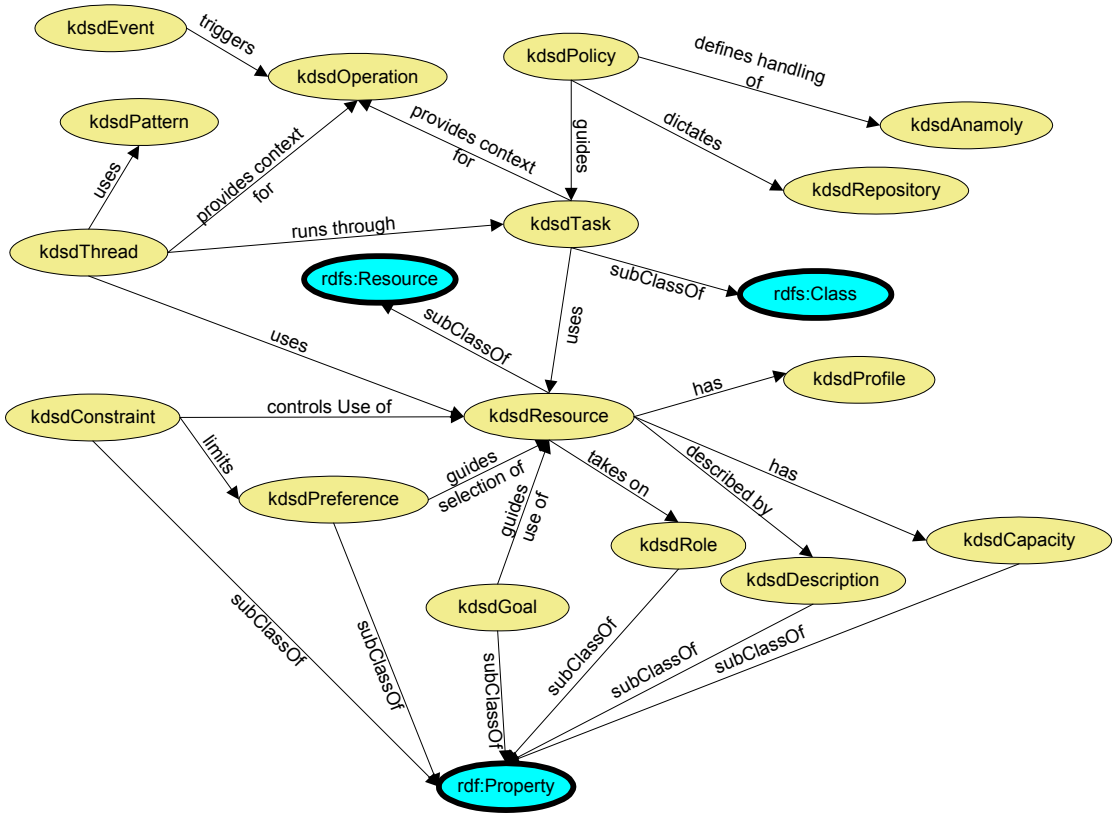

Figure 5. Meta-Meta-Model

The following discussion explains the major elements of the meta-model. kdsdPolicy objects capture knowledge about such items as Articles of Federation (i.e. joining and disbanding), service level agreements, security policies and payment/fee policies. kdsdRepository places data stores into such categories as kdsdKnowledgeSource (i.e. XML Database), kdsdPackage (i.e. Fulfillment Package), kdsdRegistry (i.e. UDDI, WSDL). kdsdAnomaly is the superclass for kdsdException,

kdsdContraintViolation and kdsdError. kdsdEvent defines occurrences that should trigger some processing to commence. Two possible subclasses of kdsdEvent are kdsdNotification and kdsdSensor.

kdsdTask is the superclass for kdsdDeliver, kdsdDeploy, kdsdDiscover, kdsdPrepare, kdsdPublish, kdsdRequest, kdsdSelect. kdsdPrepare is combined with both the kdsdPublish and kdsdRequest tasks because both tasks have a preparation task. kdsdOperation are atomic functions that carry out the purpose of a step, and can specify a process object, KDL method, or manual intervention that is needed.

kdsdThread and kdsdPattern are both superclasses for the thread objects because threads use well-established patterns. kdsdManagement is organized by the three management levels associated with Virtual Organizations: Strategic, Asset, or Value-Chain. kdsdWorkflow and kdsdTransaction coordinate and control the steps and operations of the process. kdsdQoS (Quality of Service) conveys the time, accuracy, pricing levels and throughput that are deemed to be acceptable by all parties. kdsdSecurity captures needs for security level, authentication, authorization and non-repudiation. kdsdInteroperation is a pivotal class to handle such properties as kdsdInteroperatonLevel (i.e. Data, Presentation or Process) or kdsdLanguage. kdsdInteroperation also handles the mediation needs under dswsMediation. One particular attribute is the kdsdMediationType, which conveys whether the mediation is at a data structure, business logic, message exchange protocol or service invocation level. kdsdTransportation is organized by kdsdNetwork and kdsdMessage, and handle such elements as invoked Web Service proxy, ports, transport binding, retry rules and limits, and routing. kdsdFeedback captures knowledge about metrics, responses to the requestor and ratings of the services. kdsdResponse covers a possible retry, result, timeout or alternative.

kdsdDescription provides a superclass structure for descriptive properties. kdsdBlackbox properties are properties that other services should typically not be concerned with in order to use the Web Service – it considers the Web Service as a 'black box' concerning the properties. Contrasted with kdsdBlackbox properties, kdsdGraybox properties are properties that might be useful for requestors to see about the provider. Manageability is a key property that falls into this category.

kdsdCapabilities is a very diverse and pivotal class that is a superclass for such items as kdsdInvocationType (i.e. synchronous or asynchronous) or kdsdPlatform (PDA, Server, or Desktop). kdsdDocument holds the Dublin-Core Metadata Element Set (i.e. Title, Name, Description, Date, Type, etc.) [18] as well as syntax and protocol. One particular attribute of interest is the kdsdCapabilityLevel which describes whether the resource participates at a Lite, Standard or Full level. kdsdScenario depicts the situation behind the request, while kdsdSpecialization conveys the primary focus of the object such as negotiation or coordination as in the case of agents.

kdsdCapacity conveys the availability of resources in the form of CapacityType (DiscreteCapacity or ContinuousCapacity). kdsdAllocationType conveys whether an allocation consumable or reusable. kdsdProfile is an aggregation of the properties and objects within the framework. An example of this is demonstrated in the Scenario.

The primary function of kdsdGoal is to capture the purpose of an object. It also serves as an aggregation of kdsdPostCondition, kdsdPreCondition, kdsdRule because they need to be set in the context of a definitive purpose.

The major subclasses of kdsdResource are kdsdAgent, kdsdCatalog, kdsdProtocol, kdsdService, kdsdServiceChain, kdsdSupplyChain, kdsdPartner. The various protocols are captured in the kdsdProtocol. kdsdService breaks down into kdsdMode to capture whether a service offers RPC or Document messaging. kdsdMethod conveys whether the function is behind a Web Service or not (i.e. as in the case of an agent). kdsdRole are split in kdsdServiceChain and kdsdSupplyChain subclasses, where kdsdServiceChain describes elements in the technical architecture (the service itself) and the kdsdSupplyChain describes in the organization architecture.

kdsdConstraint allows the restriction of domain (inputs) and ranges (outputs) for objects, while kdsdPreference captures the manner in which providers and requestors prefer to do business.

kdsdAccuracy, kdsdDelivery, kdsdFlexibility, kdsdPayment, kdsdPrice, kdsdQuality, kdsdPrivacy are subclasses of both kdsdConstraint and kdsdPreference. kdsdPreference because their attributes and behavior can be applied to either generalization.

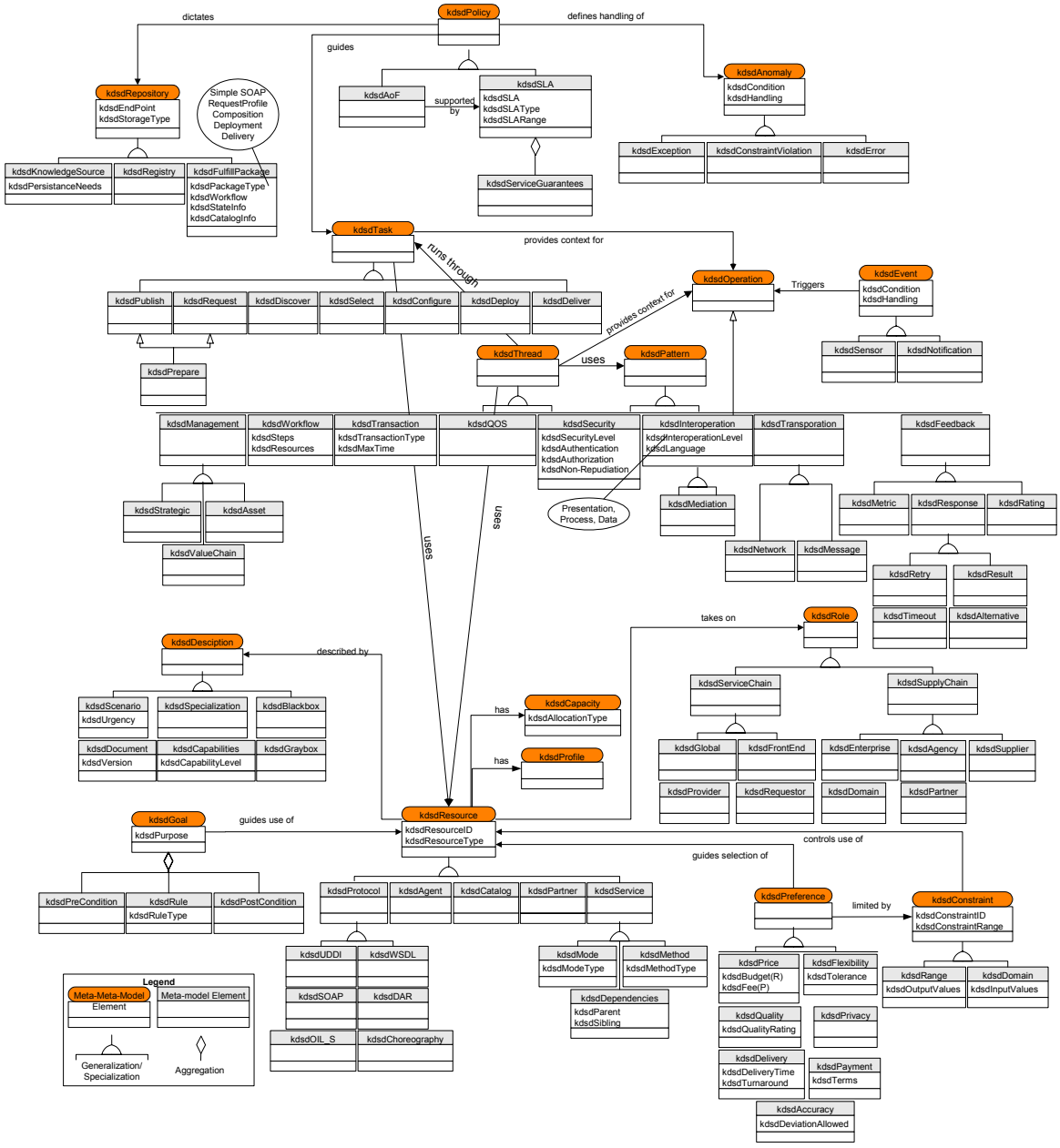

Figure 6. KDSWS Meta-Model

3.2.2 Methodology

The major processes correlate directly to the Tasks identified in the framework: Prepare for Publish, Publish, Prepare for Request, Request, Discover, Select, Configure, Deploy and Deliver. At the highest level, these tasks describe the chronological life-cycle of a Web Service; however, the detailed view reveals that iterative facets are necessary to make Web services dynamic, or automated. The methodology seemingly duplicates with the tasks in the meta-model. To explain the necessary overlap, the meta-model

elements identify the data needs for the task in the life-cycle, whereas the methodology identifies the process and uses the meta-model elements.

The Prepare for Publish process establishes the provider's knowledge base to use for the Web services automation by either brokering outside knowledge bases to be mediated directly or incorporating the knowledge into the FKA. The Publish process pulls knowledge from FKA and posts it to advertising (i.e. UDDI) and invocation (i.e. WSDL) resources.

The Prepare for Request process introduces the concept of a "Master Request" that represents the primary end-result that the user is requesting. The Master Request will likely entail simple and complex requests for Web services to be fulfilled. The Prepare for Request and the Request processes mirror those of the Publish side in purpose, except they are performed from the perspective of the request. The profiles built up from the request side need to map to those on the publish side to enable the automated selection. However, the Plan-for-Request process is much more encapsulated than its Prepare for Publish counterpart because of its need for quicker response.

The Discover process receives the request, decomposes the Master Request if it involves workflow steps, and finds providers' services that match the profile of the request, or the "Candidate Services". The Select process differentiates amongst the candidates generated from the Discover process to produce the "Master Services" ("services" because iterative processing to decompose services may produce multiple "masters" at different levels), and negotiates for acceptable terms to use the selected Web services. The Configure process composes the services into an execution plan that is then validated and verified for accuracy to produce "Certified Services".

The Deploy process coordinates and secures the resources and the environment to produce "Secured Services", and the "Fulfillment Package" (discussed in Section 3.3) is created. The Deliver process is where the Web services are executed, workflow is enacted, and services are delivered to fulfill the request.

With Web services still in the infancy stage, it seems strange to be talking about the retirement issues; however, the services that a provider registers must be viable to justify committing resources to provide the service. The Retire process involves establishing the criteria for the retirement, and using the feedback measures to evaluate the results against the criteria.

3.2.3 Specification Languages

In order to model the framework's data-centric concepts, we use the Knowledge/Data Model (KDM) [48] that incorporates an object-oriented view of data, together with knowledge regarding its usage. The KDM is an extension of the semantic data model and draws heavily upon the features of the functional data model, object-oriented paradigm, and knowledge-based systems. The KDM models the semantics of an enterprise, including data semantics, as captured by semantic data models and knowledge semantics, as captured in knowledge-based systems. The significant features of the KDM data model are:

- The incorporation of heuristics to model inferential relationships,
- The capability to organize these heuristics and to associate them with specific items involved in the inferential relationships,
- The capability to incorporate heuristics and constraints in a tightly coupled (unified) manner,
- The ability to define inferred (virtual) objects,
- A unified representational formalism for knowledge and data, and
- A mechanism that allows for abstract knowledge typing, that is, handling rules and constraints as objects.

The discussion below presents the semantic primitives available in the KDM:

- **Generalization:** Generalization (the inverse of which is **specialization**) provides the facility in the KDM to abstract similar object-types into a more general or higher-level object-type (an object-type is a collection of related objects). This is done by means of the "is-a" relationship (e.g., the object-types kdsdService and kdsdAgent and generalized into the kdsdResource object-type). This generalization

hierarchy establishes the inheritance mechanism (e.g., kdsdService and kdsdAgent inherit the properties and methods of kdsdResource).

- **Aggregation:** Aggregation (the inverse of which is **decomposition**) is an abstraction mechanism where an object is related to the components that constitute it via the "is-part-of" relationship (e.g., the objects kdsdPreCondition, kdsdPostCondition and kdsdRule are part of kdsdGoal data).
- **Classification:** Classification (the inverse of which is **instantiation**) provides a means whereby specific object instances can be grouped together and considered to be an object-type. This is accomplished through the use of the "is-instance-of" relationship (e.g., "Knowledge Sifter" is a specific instance of kdsdAgent).
- **Membership:** Membership is an abstraction mechanism that specifically supports the "is-a-member-of" relationship between objects or object-types (e.g., kdsdInteroperation is an object-type containing kdsdInteroperationLevel and kdsdLanguage members).
- **Constraint:** This primitive is used to place a constraint on some aspect of an object, operation, or relationship via the "is-constraint-on" relationship. Both implicit (e.g., only return the top 25 ranked services from a search) and explicit (e.g., the attribute kdsdInteroperationLevel is restricted to the values of "Data", "Presentation", and "Process").
- **Heuristic:** This primitive is used to attach a heuristic via the "is-heuristic-on" relationship (e.g., Partners who are in bankruptcy are a bad risk; therefore, do not use services from providers who are in bankruptcy). Heuristics are expressed in the form of rules. Heuristics allow information about an object to be inferred; thus, heuristics provide an information derivation mechanism that results in greater informational content than is present in the stored data alone [40].
- **Method:** This primitive is used to model the behavior of object-types and to manipulate object-types. For example, an object-type might invoke a "search" method in order to find available services.
- **Temporal:** The temporal relationship is used to model specific task or event oriented object-types that are related by synchronous or asynchronous characteristics (e.g., the tasks in processing a request via Web services). Synchronous objects are related to other synchronous objects by either the predecessor or successor relationship. Asynchronous objects are related to other asynchronous objects by a concurrent or parallel notion. Temporal primitives are also used for task planning and workflow analysis.

The most generic construct in the KDM is the object type and is specified in the Knowledge/Data Language (KDL) template depicted in Figure 7, which shows a general template for an object-type (class) specification employing the KDL. KDL reserved words are shown in uppercase letters. Identifiers shown in lowercase letters are place holders for user input. Optional items in the template are enclosed in square brackets, and at least one of each of the items contained in curly brackets must be part of the specification [35].

In order to model the framework's process-centric concepts, we introduce the Knowledge-based Dynamic Services/Process Model (KDSPM), which is built from the pattern set forth by the KDM/KDL. The accompanying object-type definition, Knowledge-based Dynamic Services/Process Language (KDSPL), is shown in Figure 8. Although the initial target of the KDSPM/KDSPL is used to specify SWS, the term 'service' is used in lieu of 'semantic web services' because the application of the specification is not necessarily restricted just to Web services. The specification is powerful enough to be used for non-Web services based applications.

```
object-type ::=
        OBJECT_TYPE class-name HAS
                [ SUPERTYPES: // Generalization
                        class-name { , class-name };]
                [ SUBTYPES: // Specialization
                        class-name { , class-name } [ HIDING function-list ];]
                [ ATTRIBUTES: // Aggregation
                        { attribute-name: type-name
                                [ WITH CONSTRAINT: constraint ];}]
                [ MEMBERS: // Membership (Association)
                        { member-name: [ SET OF | LIST OF ] class-name
                                [ INVERSE OF member-name [(class-name)]]
                                [ WITH CONSTRAINT: constraint ];}]
                [ CONSTRAINTS: // Knowledge to enforce integrity
                        { constraint; }]
                [ HEURISTICS: // Knowledge to derive/infer information
                        { rule; }]
                [ METHODS: // Specifications of computations and behavior
                        { method; }]
END class-name;
```

Figure 7. Syntax of KDL Object-Type Specification

The KDSPM borrows some of the primitives from the KDM. The new primitives that the KDSPM introduces are discussed below. Notice that object-type-name can be specified as a kdsd-object to denote objects from the KDL or a kdsp-object to denote objects from the KDSPL.

- **Goals:** GOALS is taken from the Web Services Modeling Framework [20] to capture the purpose of the objects. The KDM/KDL is adapted to include GOALS as well.
- **Owner:** The OWNER specifies the primary agent responsible for the process.
- **Steward:** The STEWARD argument specifies the secondary agent that may handle the process for the OWNER.
- **Predecessors:** The PREDECESSORS argument specifies process objects that hand off processing to the object.
- **Successors:** The SUCCESSORS argument specifies the next objects in the processing.
- **Target:** TARGET argument specifies the resultant entity (i.e., a particular agent) for which the object is intended to be used.
- **Semantic Web Map:** The SEMANTICWEBMAP argument specifies object-specific mapping to the Semantic Web (as opposed to the global mappings discussed in the next section). SEMANTICWEBMAP is also used in the adapted KDM/KDL.
- **Steps:** The STEPS argument details the actions involved in executing the process. The step-name and step-description annotate the step. The sequence-number denotes the chronological occurrence of the step, and mode specifies the manner in which the step is to be executed (i.e. parallel, sequential, interval, etc.). The step-predecessor designates the previous step dependency if it exists. The agent-delegate specifies the agent that executes the step. The operation argument specifies that another KDS-PL object, method or a manual intervention performs the step.

```
object-type ::=
        OBJECT_TYPE class-name HAS
                [ GOALS: ({goal-name}+);]
                [ TASK: ({object-type-name}+);]
                [ THREAD: ({object-type-name}+);]
                [ SUPERTYPES: // Generalization
                        class-name { , class-name };]
                [ SUBTYPES: // Specialization
                        class-name { , class-name } [ HIDING function-list ];]
                [ OWNER: ({object-type-name }+);]
                [ STEWARD: ({object-type-name }+);]
                [ POINT-OF-CONTACT: ({string }+);]
                [ PREDECESSOR: ({object-type-name }+);]
                [ SUCCESSOR: ({object-type-name }+);]
                [ TARGET: (object-type-name);]
                [ STEPS: (      {step-name}
                                {sequence-number}
                                {step-description}
                                {mode (parallel | sequential | interval | loop | while …)
                                        (step-predecessor {object-type-name})
                                }
                                {agent-delegate (agent-name agent-role{line staff})}
                                {resources ({object-type-name}+)}
                                {operation
                                        (operation-name)
                                        (process-object { object-type-name} |
                                        (method { method-name} |
                                        (manual intervention {string})
                                }
                                {triggers (event-name | external-trigger-name)}+
                        )
                ]
                [ CONSTRAINTS: // Knowledge to enforce integrity
                        { constraint; };]
                [ HEURISTICS: // Knowledge to derive/infer information
                        { rule; };]
                [ SEMANTICWEBMAP ({object-type-name | function-name}
                        {semanticwebobject})+;]
        END class-name;

object-type-name::= (kdsd-object | kdsp-object)
semanticwebobject ::= (     {protocol}
                            {semantic-web-data-object}
                            {semantic-web-statement}
                        )

semantic-web-statement::=(      {SUBJECT object-type-name}
                                {PREDICATE relation-name}
                                {OBJECT object-type-name}
                            )
```

Figure 8. Syntax of KDSPL Object-Type Specification

3.2.4 Mapping

The Mappings is a superclass of all mappings to ensure consistency across the methods to bridge the KDL/KDSPL to other technologies and standards. The mapping process at a high level denotes a source to

a destination along with metadata for each to denote specific details to take into consideration in the mapping. For example, the top-level superclass of mappings holds the basic structures, while the specific protocol can be specified for the specific mappings.

The mapping from the KDL to database schemas and object-oriented classes is fairly straight-forward (intentionally so). The KDL object-types map to classes or tables, attributes map directly to class attributes or database attributes, and methods map to methods. The heuristics rules map to specialized database tables that match their constructs.

The mapping to SWS is achieved at two levels: globally and locally to the object. The global mapping involves mapping the arguments of the KDL and KDSPL to the respective elements in the destination protocol. The local mapping is generated from explicitly specifying the protocol to be used in an object in an instantiation of the KDL

The profiles of the various agents are embedded throughout the KDSPL. The mapping to agent profiles identifies the points where agents are specified and organizes into a profile that contains the responsibilities of the agent within a given context. From this profile, the specification for a given agent can be engineered into a working component.

3.3 DSWS Functional Architecture

This research introduces the use of a "fulfillment package" (FP) to distribute policies and workflow steps, manage resources, and convey state of a workflow enactment in order to facilitate the maintenance of the organization's infrastructure. To distribute and share rules (as mentioned in the Grid technologies discussion), the concept is to identify catalogs and their associated metadata. Metadata (metadata versus the actual rules to reduce the payload) such as version and last update date is distributed in the FP, and the partners check to see if they need to update their enterprise knowledge base from the endpoint designated in the FP. To reduce the potentially long-running transactions' dependence on network connectivity and resources, the FP also carries the workflow steps and state.

3.3.1 Functional Federation Architecture

The primary function of the Functional Federation Architecture (FFA) is to establish the governance of the enterprise operations where possible. Although the majority of the activities within the FFA are not automated – they establish the fundamental guidance for the partners to operate as an organization. The federation may be very loose and informal, or be legitimized by formal contracts. The important facet is that these issues need to be dealt with at some level.

In the Federate Functions process, the VE handles the strategic layer of management. This is where the VE decides what functions will reside where within the organization. The enforcement mechanisms for the Articles of Federation and Service Level Agreements are established here. The roles and responsibilities are determined of the partners along with their level of participation in the organization. The governing policies set boundaries for the VE, like how a partner joins or leaves the VE, confidentiality, and which resources made available only within the organization and which are made available beyond the partners.

In the Federate Agents process, the VE addresses the value-chain management layer, and decides the distribution of agents and processes, and the associated owners/stewards within the organization. The allowable process controls and methods to handle anomalies (errors, exceptions, constraint violations) are defined. The level of management allowed over the workflow and the methods to coordinate constraints are defined as well.

In the Federate Knowledge process, the VE decides where and how the knowledge will be stored, maintained, and distributed. The ownership and stewardship of those resources also needs to be identified.

3.3.2 Functional Agent Services Architecture

Agents may be thought of as software programs [63] that act on behalf of humans, robots, or other programs, and exhibit autonomous, purposeful behavior. The architecture used in the DSWS Architecture is the Functional Agent Services Architecture (FASA) that is based on an agency-based approach in which agents are organized in agencies. The FASA is a three-layer architecture that contains line agents and staff agents to reflect that some agents are involved in work specifications while others play support roles [32, 33, 35]. Aspects from the Distributed Agent Resource (DAR) Protocol are incorporated into FASA. DAR is used to manage resources using such constructs as enterprise and local agency constraints [11]

The User Layer is supported by the User Agency that coordinates a collection of services such as task specification, planning, user profile administration, and order tracking. The concept is that users would visit portal of the e-enterprise, compose their request in terms of a high-level task, and interact with the planning agent to decompose the task into a plan that would be submitted to the next layer, Intelligent Middleware Services.

The Intelligent Middleware Services layer is supported by the Functional Services Agency whose agents include line agents for web service discovery, negotiation, contracting, and composition. These agents receive the task specification and plan from the User Agency, and search for appropriate Internet-based web services that can accomplish the tasks. Those web services may have already been vetted for use by the e-enterprise by the Services Coordination Agency residing at the Web Services Layer.

Staff agents include ontology, curation, and QoS monitoring, among others. For example, the ontology agent is responsible for maintaining the ontology of tasks and services provided by the e-enterprise. Curation agents are involved in identifying, storing, and evolving a repository of successful patterns of web services that have been successful in performing high-level user tasks.

The Web Services Layer is supported by the Services Coordination Agency whose agents support service mediation, workflow coordination, and transaction management. The Functional Services Agency submits a web service configuration plan, and the Services Coordination Agency verifies that the services are available, schedules the various sub-transactions, and executes the plan on behalf of the Functional Services Agency. The Services Coordination Agency also monitors the progress of each transaction, maintains records of the transactions, their status, and QoS for future processing and reporting. In addition, several other agents reside at this layer and perform their collaborative functions to place new services in the service registry. This involves cooperation among the registration, certification, and testing agents. In order for a new service to be a candidate for inclusion into the e-enterprise repositories, it must be tested, annotated with QoS attributes, and certified to perform at advertised levels of service [32] .

3.3.3 Functional Knowledge Architecture

The Functional Knowledge Architecture (FKA) contains knowledge that is stored in a Semantic Web form that is readily incorporated into SWS and knowledge that is pulled from stores that must be mediated to be used by SWS. An example of the Semantic Web form are ontologies stored in OWL, where an example of the non-Semantic Web form is a reference table on a mainframe computer.

3.3.4 Web Services Protocols

Protocols are an essential backbone for Web services that establish the expected means to communicate between end-points. There are numerous protocols used to support Web services, some of which are mature but most of still emerging at varying levels of maturity. The most prominent protocols for Web services are UDDI, WSDL and SOAP. BPEL4WS is gaining prominence as a processes specification standard. This research suggests new extensions to some of these protocols to facilitate interoperation with the industry base. The primary extensions involve reflecting preferences and constraints.

A sampling of the current protocols in the Web services Stack is presented below in Figure 9, which comes from an excellent overview of the Web services protocols found at [62]. For an explanation of Web

services protocols, as well as Security Assertion Markup Language (SAML) [44], please see the associated summary at [50].

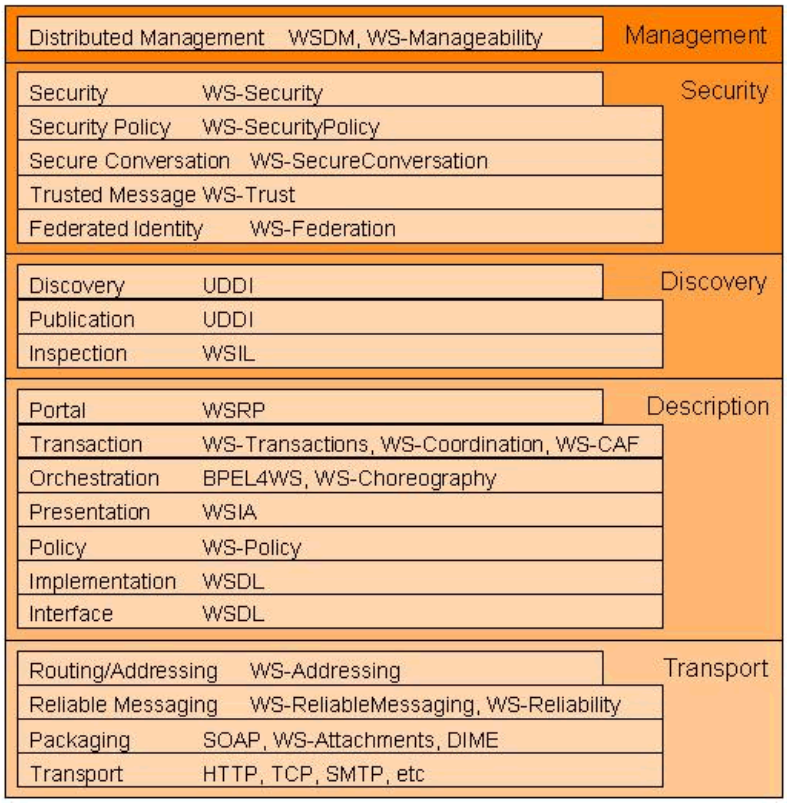

Figure 9. The Current Web Service Protocol Stack

The protocol and technology offerings for web services are numerous and rapidly expanding in order to make web services more robust. A sampling of some of the emerging advanced protocols is presented below:

Web Services Outsourcing Manager (WSOM)

Web Services Outsourcing Manager (WSOM) is a framework that enables dynamic composition of Web service flow based on customer requirements. The customer requirements are analyzed and optimized to generate an annotation document for business process outsourcing. This service-oriented architecture allows effective searching for appropriate Web services and integration of them into one composite Web service for performing a specific task. Meanwhile, it provides a seamless, integrated framework for composition of template-based business processes and event-driven business processes [30].

Web Services Choreography Interface (WSCI)

WSCI describes how Web Service operations can be choreographed in the context of a message exchange in which the Web Service participates. WSCI also describes how the choreography of operations should expose relevant information, such as message correlation, exception handling, transaction description and dynamic participation capabilities [2].

Web Services Business Integration (WSBI)

A standards-based approach to integration that capitalizes on the evolutionary cycle of infrastructure software that combines the current state-of-the-art of Web service Standards with an integration "stack" of technology of management and optimization, BPM and workflow, security, reliable messaging and transactions, data transformation and business logic, interoperability, and applications [59].

Web Services Choreography

"Web Services Choreography concerns the interactions of services with their users. Any user of a Web Service, automated or otherwise, is a client of that service. These users may, in turn, be other Web Services, applications or human beings. Transactions among Web Services and their clients must clearly be well defined at the time of their execution, and may consist of multiple separate interactions whose composition constitutes a complete transaction. This composition, its message protocols, interfaces, sequencing, and associated logic, is considered to be a choreography [3]."

4 Scenario

The scenario is based on the emergency services domain responding to a hurricane situation, and it highlights the primary features of the framework. The scenario revolves around the tasks in the life-cycle as they are performed to fulfill the request. The items prefixed with "kdsd" denote KDL objects involved in the flow, and items prefixed with "kdsp" denote KDSPL objects used in the process. The focus of the scenario is to show an abstraction of the elements needed to manage such an event versus the actual data of a hurricane event.

The scenario, as shown in Figure 10, is explained in two manners: chronologically as the sequence of events take place during scenario execution, and by notations of the various elements surrounding the execution. The steps are numbered, whereas the notations are marked by underlined alphabetic characters (e.g., **A**).

Steps 1a. Embed Provider/Service Knowledge in UDDI (apriori) and 1b. Embed Provider/Service Knowledge in WSDL (a priori) entail publishing the enhanced knowledge of the provider, and the services it provides, in semantically-enhanced ontological markup to WSDL and UDDI repositories before the request is made (a priori). 2. Event Occurs represents the trigger within the system to indicate that a hurricane has occurred and requests need to be made to commence fulfilling the associated needs from the emergency as represented in 3. Commence Request.

Some preparatory work has been done in Prepare for Request prior to the request in order to plan for the processing of the request. For example, two other a priori items, kdsdHurricaneProfile and kdsdHurricanePattern, are combined in 4. Aggregate Knowledge to form the foundation of the request profile. kdsdHurricaneProfile is an instantiation of the kdsdScenario class, and it stores knowledge about the scenario called Hurricane such as the urgency of the scenario so the proper priority can be placed on the request. kdsdHurricanePattern establishes the workflow steps necessary to handle a typical hurricane situation. 5. Compile Master Request assigns the pulls the profiles together and assigns constraints of preferences that pertain to where the hurricane hit and what special circumstances need to be considered in processing the request.

As seen with 6. Perform Subprocesses and 7. Process workflow to find services exemplifies that the KDSPL can handle both the steps to process the request as well as the workflow depicted in the service specification. In the specific case of the Discover subprocesses of kdspClassifyRequest, kdspSearchForServices and kdspCompileResults, an agent such as KnowledgeSifter [34] is designated in the specification to find potential services that can fulfill the requests. Knowledge Sifter is an agent-based ontology-driven search agent that can mediate ontological terms to access heterogeneous repositories. It can therefore access application-domain-specific registries to select services based on user-specified features, such as quantity, availability, transportation costs, etc. In the 7. Process workflow to find services process, the workflow is iterated through the first time to identify and configure those Web services involved in fulfilling the request.

The kdspDifferentiate, kdspCompare, kdspNegotiate steps in the Select process culminate in the 8. Identify Master Service classifying a given Web Service as the primary service to handle that part of the process. The "Master Service" may itself be further decomposed to identify other simple services or other "Master Services". An important point in understanding the scenario at this juncture is that 9. Iterate until all services ID'd reflects the Discover, Search and Configure processes are performed until all the Web

services that are necessary to proceed are identified. Some later steps may require repeating these processes to find services downstream in the execution.

10. Build Fulfillment Package involves preparing the package that will carry execution workflow, state information and enterprise catalog information throughout the Deliver process. 11. Enact workflow to deliver services iterates through the workflow once again in order to deliver the services. Another important point is that the Configure process may adjust and refine this workflow to optimize the Deliver process. In the execution of the request, 12. Anomaly Event Occurs when it is found the there are not enough blankets, which triggers kdspConstraintViolation. The workflow designated in kdspConstraintViolation exemplifies the versatility of the three means to specify operations in 13. Process Alternatives, 14. Invoke KDL Method and 15. Perform Manual Operation. 13. Process Alternatives reflects that the framework incorporates the identification and processing of alternatives and the execution of another process object; however, this example only identifies the alternatives. The kdsdSolicitRequestor method from the KDL specification is invoked to ask the requestor what alternative path to take. A manual operation can also be specified as is shown by requiring the requestor to make a choice.

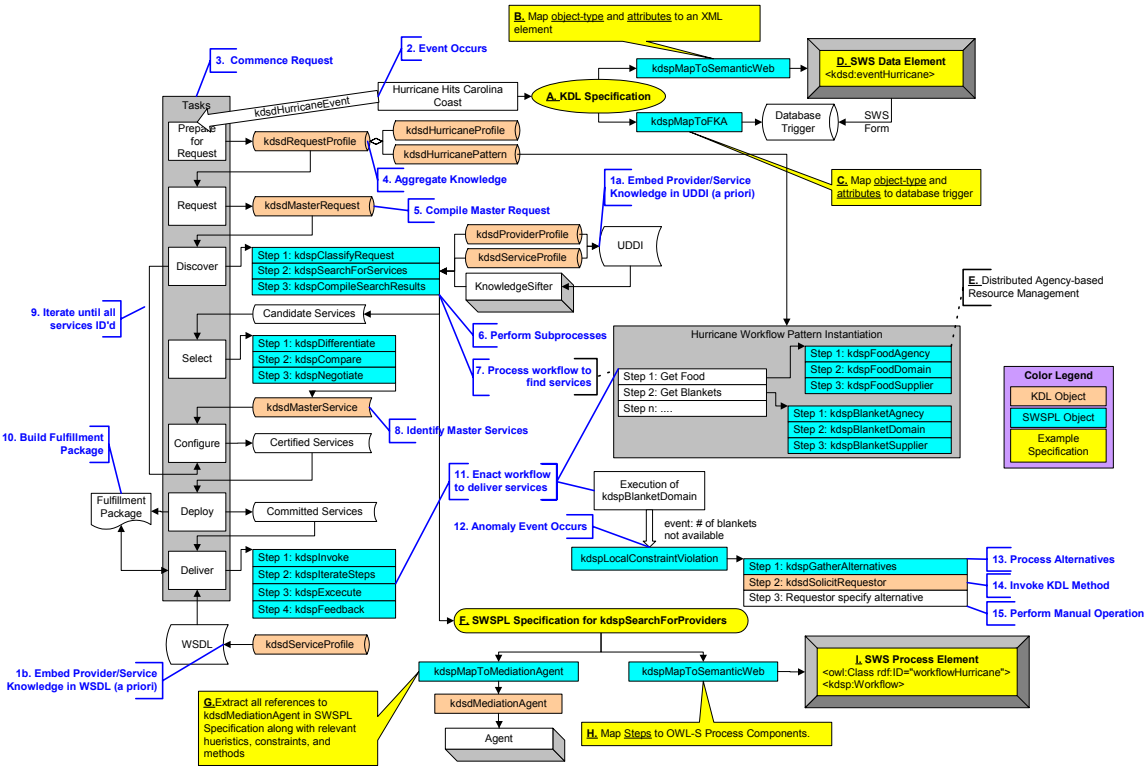

Figure 10. Scenario Process and Objects

Notation **A** depicts the KDL specification of the object kdsdHurricaneEvent as shown in Figure 11. The SUPERCLASS references a subclass (kdsdSensor) of the KDL object kdsdEvent The ATTRIBUTES show typical information that the event would pass along for processing. The heuristic indicates that if the temperature is below 60 Fahrenheit then more blankets are needed. The METHOD kdspPrepareForHurricane references a process object that will specify the workflow to prepare for the hurricane.

Notation **F** provides a sample KDS-PL Specification for the kdspSearchForProviders object shown in Figure 12. Notice that the TASK and THREAD (kdspDiscover and kdspManagement, respectively) establish the context of the process. The OWNER references the KDL object kdsdSearchAgent, which has its own set of attributes and methods in its KDL object specification. The STEWARD specifies kdsdKnowledgeSifter, which would be specified as a kdsdFunctionPoint object. kdsdFunctionPoint is a

subclass of kdsdRepository which stores the endpoint, or URI, of the agent. As a PREDECESSOR, kdspClassifyRequest is the previous process in the chain. The next step in the process will be the SUCCESSOR kdspCompileSearchResults.

object-type::=kdsdHurricaneEvent						
:SUPERTYPES	kdsdSensor					
:ATTRIBUTES	kdsdLocation	:TYPE	String	:CONSTRAINT		
	kdsdStrength	:TYPE	Integer	:CONSTRAINT	'< 5'	
	kdsdTemperature	:TYPE	Integer	:CONSTRAINT	'Celcius scale'	
:CONSTRAINT						
:HEURISTICS	If kdsdTemperature < 60F, secure extra blankets					
:METHODS	kdspPrepareForHurricane					

Figure 11. kdsdHurricaneEvent KDL Specification

This sample process object has the single step of SearchUDDI, with kdsdKnowledgeSifter designated as the AGENT-DELEGATE as well. The operation is specified as the Search method from kdsdKnowledgeSifter. The CONSTRAINT kdsdSearchReturnLimit specifies to limit the search results to the top 25 ranked services. The rule in HEURISTICS says that the enterprise does not want to deal with businesses that are in bankruptcy. The SEMANTICWEBMAP maps the AGENT-DELEGATE (Knowledge Sifter) object to a OWL-format specification called searchStep.

object-type::=kdspSearchForProviders			
:GOALS	ProviderSearchGoal (Find services from providers that meet the goals of the request)		
:TASK	kdspDiscover		
:THREAD	kdspManagement		
:OWNER	kdsdSearchAgent		
:STEWARD	kdsdKnowledgeSifter		
:PREDECESSORS	kdspClassifyRequest		
:SUCCESSORS	kdspCompileSearchResults		
	:STEPNAME	SearchUDDI	
	:SEQUENCE-NUMBER	1	
	:STEP-DESCRIPTION	Search the UDDI registry for acceptable providers and	
	:MODE	Sequential	
	:AGENT-DELEGATE	kdsdKnowledgeSifter	
		:AGENT-ROLE	LINE
	:OPERATION	searchUDDI	
		:METHOD-NAME	kdsdKnowledgeSifter.Search
:CONSTRAINT	kdsdSearchReturnLimit (Return only the top 25)		
:HEURISTICS	Partners who are in bankruptcy are a bad risk; therefore, do not use services from providers who are in bankruptcy"		
:SEMANTICWEBMAP	:FUNCTION-NAME	AGENT-DELEGATE	
	:PROTOCOL	OWL	
	:OBJECT-NAME	searchStep	

Figure 12. kdspSearchForProviders KDS-PL Specification

Notations **B** (kdspMapToSemanticWeb) and **C** (kdspMapToFKA) both use the object-type and the attributes to their targets. Notations **G** (kdspMapToMediationAgent) and **H** (kdspMapToSemanticWeb) show the pseudo-code to map to their respective targets. Notations **D** (kdsdHurricaneEvent) and **I** (kdspSearchForProviders) show a potential tagging for the kdsdHurricaneEvent in SWS format. The arrow from notation **D** back to the database trigger denotes that the SWS format can be stored in an XML database. Notation **E** exemplifies the instantiation of workflow based on the Distributed Agency-based Resource Management methodology.

5 Conclusions

This paper has introduced the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework which supports the modeling, specification, design, implementation and deployment of systems composed of SWS. The goal of the framework is to support the automatic discovery, composition and management of Web services for the VE. This VE is the paradigm we use to denote that the KDSWS can handle both Inter-Enterprise and Intra-Enterprise coordination and interoperation issues.

At present, Web services technology is in its infancy, and it has adherents in both Academe and Industry. In order to accelerate the introduction of new concepts and systems, we need approaches which automate the entire Web services life-cycle including the specification, design, implementation, deployment, and management. The KDSWS Framework provides a formal model-based approach to Web services specification. The meta-meta-model combines KDSWS constructs with RDF and RDFS concepts. The meta-meta model is couched in UML terminology which can be mapped to OWL.

There are so many issues involved in this area, that we propose the KDSWS Framework as a way to combine three important, and inter-related, viewpoints, as depicted in Figure 4.

1) The KDSWS Process viewpoint addresses issues related to workflow, transaction-control, security, and interoperation and are handled by "threads".

2) The KDSWS Design Specification viewpoint models objects, relationships, constraints, heuristics, and processes using the Knowledge/Data Model and Language (KDM/KDL) and extensions (KDSPM and KDSPL) to handle special features of Semantic Web Service specifications. The specification can then be mapped to standard protocols such as UDDI, WSDL, BEPLWS and OWL-S.

3) The KDSWS Functional Architecture provides an agent-based architecture to implement systems via composable Semantic Web Services. The architecture includes Functional Architectures for knowledge represented in repositories, federation policy, rules and agents, Web service protocols, and agent services to manage various aspects of deploying Semantic Web Services.

Referring to the previously presented issues that Web services need to address [13], this framework addresses *Semantic Unification* with the use of mediation and integrated data/process specification. *Message Behavior* is addressed in the transportation and transaction threads. *Endpoint Discovery* is enhanced through the proposed extensions to UDDI and WSDL. *Message Security and Trust Relationships* is dealt with specifically in the Security thread as well in the Federation activities. *Process Management* is the focus of the KDSPL process and workflow steps. *Integration Standards* are facilitated with the integrated design and allows adapting more easily to emerging standards. *Legacy Application Connectivity* is handled to by the federation activities and the associated mediation.

Finally, our research indicates that the KDSWS semantic modeling techniques and methodology, when applied to service-oriented systems exemplified by Semantic Web Services, helps to address the plethora of issues needed to successfully deploy them in real-world applications. The multiple viewpoints help to isolate and identify important issues and the mappings from viewpoint to viewpoint assure that the structures, operations, and constraints are properly mapped and preserved.

Acknowledgements

This work was sponsored by a NURI from the National Geospatial-Intelligence Agency (NGA). This work was also supported in part by the Advanced Research and Development Activity (ARDA). Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the U. S. Government

6 References

1. Arkin, A. Business Process Modeling Language, BMPI.org, 2002.
2. Arkin, A., Askary, S., Fordin, S., Jekeli, W., Kawaguchi, K., Orchard, D., Pogliani, S., Riemer, K., Struble, S., Takacsi-Nagy, P., Trickovic, I. and Zimek, S. Web Service Choreography Interface (WSCI) 1.0, W3C, 2002.
3. Austin, D., Barbir, A., Peters, E. and Ross-Talbot, S. Web Services Choreography Requirements 1.0, W3C, 2003.
4. Bajaj, S., Della-Libera, G., Dixon, B. and al., e. Web Services Federation Language (WS-Federation), 2003.
5. Bechhofer, S., Harmelen, F.v., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F. and Stein, L.A. OWL Web Ontology Language Reference, W3C, 2003.
6. Berners-Lee, T. Web Services - Semantic Web, 2003.
7. Berners-Lee, T., Hendler, J. and Lassila, O. The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities *Scientific American*, 2001, 34-43.
8. Booth, D., Haas, H., McCabe, F. and Newcomer, E. Web Services Architecture, W3C, 2003.
9. Bosworth, A., Developing Web Services. in *Proceedings 17th International Conference on Data Engineering*, (2001), 477 -481.
10. Brittenham, P. An overview of the Web Services Inspection Language, 2002.
11. Brodsky, A., Kerschberg, L. and Varas, S. Resource Management in Agent-Based Distributed Environments. *Lecture Notes in Computer Science*, *1652/1999*. 61-85.
12. Brown, A., Johnston, S. and Kelly, K. Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications *A Rational software whitepaper from IBM*, 2003.
13. Bussler, C., Fensel, D. and Sadeh, N. Semantic Web Services and Their Role in Enterprise Application Integration and E-Commerce, 2003.
14. Cerami, E. *Web Services Essentials*. O'Reilly, Beijing ; Sebastopol, CA, 2002.
15. Chinnici, R., Gudgin, M., Moreau, J.-J., Schlimmer, J. and Weerawarana, S. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C, 2002.
16. DAML.org. OWL-S: Semantic Markup for Web Services, 2003.
17. DAML.org. Reference description of the DAML+OIL (March 2001) ontology markup language, 2001.
18. DublinCore. Dublin Core Metadata Element Set, Version 1.1: Reference Description, 2003.
19. ebXML. About ebXML.
20. Fensel, D. and Bussler, C. The Web Service Modeling Framework WSMF.
21. Fensel, D., van Harmelen, F., Horrocks, I., McGuinness, D.L. and Patel-Schneider, P.F. OIL: an ontology infrastructure for the Semantic Web *IEEE Intelligent Systems*, 2001, 38-45.
22. Foster, I., Kesselman, C. and Tuecke, S. The Anatomy of the Grid: Enabling Scalable Virtual Organizations.
23. Giovanni Della-Libera, M., Brendan Dixon, M., Mike Dusche, M. and al., e. Federation of Identities in a Web Services World.
24. Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J. and Nielsen, H.F. SOAP Version 1.2 Part 1: Messaging Framework, W3C, 2003.
25. Haas, H. and Orchard, D. Web Services Architecture Usage Scenarios, W3C, 2003.
26. Heidel, G. Web Services Standards: Can there be a consensus? *www.wsj2.com*.
27. Hendler, J. DAML: DARPA Agent Markup Language effort, (http://www.daml.org/), 2002.
28. IBM. Specification: Business Process Execution Language for Web Services Version 1.1, 2003.
29. IBM. Specification: Web Services Security, http://www-106.ibm.com/developerworks/library/ws-secure/, 2002.

30. IBM. Web Services Outsourcing Manager, 2002.
31. Kaye, D. *Loosely Coupled: The Missing Pieces of Web Services*. RDS Press, Marion County, California, 2003.
32. Kerschberg, L. Functional Approach to in Internet-Based Applications: Enabling the Semantic Web, E-Business, Web Services and Agent-Based Knowledge Management. in Gray, P.M.D., Kerschberg, L., King, P. and Poulovassilis, A. eds. *The Functional Approach to Data Management*, Springer, Heidelberg, 2003, 369-392.
33. Kerschberg, L. (ed.), *Knowledge Management in Heterogeneous Data Warehouse Environments*. Springer, Munich, Germany, 2001.
34. Kerschberg, L., Chowdhury, M., Damiano, A., Jeong, H., Mitchell, S., Si, J. and Smith, S., Knowledge Sifter: Ontology-Driven Search over Heterogeneous Databases. in *SSDBM 2004, International Conference on Scientific and Statistical Database Management*, (Santorini Island, Greece, 2004 (Submitted for Publication)), IEEE.
35. Kerschberg, L. and Weishar, D.J. Conceptual Models and Architectures for Advanced Information Systems. *Applied Intelligence*, *13*. 149-164.
36. Ko, I.-Y. and Neches, R. Composing Web Services for Large-Scale Tasks *IEEE Internet Computing*, 2003, 52-59.
37. Lassila, O. and Swick, R.R. Resource Description Framework (RDF) Model and Syntax Specification, W3C, 1999.
38. McIlraith, S.A., Son, T.C. and Zeng, H. Semantic Web services. *Intelligent Systems, IEEE*, *16* (2). 46-53.
39. Menasce, D.A. QoS issues in Web services. *Internet Computing, IEEE*, *6* (6). 72-75.
40. Miller, J., Potter, W., Kochut, K., Keskin, A. and Ucar, E. The Active KDL Object Oriented Database System and Its Application to Simulation Support. *Journal of Object-Oriented Programming, Special Issue on Databases*. 30-45.
41. Miller, J.A., Potter, W.D. and Kochut, K.J. On the Integration of Knowledge, Data, and Models.
42. Nayak, N., Bhaskaran, K. and Das, R., Virtual enterprises-building blocks for dynamic e-business VO -. in *Information Technology for Virtual Enterprises, 2001. ITVE 2001. Proceedings. Workshop on*, (2001), 80-87.
43. OASIS. Management Using Web Services: Architecture, 2003.
44. OASIS. Security Assertion Markup Language (SAML) (http://www.oasis-open.org/committees/security), OASIS, 2002.
45. OMG. Meta Object Facility (MOF) Specification Version 1.3, http://www.omg.org/docs/formal/00-04-03.pdf. ObjectManagementGroup ed., Object Management Group, 2000.
46. Ontoknowledge. Description of OIL.
47. Paolucci, M. and Sycara, K. Autonomous Semantic Web Services *IEEE Internet Computing*, 2003, 34-41.
48. Potter, W.D. and Kerschberg, L. The Knowledge/Data Model: An Integrated Approach to Modeling Knowledge and Data. in Meersman, R.A. and Sernadas, A.C. eds. *Data and Knowledge (DS-2)*, Amsterdam: North Holland, 1988.
49. Reid, R.L., Rogers, K.J., Johnson, M.E. and Liles, D.H. Engineering the Virtual Enterprise, Automation & Robotics Research Institute.
50. Roadmap, C.W.S. Web Services Protocols Summary.
51. Scheer, A.-W.a.N., Markus *ARIS Architecture and Reference Models for Business Process Management*, 2000.
52. searchcio.techtarget.com. Grid computing.
53. Shue, M. Business Integration. webMethods ed., 2003.
54. Stevens, M. Service-Oriented Architecture Introduction, Part 1, 2002.
55. UDDI. UDDI Version 3 Features List.
56. Vielmetti, E. The (R)evolution of Useful Web Services. *IEEE Communications Magazine*, *37* (9). 92-94.

57. W3C. Extensible Markup Language (XML):Introduction.
58. webMethods. GLUE User Guide, 2003.
59. webMethods An Introduction to Web Service-Based Integration (WSBI).
60. Webservices.org. Web Services Security Roadmap issued jointly by Microsoft, IBM and Verisign.
61. Whatis.techtarget.com. Virtual Organization.
62. Wilkes, L. The Web Services Protocol Stack, CBDI Web Services Roadmap, 2004.
63. Wooldridge, M. Issues in Agent-Based Software Engineering. *Cooperative Information Agents*. 1-18.
64. Wu, D., Sirin, E., Parsia, B., Hendler, J. and Nau, D., Automatic web services composition using SHOP2. in *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, (Sanibel Island, Florida, 2003).