International Journal of Cooperative Information Systems
Vol. 13, No. 4 (2004) 441–485
© World Scientific Publishing Company

A FRAMEWORK FOR DYNAMIC SEMANTIC WEB SERVICES MANAGEMENT

RANDY HOWARD* and LARRY KERSCHBERG†

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

The use 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. 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 Organizations, 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 needed for their successful deployment in enterprise-scale systems for Virtual Organizations.

This paper introduces such a framework, 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 with an emphasis on adaptability to rapidly changing environments and standards: (1) semantic specification of both service's and requestor's capabilities, constraints and preferences including quality of service, trust, and security; (2) transaction control and workflow management; and (3) resource management, interoperation and evolution of the Virtual Organization.

Keywords: Semantic web services; virtual organization; ontology; workflow; agent-based systems.

1. Introduction

The relatively new concept of *Web services* is important to both e-Business and e-Government in that applications may exchange functionality and information over the Internet. 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.[1]

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. These standards include Universal Description, Discovery and Integration (UDDI),[2] Simple Object Access Protocol (SOAP),[3] and Web Services Description Language (WSDL).[4]

"Semantic Web Services" (SWS)[5] is the term that describes our research approach. The ultimate vision of SWS is the dynamic discovery, configuration, and deployment of a Virtual Organization (VO) from services distributed over heterogeneous systems, thereby creating a VO from collections of services. However, this vision is far from reality, as companies presently 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.

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 VO. SWS allow Web information to be structured not only for human consumption, but also for machine processing.[6] This base of Semantic Web technologies involves Resource Description Framework (RDF),[7] DARPA Agent Markup Language (DAML),[8] Ontology Inference Layer (OIL),[9,10] DAML+OIL,[11] and Web Ontology Language (OWL).[12] OWL-S[13] (formerly DAML-S[14] and an extension of OWL) is a leading specification for the semantic description of Web services in order to facilitate their automation. 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".[15] 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.[16,17]

This paper introduces 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 both service's and requestor's capabilities, constraints and preferences including quality of service, trust, and security; and (2) resource management, interoperation and evolution of the Virtual Organization.

1.1. *Problem statement*

Enterprise Application Integration (EAI) across a VO is difficult due to: (1) a multitude of semantics and protocols; (2) the need to propagate and synchronize rules; (3) constraints on elements; and (4) the temporal nature of a VO.[18] 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 VO. Present solutions require the hand-crafting of Web services and their interfaces, although recent literature[19] has begun to address these issues. 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, 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. To effectively differentiate dynamically among similar Web services requires additional information about the request, requestor, organization, scenario, and suppliers that are involved in providing the solution.
- There is a need to dynamically manage the enactment of workflow and transaction control.

The handling of these issues needs to be streamlined in order to deal with the heterogeneous and constantly changing environments within an enterprise, as well as the rapidly progressing base of standards and protocols that support Web services. Many integration solutions are rigid because they are developed for *ad hoc* and individual interfaces. Also, many vendor solutions are proprietary, and generally target only the needs of their general customer base rather than facilitating customization for specific customer needs.

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 address such issues as pricing, resource management, quality of service, scalability and delivery schedule.

1.2. *Issues with existing approaches*

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[5] 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,[20] WS-Security[21]) 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,[22] BPEL4WS, Business Process Modeling Language (BPML)[23]) 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.

As mentioned, Semantic Web Services attempt to address the shortcomings of Web services with respect to automation, and OWL-S is being established as a primary specification for this endeavor. However, recent research[24–26] has documented several shortcomings attributed to OWL-S. With respect to this research, the notable limitations are:

- **Conceptual ambiguity** in that the major concepts of OWL-S (service, grounding, and process) are still being clarified. For example, the concept of service is regarded as "any Web-accessible program/sensor/device".[13,25]
- **Narrow scope** because OWL-S also does not address fully the real-world issues[25] related to the VO. For example, OWL-S considers the service and the process aspects as the primary elements, but does not show how to specify federation, agent and data store aspects.
- **Loose design** means that OWL-S does not adequately differentiate the context of the service-related tasks such as discovery, composition and invocation.[25]
- **Coupled function and description** because OWL-S lacks an explicit and declarative decoupling between the functional features of a process (what) and the structural description of such processes (how).[24]

- **Static process declaration** means that OWL-S does not provide for the dynamic adjustment of an agent's Process model during the interaction.[26]

As seen in Ref. 27, to achieve an end-to-end solution involves linking the many disparate protocols and technologies, despite the fact that they are all still based on the XML foundation. Approaches such as Business Process Execution Language for Web Services (BPEL4WS),[28] Web Services Choreography Interface (WSCI)[29] and the World Wide Web Consortium's (W3C) WS-Choreography group[30] do not address the full life-cycle of activities to automate Web services. The nuances of these various technologies still need to be mediated to some level to be truly interoperable.[31] Additionally, these protocols do not address the aspects of specifying agents and their operations, designating knowledge repositories, and policies are just now coming into the literature. However, it will be interesting to see the progress of the recently-formed Web Services Modeling Ontology[32] initiative that is working towards the standardization and a common architecture and platform for Semantic Web Services.

The vision of ebXML (Electronic Business using eXtensible Markup Language) is "to create a single global electronic marketplace where businesses can find each other, agree to become trading partners, and conduct business".[33] ebXML describes itself as "a modular suite of specifications that enables enterprises of any size and in any geographical location to conduct business over the Internet", and allows "companies now have a standard method to exchange business messages, conduct trading relationships, communicate data in common terms and define and register business processes".[34] However, ebXML positions itself (unsuccessfully to date) as an alternative to the Web services architectural components of UDDI and WSDL.[33,35] However, there are aspects of ebXML that prove very useful to Web services and to this research with regard to business interaction standards. RosettaNet Partner Interface Processes® (PIPs®) also provide guidance for business interactions as well, but not necessarily with a Web service focus.[36]

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. Next, the management needs of Web services within a VO are presented.

The Knowledge-based Dynamic Semantic Web Services Framework is presented in Sec. 3 with a detailed discussion of its components. The framework is decomposed into three hierarchical layers consisting of Virtual Organization 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 an application 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 Organization

2.1. *Virtual organization 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.[37]

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.[18]

The next steps in the evolution of EAI and IEI are towards that of a VO. For Web services to perform a significant role in a VO environment, many issues need to be addressed to accommodate the disjointed, distributed, and temporal nature of the VO. A VO is a dynamic collection of individuals, institutions, and resources.[38]

As stated in Ref. 18, "The activities of the virtual enterprise cycle are accomplished by processes that are owned and operated by individual members of the VO 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 VO offers unique challenges because factions, herein referred to as partners, often compete and conflict with each other. Additionally, there often 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 cannot 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.*[39]

Grid computing (or the use of a computational grid) is often discussed in the context of Virtual Organizations, and involves the provisioning of resources encapsulated behind a service-oriented view to insulate the client from the burden of knowing which resources actually fulfill the request or where the resources are located.[38,40] 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, files, data, and other resources, as is required by a range of collaborative problem-solving and resource-brokering strategies emerging in industry, science, and engineering. 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 is called a Virtual Organization.[38]

Clients can make requests for grid resources without having to know which precise resource within the grid services the request. Recent research initiatives in the Grid community provide increasing levels of sophistication. Some enabling technologies, such as ontologies and reasoning, knowledge management, knowledge discovery and agent grids, are currently offered by the depicted layers, but their main impact will be really evident when they will be used internally to enhance Grid management and operation. It is predicted that peer-to-peer will be the orthogonal technology on which main tasks such as presence management, resource discovery and sharing, collaboration and self-configuration will be based.[41,42]

Regarding the Open Grid Services Architecture (OGSA) discussed in Ref. 40, the term Grid service is a "(potentially transient) stateful service instance supporting reliable and secure invocation (when required), lifetime management, notification, policy management, credential management, and virtualization", where virtualize means that "resources at multiple levels, so that the same abstractions and mechanisms can be used both within distributed Grids supporting collaboration across organizational domains and within hosting environments spanning multiple tiers within a single IT domain".

The WS-Resource Framework (WSRF)[43] is being introduced as converging Grid and Web services in order to restate OGSI concepts in terms of Web services. Three main concerns noted about the OGSI are that it places "too much stuff in one specification", it does not work well with existing Web services tooling, and is too object-oriented. Czajkowski[44] explains the five technical specifications that define the normative description of the WS-Resource approach in terms of specific Web services message exchanges and related XML definitions. These technical specifications define the means by which:

- a WS-Resource can be destroyed, either synchronously with respect to a destroy request or through a mechanism offering time-based (scheduled) destruction, and

specified resource properties [WS-ResourceProperties] may be used to inspect and monitor the lifetime of a WS-Resource (WSResourceLifetime);
- the type definition of a WS-Resource can be composed from the interface description of a Web service and an XML resource properties document, and the WS-Resource's state can be queried and modified via Web services message exchanges (WS-ResourceProperties);
- a Web service endpoint reference (WS-Addressing) can be renewed in the event the addressing or policy information contained within it becomes invalid or stale (WS-RenewableReferences);
- heterogeneous by-reference collections of Web services can be defined, whether or not the services are WS-Resources (WS-ServiceGroups);
- fault reporting can be made more standardized through use of an XML Schema type for base faults and rules for how this base fault type is used and extended by Web services (WS-BaseFaults).

2.3. *Web services management*

Interoperability is the central issue for a VO, 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).[38] Users make requests to systems that require many layers of behind-the-scenes processing be managed to service users' needs. The management levels within a VO are strategic (infrastructure), asset (resource management), and value-chain (processes).[45] The issues that exist with Web services operating across a VO 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 versus 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 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 structure for 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.

The W3C's Management Model focuses on those aspects of the Web services architecture related 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.[46] The protocol indicated for using Web services to manage distributed resources is the Web Services Distributed Management (WSDM).[47] 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[48,49]; 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 introduces the Knowledge-based Dynamic Semantic Web Services (KDSWS) Framework[50] to deliver a comprehensive and integrated end-to-end solution to dynamically prepare, publish, request, discover, select, configure, deploy and deliver Semantic Web Services (SWS). The framework positions itself as an enterprise-scale foundation to ultimately provide VOs with the interfaces necessary to interoperate via Semantic Web Services using, and quickly adapt to, the plethora of prevailing Semantic Web technologies and protocols. Thus, this approach considers facets that are critical to the success of any organization (i.e. policies have to be set, rules have to be established, etc.)

Like the Web Services Modeling Framework (WSMF),[51] ontologies and mediation are an integral part of the KDSWS, and are manifested in the mappings and heuristics. The structure and design of the components target, or map to, ontological implementations such as OWL and OWL-S, albeit with potential extensions to facilitate the additional functionality. A primary function of the heuristics is to enhance the mappings ability to dynamically mediate between disparate components.

This framework has similar guiding principles, with respect to incorporating knowledge and providing a "protocol-independent" (versus language-independent) conceptual base, to those found in Ref. 24. This framework also has similarities with the Conceptual Architecture using ebXML described in Ref. 35 with respect to incorporating policies, agreements, and mechanisms for discovery and composition. However, the KDSWS approaches the solution space from a different perspective by placing more emphasis on a total enterprise solution (by specifying agent and storage needs as well) and by providing a "way-forward" via a guiding methodology as to how to use the structures. To this end, the KDSWS Framework is an enabling mechanism and foundation for implementations. See Sec. 5 to create an implementation for this framework.

This research also positions itself as a meta-design specification to focus on capturing the knowledge and intended purpose of the elements and their interactions. Having a meta-approach to the research, some items are left at an abstract level. Although mappings to detail implementations are presented, as with any design endeavor, it is assumed that the structures presented here will require implementation-level details to be developed to realize the actual benefit.

The KDL and KDSPL languages are semantic language; thus, they dispense with low-level details in order to capture this knowledge and intent more effectively. The low-level details such as exact algorithms, data structures, class definition are left to additional software and procedures that can easily be inferred, and is beyond the scope of this research provide the implementation-level of specification. Also, specifying at this semantic level is easier than working with XML which tends to be too verbose to work with manually.

The specification is presented at multiple levels, and the power of a knowledge-based approach is that these elements are presented in an ontological paradigm where the proper relations can be easily inferred. Some of the meta-model elements take on a role as both KDL and KDSPL elements in order to convey both the data and process aspects on the structures.

It is important to realize that the support of a Virtual Organization requires more overhead than that of a simple and defined interface between applications. The support of dynamic operations between partners requires additional support and maintenance, as does any enterprise system, to perpetuate effective and efficient functional components. Although the upfront work typically puts new entrants into the VO at somewhat of a disadvantage, it is the intent of this research to enable new partners to know what is expected and reduce the barrier to entry in order to participate within the VO.

As shown in Fig. 1, the framework consists of the KDSWS Processes, KDSWS Design Specification, and KDSWS Functional Architecture (note that underlines denote framework components typically, but not always, specified in the diagrams). Three layers are proposed because they correspond to the Requirements (i.e. Processes dictate what has to be done), Design (i.e. Specifications dictate how it is to be done), and Implementation (i.e. the Functional Architecture realizes the design specification) phases in most development efforts. The KDSWS Processes layer is segregated into Tasks and Threads; such segregation addresses the "loose design" issue discussed earlier by providing the ability to specify behavior based on the context in which an operation is invoked. The specification languages have been crafted to incorporate the notion of context.

Tasks are well-delineated steps to deliver functionality via Web services, and the KDSWS proposes the following collection of tasks for the Web services life-cycle: Prepare for Publish, Publish, Prepare for Request, Request, Discover, Select, Configure, Deploy, Deliver, Retire, and Interface. For additional context

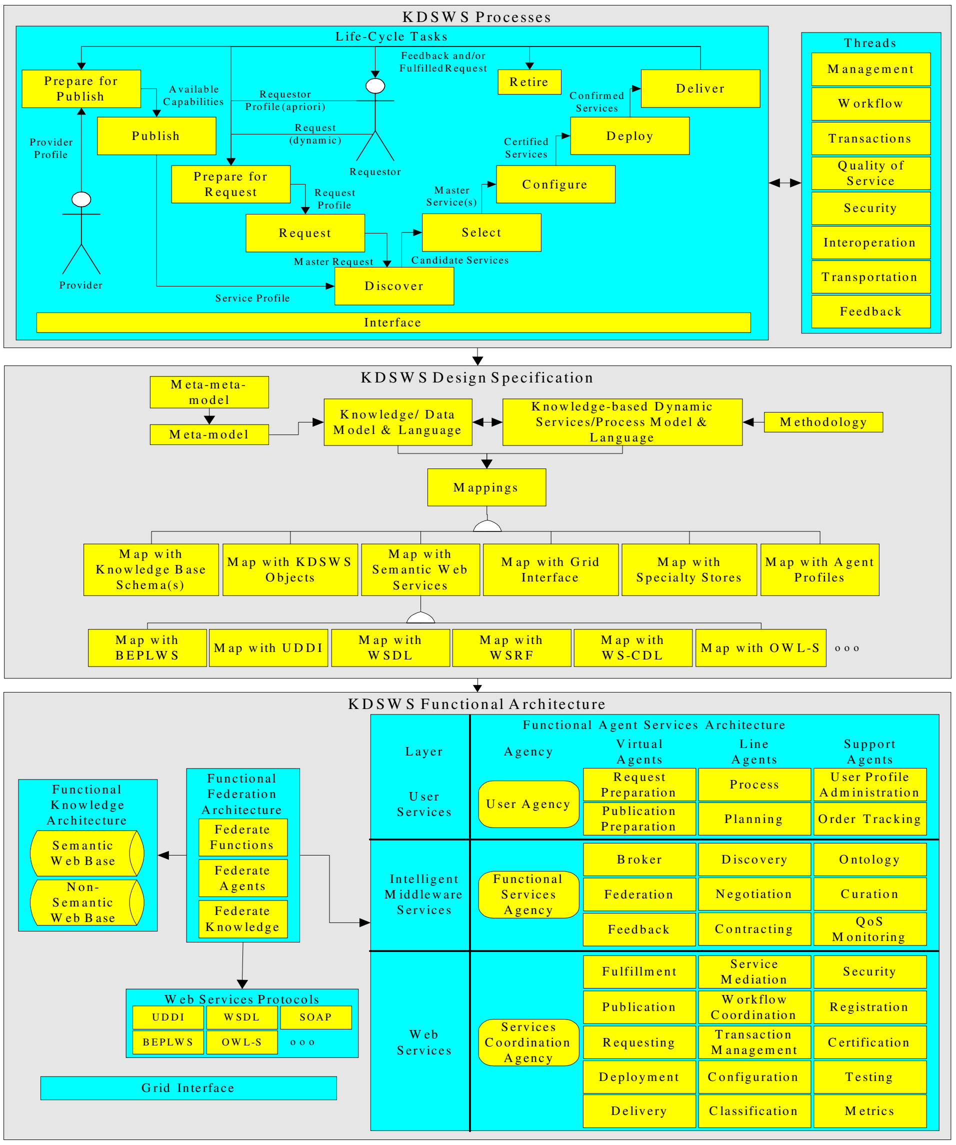

Fig. 1. KDSWS framework.

designation, threads are defined as 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. This context is carried into the specification to enable behavior to be tailored.

The KDSWS Design Specification addresses the "narrow scope" issue discussed earlier by allowing the enterprise-wide data and processes needs to be stipulated for the KDSWS Processes, as well as for the backend, middleware and user services, which provides a comprehensive (by addressing these backend, middleware, and user needs as well as the end-to-end issues) and integrated (by addressing both data and process needs) solution. The data aspects are captured in the meta-model and are modeled by an adapted version of the Knowledge/Data Model and Language (KDM/KDL).[52] The process aspects are captured in the methodology and are modeled by the Knowledge-based Dynamic Services/Process Model and Language (KDSPM/L). This separation addresses the "coupled function and description" issue addressed earlier with regards to the shortcomings of OWL-S. This "meta-model/process"-based approach enables an extensible framework that provides a macro-level shell to plug-in different approaches to the facets of the Web services life-cycle (e.g. negotiation, policy management, etc.) yet still provides enough structure to guide implementations.

The specification can be carried from the top-level methodology down to the instantiation level that provides a consistent chain from the design down to the implementation of the specification. The KDSWS Specifications transition into components within the KDSWS Functional Architecture as explained in Sec. 3.2.4.

The KDSWS Functional Architecture (KDS-FA) 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 via such items as the :GOALS primitive.

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,[27,46,53] but this approach separates the activities within the life-cycle of Web services from the functional perspective to provide a crisper set of concepts.

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 both atomic and composite 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 correlate 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 faster 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 confirms that the resources and the environment are ready to produce "Confirmed Services", and the "Fulfillment Package" is created. The Deliver process executes the Web services, enacts the workflow, and delivers services 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, and prevent the requestors from brokering obsolete services. The Retire process involves establishing the criteria for Web service retirement, and using the feedback mechanisms to evaluate the results against the criteria.

The Interface task fits into a different category than the life-cycle tasks previously discussed because it occurs primarily in the front-end of the life-cycle; however, it can occur throughout the stages. The Interface task handles the import and exporting of knowledge.

The Management layer directs the activities of the other threads. Workflow involves the management of the steps to achieve the goals of 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[54] are maintained. Security provides for authentication, integrity, privacy and non-repudiation. Interoperation ensures that the service integrates with other services and protocols. Transportation ensures that messages are handled properly. 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 to ensure the elements are consistent both vertically and horizontally. Similar to OIL and OWL, the specification has three levels of capabilities; the levels are named Lite, Standard and Full. This allows 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.

As mentioned, the meta-model is specified via an adaptation of the KDM/KDL,[52,55] while the methodology is specified by the new, albeit extension of KDM/KDML, (KDSPM/KDSPL). 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. As a note, the items prefixed with "kdsd" denote KDL objects involved in the flow, and items prefixed with "kdsp" denote KDSPL objects used in the process.

3.2.1. *KDSWS meta-models*

Similar to the Meta-Object-Facility (MOF),[56] this meta-model also establishes four levels of metadata architecture — the meta-meta-model, meta-model, model, and information. The meta-meta-model shown in Fig. 2 displays the links relating the RDF/RDFS and OWL data types (because OWL uses the RDF Schema data structures).[12] Some of the "attributes" in the meta-model in Figs. 3 and 4 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 WSMF, OWL-S, Sheth[57] and Web Services Architecture Usage Scenarios,[58] and they represent the essential concepts that are necessary to model the functions of Semantic Web Services. The meta-model is the base for data objects that will be infused throughout the specification. The semantics used within the specification may directly from the structures in the meta-model, or may be derived via rules in the heuristics.

The meta-model focuses on developing the "building blocks" (i.e. constraints, preferences, profiles, capabilities, etc.) versus focusing on the higher-level and visible, finished products that are aggregated (e.g. a service, a profile, or request) on which some approaches place their primary attention (i.e. "conceptual ambiguity", as it relates to OWL-S shortcomings mentioned in the Introduction, is the result). This "building block" approach makes the framework elements composable, reusable and extensible. The meta-model distinctly defines process-based (e.g. task, thread, event as shown in Fig. 3) and resource-based (e.g. service, agent, protocol as shown in Fig. 4) classes in order to clarify the concepts involved in delivering Web services functionality. The lower level "building blocks" can easily be aggregated to compose the higher-level elements. By making entities such as constraints and preferences a first-level super-class, the benefits of generalization/specialization are achieved for these components instead of being buried within attribute specifications. In the case of a composite service (one that calls other services), the provider

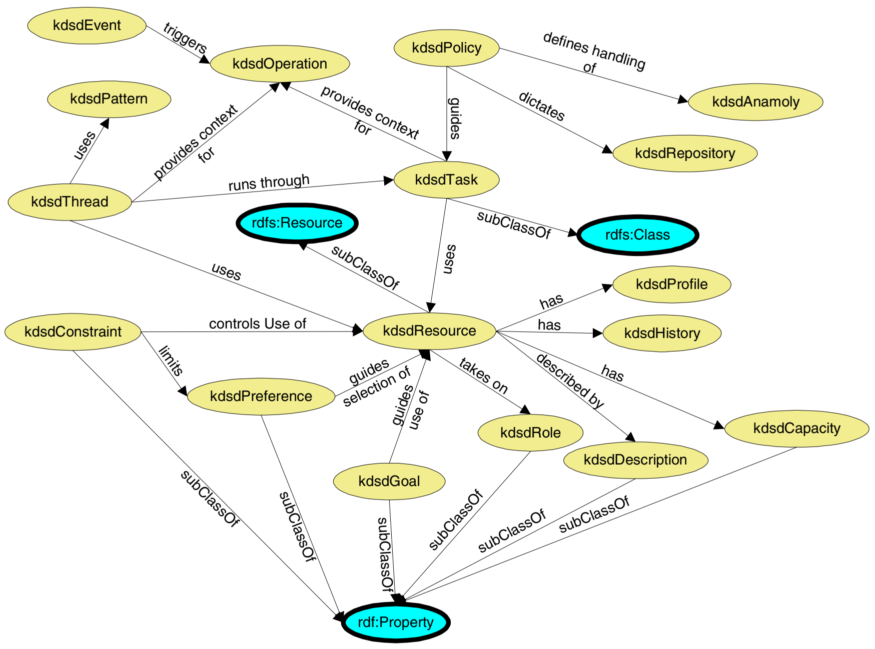

Fig. 2. Meta-meta-model.

is also a requestor in the same execution stream. The super-class level functions handle the constraints or preferences that are common to both the requestor/provider, while the specialized functions are used as the service transitions between roles (i.e. requestor and provider). By creating the building blocks at the meta-model level, it enables new components to be incorporated much more easily.

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 super-class for kdsdException, kdsdContraintViolation and kdsdError. kdsdEvent defines occurrences that should trigger some processing to commence, and two possible subclasses of kdsdEvent are kdsdSubscription and kdsdSensor.

kdsdTask is the super-class for kdsdPrepare, kdsdPublish, kdsdRequest, kdsdDiscover, kdsdSelect, kdsdConfigure, kdsdDeploy, kdsdDeliver, kdsdRetire, and kdsdInterface. kdsdPrepare is combined with both the kdsdPublish and kdsdRequest tasks because both tasks have a preparation task. kdsdDiscover, kdsdSelect and kdsdConfigure aggregate to kdsdBroker. 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.

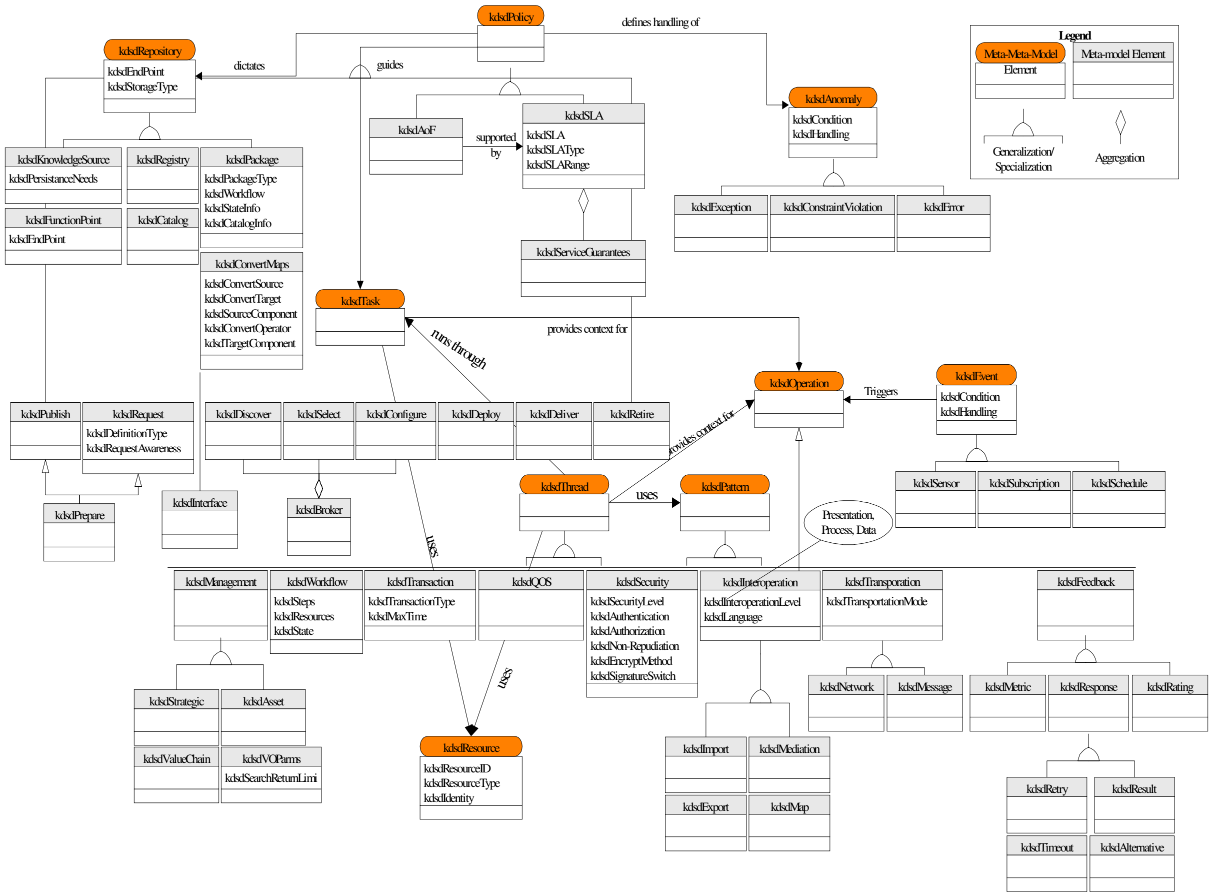

Fig. 3. KDSWS process-based meta-model classes.

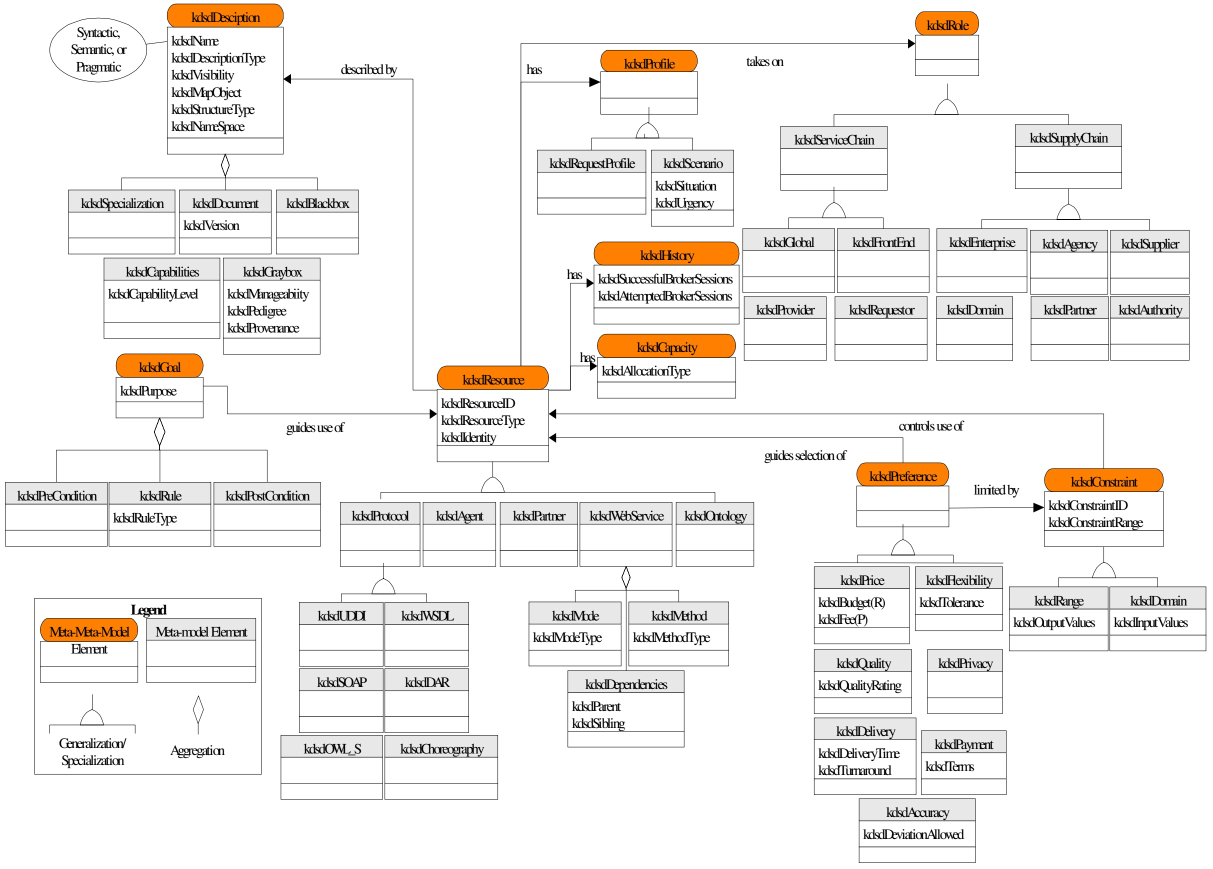

Fig. 4. KDSWS resource-based meta-model classes.

kdsdThread and kdsdPattern are both super-classes for the thread objects because threads use well-established patterns. kdsdManagement is organized by the three management levels associated with Virtual Organizations — kdsdStrategic, kdsdAsset, or kdsdValue-Chain. Another subclass of kdsdManagement is kdsdVOParms to support parameters for the Virtual Organization. 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 kdsdMediation, import needs under kdsdImport, export needs under kdsdExport, and mapping between objects under kdsdMap. 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. kdsdTransportationMode indicates which general networks are available such as extranet, intranet and internet. 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 super-class structure for descriptive properties. kdsdVisibility is used to determine where a component is made available to see or use. 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" with respect to these properties. Contrasted with kdsdBlackbox properties, kdsdGraybox properties are properties that might be useful for requestors to see about the provider. kdsdManageability is a key property that falls into this category.

kdsdCapabilities is a very diverse and pivotal class that is a super-class 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.)[59] as well as syntax and protocol. One particular attribute of interest is the kdsdCapabilityLevel which describes whether the resource participates at a kdsdLite, kdsdStandard or kdsdFull level. kdsdSpecialization conveys the primary focus of the object such as negotiation or coordination as in the case of agents.

kdsdHistory stores the past activity of a resource, while kdsdCapacity conveys the availability of resources in the form of CapacityType (DiscreteCapacity or ContinuousCapacity). kdsdAllocationType conveys whether an allocation is consumable or reusable. kdsdProfile is an aggregation of the properties and objects within the framework. One such profile is kdsdRequestProfile profiles requests so that agents can bin requests as they arrive. kdsdScenario depicts the situation behind the request. Scenarios are coming into the literature as of late, and an example of a hurricane scenario is demonstrated in Sec. 4.

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 (i.e. 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.

3.2.2. *KDSWS methodology*

The top-level processes, as shown in Fig. 5, correlate directly to the Tasks identified in the framework — Prepare (Publish and Request are consolidated for presentation efficiency), Publish, Request, Discover, Select, Configure, Deploy and Deliver. 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 makes use of the meta-model elements.

The diagram delimits the tasks (e.g. Prepare, Publish, Request, etc.) by the vertical separator lines. The methodology also shows the processes within the context of the KDSWS Threads. Notice that the majority of activity takes place in the management thread where policies can be monitored throughout the life-cycle of the delivery. Specialized management is distributed throughout the threads as needed. Also notice that the bulk of the process reside at either the front and back of the life-cycle, which depicts a loosely-coupled asynchronous environment.

The flow of the KDSPL (process) objects is identified via the solid lines, while the dashed not-so-solid lines indicate specialized processes occurring with the parent process to which it is connected. In order to provide a concise view of the overall methodology, a general chronology of events is assumed, and the temporal and cyclical facets of the life-cycle are discussed rather than diagramed. As the

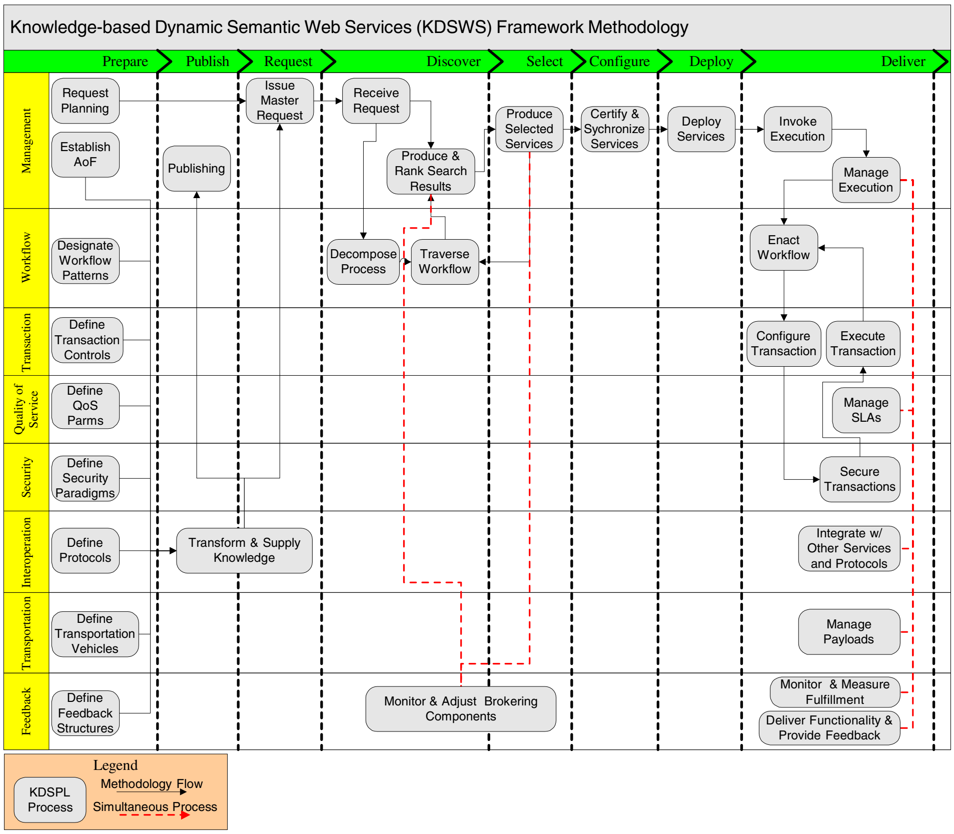

Fig. 5. KDSWS top-level methodology.

KDSWS Meta-model discussion references this section to further explain context and use, this discussion may require referencing back to the "definition" presented in the KDSWS Meta-model discussion. Also, it is suggested that the reader take a look at Sec. 5 to understand some of the implementation mechanics of the components. As a note, the "s" for plural forms of the singular-based objects are purposely not underlined to distinguish the singular-based objects from the plural-based objects.

The Establish AoF process sets up the policies, rules and catalogs for the VO. Policies are implemented as rules, so the heuristics on the kdsdPolicy objects are extensible. The catalogs are references for what is deemed as an "offering", and can be designated as "available" at the enterprise level, local level or both. The designation is made by making kdsdDescription an attribute (with a type of "object") of the kdsdResource (superclass of kdsdCatalog) and setting the kdsdVisibility attribute to the appropriate value.

Since the Prepare task primarily identifies what is available, the primary output is a map of name-value pairs of those offerings. From the Prepare for Publish perspective, these maps reflect what is valid to offer; whereas, the Prepare for Request reflects what is valid to use.

The Designate Workflow Patterns process produces a map of the available kdsdScenario objects and their associated kdsdWorkflow kdsdPatterns. The Define Transaction Controls process creates a map of the available kdsdTransactionTypes and their controls (e.g. Max Time). The Define QoS Parms process creates a map of available kdsdSLAType (e.g. blankets) with the associated kdsdServiceGuarantees (e.g. delivery time) and kdsdServiceLevelRanges, where kdsdServiceLevelRanges reflects the minimum and/or maximum values of that service type. The Define Security Paradigms process creates a map of available kdsdSecurityParadigms (e.g. Partner versus guest access) and the associated kdsdProtocols that support each paradigm. The Define Protocols process creates a map of the available kdsdOperations and their supported kdsdProtocols. The Define Transportation Vehicles process creates a map of the available operations with their supported kdsdTransportationModes (e.g. extranet, intranet and internet). The Define Feedback Structures process defines kdsdStatusTemplates, kdsdMetricGoals, kdsdRatingStructures and kdsdResultsTemplates, which are not shown in the meta-model. These exemplify classes under kdsdProfile that provide infrastructure support for the optimization of a system.

The Transform & Supply Knowledge process handles transforming and supplying knowledge (i.e. knowledge also includes data and information) into a semantic form. The knowledge may be stored in a semantic form (e.g. XML databases that can support the storage of OWL-like structures), or may require mediation to convert knowledge from a non-semantic into a semantic form for use. This process is used both for the Publish and Request activities.

The Publishing process assigns the available resources to the respective business and service information in the appropriate registries (i.e. UDDI as designated by a subclass of kdsdRegistry) and WSDLs. This research envisions that domain-specific registries are required to support this automation in conjunction with domain-specific ontologies. The context of a domain is vital to correctly interpret and match the semantics.

These next processes commence the request processing chain of processes versus the publishing chain just discussed. The Request Planning process involves handling kdsdSubscriptions and creating the profiles for the request (in the form of kdsdRequestProfiles) and the requestor's environment so that turnaround time can be reduced by binning the request into a prescribed course of action. kdsdSubscriptions is placed under events because their purpose is to notify subscribers when certain events occur (e.g. product to be distributed or news).

The Issue Master Request process collects the kdsdRequest, kdsdConstraints, kdsdPreferences, and kdsdSearchPriorities to form the kdsdMasterRequest. kdsdMasterRequest is a dynamically aggregated object where rules are applied as how to form the object. One way this is accomplished is by defining a TARGET in a KDSPL process object (see next section) where the process's purpose is to create an object by invoking the rules in the process object. Another way is to specify an attribute type of "process" that will direct an agent to execute a process to instantiate the object. Yet another method is to dynamically map an object is to specify the mapping in the rules for the object as it is instantiated. The last method is explained in Sec. 5 by adding the kdsdMapObject attribute.

The Receive Request process senses that a request has been submitted either by subscription or submission, and determines if the request requires brokering or it is specified.

The iterative nature of this methodology commences with the Decompose Process process where the process specification is mapped into the workflow. The Traverse Workflow process iterates through workflow to execute the steps specified in the process or kdsdWorkPattern. The Configuration Package, or configuration plan as mentioned below, is built up as this loop is completed.

The brokering activities occur primarily in the Produce & Rank Search Results and Produce Selected Services processes. Current research suggests syntactic, semantic and pragmatic brokering types, or semiotic brokering to reflect all three. Syntactic brokering uses the structure or format of a task specification to match a requester with a service provider, semantic brokering uses the request's meaning and information content to match with the meaning of the offered services of the provider,[60] and pragmatic,[61] involves using the context of the interaction to broker services.

The discovery of available services depends on search capabilities that involve the semantic specification of constraints and preferences of the requestor to match with the same semantic specification of the provider and the provider's services. We cite KnowledgeSifter[62] as such as a semiotically-enabled brokering facility in that the traditional syntactic search engines are used, ontologies are used for semantic meaning, and the ontologies are domain-specific to set context for the pragmatic criteria. By facilitating the semiotic brokering about the resource that fulfills the request, the client can make decisions as to whether the result is acceptable or not, and measure satisfaction with the resource if it chooses to use it for future requests. A key enabler of this measurement and "cognition of success" for future use will be Knowledge Discovery and Data Mining techniques.

Another key enabler along these lines is to develop organizational ontologies that can be complemented with personal ontologies created by users over the course of their decision-making and investigations about a service's result(s). These reflect personal preferences regarding: (1) how concepts are related and organized, (2) preferred search engines, and (3) opinions regarding the authoritativeness of a source and the accuracy of its information. These preferences can be used to rank, sift and winnow the results returned.

The search results are ranked according to the kdsdSearchPriorities. The Produce Selected Services process differentiates the services using kdsdSLA and kdsdQoS parameters with the track records of the providers and services in the respective subclasses of kdsdHistory, negotiates using kdsdNegotiationPolicy and kdsdSelectionPolicy, and ultimately selects the services and any alternatives that may have been identified. The Monitor & Adjust Brokering Components process optimizes resources such as kdsdSearchRequestProfiles and kdsdAgentProfiles, kdsdSearchAgentCapabilities, and kdsdNegotiationPerformance (a subclass of kdsdHistory) that are involved in the brokering activity. The Certify & Synchronize Services process validates and verifies the resultant services, and then merges and synchronizes them with the workflow to create an optimum execution plan (e.g. time and sequence a given provider's services to execute together).

At this juncture, these processes transition from an emphasis on brokering to delivering the services' functionality or product. The Deploy Services process applies the Articles of Federation embodied in kdsdAoF, coordinates resources, creates the Fulfillment Package using information from the Configuration Package, checks and opens ports (in the case of critical endpoints), establishes time and security boundaries from the kdsdTransaction and kdsdSecurity objects, profiles the environment, and captures the profile of the request.

The Invoke Execution process commences the execution cycle. The Manage Execution process is a supervisory process over the subsequent delivery steps, which handles the issues concerning the threads via the simultaneous (i.e. simultaneous because they execute parallel with the parent to handle specialized functionality) processes that will now be discussed. The Manage SLA simultaneous-process enforces the Service Level Agreements (SLA), while the Handle Anomalies simultaneous-process deals with such situations as errors, exceptions and constraint violations. The Manage Limits, Tolerances and Workload simultaneous-process manages the Quality of Service aspects. The Integrate w/Other Services and Protocols simultaneous-process coordinates and mediates the interactions with the interfaces of surrounding disparate processes. The Manage Payloads simultaneous-process manages the messaging needs. The Monitor & Measure Fulfillment simultaneous-process supports the metrics needs of the enterprise, and ensures that the interactions with the requestor, albeit human or machine, effectively to keep the workflow proceeding. The Deliver Functionality & Provide Feedback simultaneous-process determines whether products need to be delivered, notifications sent, or responses sent to requestors immediately as the individual services are executed or packaged and held until the entire workflow is complete.

The Enact Workflow process first determines if workflow is involved. If not, the flow proceeds to Secure Transaction process; otherwise, the workflow is once again managed and iterated through. The Configure Transaction process groups and sequences the transactions together in order to optimize the execution. The Secure Transaction process authenticates and authorizes the request and requestor, validates the activity, and encrypts the messages if necessary. The Execute Transaction process executes the transaction group, and then returns to the Enact Workflow process for subsequent steps.

3.2.3. *KDSWS specification languages*

In order to model the framework's data-centric concepts, we use an extended version of the Knowledge/Data Model (KDM).[52] 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,[63] 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 most generic construct in the KDM is the object type and is specified in the Knowledge/Data Language (KDL) template depicted in Fig. 6, which shows a general template for an object-type (class) specification employing the KDL.[64] The KDL and KDSPL's 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 using the extended BNF grammar.[65] 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. 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.
- 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

```
object-type ::=
        OBJECT_TYPE class-name HAS
                [ DESCRIPTION: string;]
                [ GOALS: class-name { , class-name };]
                [ SUPERTYPES: // Generalization
                        class-name { , class-name };]
                [ SUBTYPES: // Specialization
                        class-name { , class-name } [ HIDING function-list ];]
                [ ATTRIBUTES: // Aggregation
                        { attribute-name: type-name
                                [ SET OF | LIST OF | MAP OF ] value-type [, value-type
                                [ WITH CONSTRAINT: constraint ];}+]
                [ MEMBERS: // Membership (Association)
                        { member-name: [ SET OF | LIST OF ] class-name
                                [ INVERSE OF member-name [(class-name)]]
                                [ WITH CONSTRAINT: constraint ];}+]
                [ APPLICABILITY: (METHODOLOGY | GLOBAL | LOCAL);]
                [ AGENCIES:
                        { class-name:
                        AGENCY-ROLE: string
                        POINT-OF-CONTACT: string
                        POINT-OF-CONTACT-ROLE: string;}+]
                [ DISTRIBUTION:
                        {END-POINT: string
                        DISTRIBUTION-ROLE: class-name
                        DISTRIBUTION-SCHEDULE: schedule;}+]
                [ CONSTRAINTS: // Knowledge to enforce integrity
                        CONSTRAINT-ID: constraint-ID
                        CONSTRAINT-CATEGORIES: class-name { , class-name; }]
                        constraint { , constraint; };]]
                [ PREFERENCES: // Knowledge to reflect preferences
                        PREFERENCE-ID: preference-ID
                        PREFERENCE-CATEGORIES: class-name { , class-name; }
                        preference { , preference; };]
                [ HEURISTICS: // Knowledge to derive/infer information
                        HEURISTIC-ID: heuristic-ID
                        HEURISTIC-CATEGORIES: class-name { class-name; }
                        rule { , rule; };]
                [ METHODS: // Specifications of computations and behavior
                        METHOD-ID
                        { method; };]
END class-name;
```

Fig. 6. Syntax of KDL object-type specification.

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.[66]
- **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 extensions to the KDL introduced in this research are:

- **Description:** DESCRIPTION allows the meaning to be specified in free-form verbiage.
- **Goals:** GOALS is taken from the WSMF to capture the purpose of the objects.
- **Applicability:** APPLICABILITY level specifies whether security elements pertain to the methodology, global to the enterprise, or local to the agencies or partners.
- **Agencies:** AGENCIES specifies the partner(s) to which the object applies in the role specified in the AGENCY-ROLE (e.g. user agency, supplier). POINT-OF-CONTACT specifies the contact point in the role specified in the POINT-OF-CONTACT-ROLE element.
- **Distribution:** DISTRIBUTION dictates where to distribute policies and resources to the enterprise in the role specified in the DISTRIBUTION-ROLE element. DISTRIBUTION-SCHEDULE specifies when to distribute the resources.
- **Preferences:** This primitive is used like constraints, and have the same construct as constraints. Preferences convey desires of an entity, but are weaker than constraints.
- **Constraint-, Preference-, and Heuristic-ID:** These IDs are assigned to these primitives in order to facilitate re-use throughout the specification base.

- **Constraint-, Preference-, and Heuristic-Categories:** These CATEGORIES provide the ability to group the constraints and rules together such as security and messaging.

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 Fig. 7. Although the target of the KDSPM/KDSPL is used to specify SWS, 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. Some of the new primitives that the KDSPM introduces are discussed below:

- **Owner:** The OWNER specifies the primary agent responsible for the process.
- **Steward:** The STEWARD argument specifies the secondary agent that may handle the process.
- **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.
- **Targets:** The TARGETS argument specifies the resultant entity (i.e. a particular agent or object) for which the object is intended in the specified TARGET-ROLE.
- **Steps:** The STEPS argument details the actions involved in executing the process. The STEPNAME and STEP-DESCRIPTION annotate the step. The SEQUENCE-NUMBER denotes the chronological occurrence of the step. DELEGATE specifies what object executes the step, and because object take on various forms, DELEGATE-TYPE and DELEGATE-ROLE specify the type and role properties that the object assumes under this step. The OPERATION argument specifies that another KDSPL PROCESS-OBJECT, METHOD-NAME or MANUAL-INTERVENTION performs the step. RESOURCES specify which resources are necessary to support the step's execution. The STEP-SUCCESSORS information contains the STEP-SUCCESSOR-MODE that specifies the manner (i.e. parallel, sequential, interval, etc.) in which the next step is to execute the next step in STEP-SUCCESSOR-BRANCH under the specified STEP-CONTROL-CONDITION.

3.2.4. *KDSWS mapping*

The Mappings is a super-class 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 and a destination along with metadata for each to denote specific details to take into consideration in the mapping. For example, the first-level super-class of mappings holds the basic

```
object-type ::=
        OBJECT_TYPE class-name HAS
                [ DESCRIPTION: string;]
                [ GOALS: class-name { , class-name };]
                [ TASK: class-name { , class-name};]
                [ THREAD: class-name{ , class-name};]
                [ SUPERTYPES: // Generalization
                        class-name { , class-name };]
                [ SUBTYPES: // Specialization
                        class-name { , class-name } [ HIDING function-list ];]
                [ OWNER: class-name ;]
                [ STEWARD: class-name;]
                [ APPLICABILITY: (METHODOLOGY | GLOBAL | LOCAL);]
                [ AGENCIES:
                        { class-name:
                        AGENCY-ROLE: class-name
                        POINT-OF-CONTACT: string
                        POINT-OF-CONTACT-ROLE: class-name;}+]
                [ DISTRIBUTION: { end-point
                        DISTRIBUTION-ROLE: class-name
                        DISTRIBUTION-SCHEDULE: schedule;}+]
                [ PREDECESSORS: class-name{ , class-name };]
                [ SUCCESSORS: class-name{ , class-name };]
                [ TARGETS: {class-name
                        TARGET-ROLE: class-name;}+]
                [ TRIGGERS: (event-name | external-event-name);]
                [ STEPS:    {STEPNAME: step-name;
                            SEQUENCE-NUMBER: sequence-number;
                            STEP-DESCRIPTION: step-description;
                            [DELEGATE: class-name;
                                DELEGATE-TYPE: (AGENT | WEB SERVICE | HUMAN);
                                [DELEGATE-ROLE: (VIRTUAL | LINE | STAFF | GRID | PRIMARY | SECONDARY)];]
                            [OPERATION: class-name;
                                METHOD-NAME: method-name;
                                [MANUAL-INTERVENTION: manual-intervention;]
                                [PROCESS-OBJECT: class-name;]]
                            [RESOURCES: class-name{ , class-name };]
                            [STEP-SUCCESSORS:
                                STEP-SUCCESSOR-MODE: (PARALLEL | SEQUENTIAL |
                                        INTERVAL | LOOP | WHILE | CONDITIONAL | DECISION
                                        SIMULTANEOUS);
                                STEP-SUCCESSOR-BRANCH:  step-control-branch;
                                [STEP-CONTROL-CONDITION: step-control-condition];}+]
                [ CONSTRAINTS: // Knowledge to enforce integrity
                        CONSTRAINT-ID: constraint-ID
                        CONSTRAINT-CATEGORIES: class-name { , class-name; }
                        constraint { , constraint; };]
                [ PREFERENCES: // Knowledge to reflect preferences
                        PREFERENCE-ID: preference-ID
                        PREFERENCE-CATEGORIES: class-name { , class-name; }
                        preference { , preference; };]
                [ HEURISTICS: // Knowledge to derive/infer information
                        HEURISTIC-ID: heuristic-ID
                        HEURISTIC-CATEGORIES: class-name { class-name; }
                        rule { , rule; };]

END class-name;
```

Fig. 7. Syntax of KDSPL object-type specification.

structures, while a specific protocol can be specified for the specific mappings relative to that protocol. The mappings fall into four classes: internal for mapping within the KDSWS Specifications, export for mapping KDSWS Specifications to external components, import for mapping external components into the KDSWS Specifications, and interactive mappings that are bi-directional. It is important to note that this pertains to the KDSWS Specification, so elements within the KDSWS Functional Architecture are considered external components as well.

The internal mapping to other KDSWS objects (both KDL and KDSPL) enables the aggregation of the "building blocks" to be defined by rules versus relationships. Because the rules are much easier to change than hard-coded logic, the framework is much more flexible in adapting to change. This rules-based approach facilitates the dynamic generation of the framework's profiles (e.g. requestors, providers, situations, services, etc.) in order to drive the matchmaking of a request to service(s). By combining the rules with events, the framework allows the dynamic adjustment of the process to address the "static process declaration" issue by having certain conditions invoke specific behavior.

The interactive mapping to, and from, SWS technologies is achieved at two levels: globally and local 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 interoperation with specialty engines such as Workflow Management Systems (WfMS), Expert Systems, or Ontology Engines enhance the capabilities of this framework by providing mappings to these specialized tools. This mapping can be either be an import or interactive class, but the specialty data is generally imported to allow the specialty engine "specialize" and be the master of its data.

The profiles of the various agents are embedded throughout the KDSPL. This interactive mapping with 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. Although beyond the scope of this discussion, the interactive category is derived from the agent updating its own specification via learning during its activity.

The export mapping from the KDL to database schemas and object-oriented classes map to classes or tables, attributes map directly to class attributes or database attributes, and methods map to methods. The heuristic rules map to rules in active databases or applications.

3.3. *KDSWS functional architecture*

This research introduces the concept of packages, three in particular — "Configuration Package" (CP) and a "Fulfillment Package" (FP), and "Enactment Package" (EP). Because the loosely-coupled and asynchronous environment cannot depend on a session or given database to maintain state, these packages are necessary to maintain the state and distribute information for the process. A CP is used during the brokering activities in order to collect and configure the results of the selected services, where a FP is used during the execution of the Web services. An EP is a specialized package used for workflow steps to manage the workflow. The CP and FP distribute policies and workflow steps, convey information respective to their focus areas, and convey state of a workflow enactment in order to facilitate the maintenance of the organization's infrastructure. In order to distribute and share rules, the approach is to identify catalogs and their associated metadata. Metadata attributes such as version and last update date are 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. The metadata fields are included (as opposed to the actual rules) in order to reduce the payload. To reduce the potentially long-running transactions' dependence on network connectivity and resources, the packages also carry the workflow steps with the associated controls and state of the workflow.

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 items that can be automated are embodied in the heuristics. 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. 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 VO.

In the Federate Functions process, the VO handles the strategic layer of management. This is where the VO 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 VO, such as how a partner joins or leaves the VO, confidentiality, and which resources made available only within the organization and which are made available beyond the partners.

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

3.3.2. *Functional agent services architecture*

The agent architecture used in the KDSWS Architecture is the Functional Agent Services Architecture (FASA)[67] that is based on an agency-based approach in which agents are organized in agencies to further establish the context of where operation is invoked. 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.[68] The FASA consists of three layers — User, Intelligent Middleware and Web Services. This three-layer architecture contains virtual agents that specify agents' aggregation and coordinate the activities of other agents, line agents are involved in work specifications, and support agents that line agents use for generic support functions.

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 a 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, the Intelligent Middleware Services.

The Intelligent Middleware Services layer is supported by the Functional Services Agency whose 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. For example, 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 verify that the services are available, schedule the various subtransactions, and execute the configuration plan that is submitted by 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. 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.[67]

3.3.3. *Functional knowledge architecture*

The Functional Knowledge Architecture (FKA) contains knowledge (e.g. rules and catalogs, where catalogs are the available offerings to choose from) 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.[69] 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. This knowledge includes aspects of linguistics to focus on usage and reference as well as the traditional relations between the elements.[70]

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 (see Ref. 27) used to support Web services, some of which are mature, but most are still emerging at varying levels of maturity. Additionally, the various technologies and protocols often compete to become the industry standard; thus, increasing the instability of the protocol base. Semantic expansions to these protocols will likely be necessary to take full advantage of the expanded functionality in this framework.

3.3.5. *Grid interface*

The framework treats the Grid Interface as yet another service layer for resources that clients have available within the VO. Clients can choose to use the Grid's interface as a blackbox, designate a preferred resource within the Grid as a preference (as a graybox), or access the resources directly (i.e. if permitted to do so). The framework can designate services as having a "grid-edge" visibility to reflect that they "front" the grid for client requests; however, this does not preclude the continued use of Web services within the Grid.

Conceptually, data and metadata must be packaged to travel the Grid to the processes that will execute them. (The "fulfillment package" mentioned earlier can be very useful for this transportation.) In the context of the Semantic Web, we must ensure that pedigree and provenance, as shown in the meta-model's kdsdGraybox (graybox indicating some visibility behind the blackbox border and supports the intent of the WSRF protocol discussed in Sec. 2.2) subclass under the kdsdDescription class, are maintained via appropriate namespaces. For example, Table 1 presents a set of abstract classes representing concepts that reflect best practice for scientific information in a grid setting.[71] If WSRF is adopted, the mapping to WSRF's technical specifications will also be useful in enabling the additional visibility into the Grid.

Recently, the issue of creating an ontology for Grid environments has been proposed.[71] The authors point out that "GRID environments are service-oriented

Table 1. Scientific information classes.

Pedigree	Represents a line of ancestry from creation through various transformations to arrive at the current data set. It also includes information related to the scientific project and data identity.
Scientific_Use	Describes how a scientist used the data, what experiments were performed, what were the parameters and configuration of models.
Dataset	Describes data typically stored storage facilities, and may include parameters, location, and the study that produces this data.
Service	Concerns how a service may be invoked and what its capabilities are in a gridded architecture.
Access	Concerns whom is allowed to access the data, security and authentication.
Other	Includes annotations, comments, and evaluations.

and emphasize operations that can be performed on data using associated metadata schemas, rather than focusing upon the content of metadata schemas and relationships between schema elements". In order for middleware services to be able to respond to service requests, the metadata (both schema and instance levels) must be available to that service. The metaphor for Knowledge Sifter, a Semantic Web-enabled search agent, is that each query has an associated OWL-schema instantiation which is exchanged by the agents and updated based on the services performed. This captures the pedigree and provenance of the entire collection of activities associated with the instance.[72]

4. Application 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 focus of the scenario is to further explain how this framework works and the benefits that it provides. The scenario, as shown in Fig. 11, is explained in two manners — chronologically as the sequence of events that 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). As a note, not all of the KDL and KDSPL objects are diagrammed, but are presented in the discussion to enhance the understanding.

To demonstrate the tailoring ability of the framework, notice the specialization of the profiles in steps 1(a). Embed Provider/Service Knowledge in UDDI (*a priori*) and 1(b). Embed Provider/Service Knowledge in WSDL (*a priori*). These steps entail executing the steps in the kdsdPublishHurricaneServices process that is specific to the emergency services and hurricane domains, which pulls in the associated kdsdHurricaneConstraints and kdsdHurricanePreferences embodied in kdsdHurricaneServiceProfile and kdsdEmergencyServicesProviderProfile. kdspPublishHurricaneServices's mapping rules specify to post the semantically-enhanced ontological markup to extensions of WSDL and UDDI repositories. 2. Event Occurs represents the kdspHurricaneEvent sensed with the system matches that of kdsdHuuricaneProfile and commences fulfilling the associated needs for the emergency as represented in 3. Commence Request by executing the kdspReceiveRequest process.

To demonstrate 4. Aggregate Knowledge, Fig. 8 shows an example of a partial specification for kdsdHurricaneRequestProfile. In this case, kdsdHurricaneProfile and kdsdHurricanePattern, are included as attributes to compose the object.

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 kdsdHurricaneRequestProfile and

object-type::=kdsdHurricaneRequestProfile				
	:ATTRIBUTES	kdsdHurricaneProfile	:TYPE	Object
		kdsdHurricanePattern	:TYPE	Object

Fig. 8. Aggregation example within kdsdHurricaneRequestProfile KDL object.

object-type::=kdspMapMasterRequest			
:TARGET	kdsdHurricanceMasterRequest		
:STEPS	:STEPNAME	kdspAppend_kdsdHurricaneRequestProfile	
	:SEQUENCE-NUMBER		1
	:STEP-SUCCESSOR-MODE	Sequential	
	:STEP-SUCCESSOR-BRANCH	kdspAppend_kdsdHurricaneSearchPriorities	
	:STEPNAME	kdspAppend_kdsdHurricaneSearchPriorities	
	:SEQUENCE-NUMBER		2
	:STEP-SUCCESSOR-MODE	Sequential	
	:STEP-SUCCESSOR-BRANCH	kdspAppend_kdsdHurricaneEvent	
	:STEPNAME	kdspAppend_kdsdHurricaneEvent	
	:SEQUENCE-NUMBER		3

Fig. 9. Example of dynamic mapping of a KDL object using KDSPL.

kdsdHurricaneSearchPriorities 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 that are embodied in kdsdHurricaneEvent. Figure 9 demonstrates this using the TARGET and primitive in KDSPL.

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[62] 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.

object-type::=kdspSupplyBlankets		
:STEPS	:STEPNAME	kdspRequisitionBlankets
	:SEQUENCE-NUMBER	1
	:STEP-SUCCESSOR-MODE	Sequential
:HEURISTICS	If kdspLocalBlanket ConstraintViolation, Execute kdspLocalBlanketConstraintViolation	

Fig. 10. Example of dynamic adjustment of workflow.

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 that there are not enough blankets, which triggers kdspLocalBlanketConstraintViolation. Figure 10 shows an example of the dynamic adjustment of the workflow based on this event by using the HEURISTICS option in KDSPL.

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.

Notation A depicts the KDL specification of the object kdsdHurricaneEvent as shown in Fig. 12. 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 KDSPL Specification for the kdspSearchForProviders object shown in Fig. 13. Notice that the TASK and THREAD (kdspDiscover and kdspManagement, respectively) establish the context of the process. In this case, the KDSPL can be tailored per the task and thread in order to specialize the behavior. 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

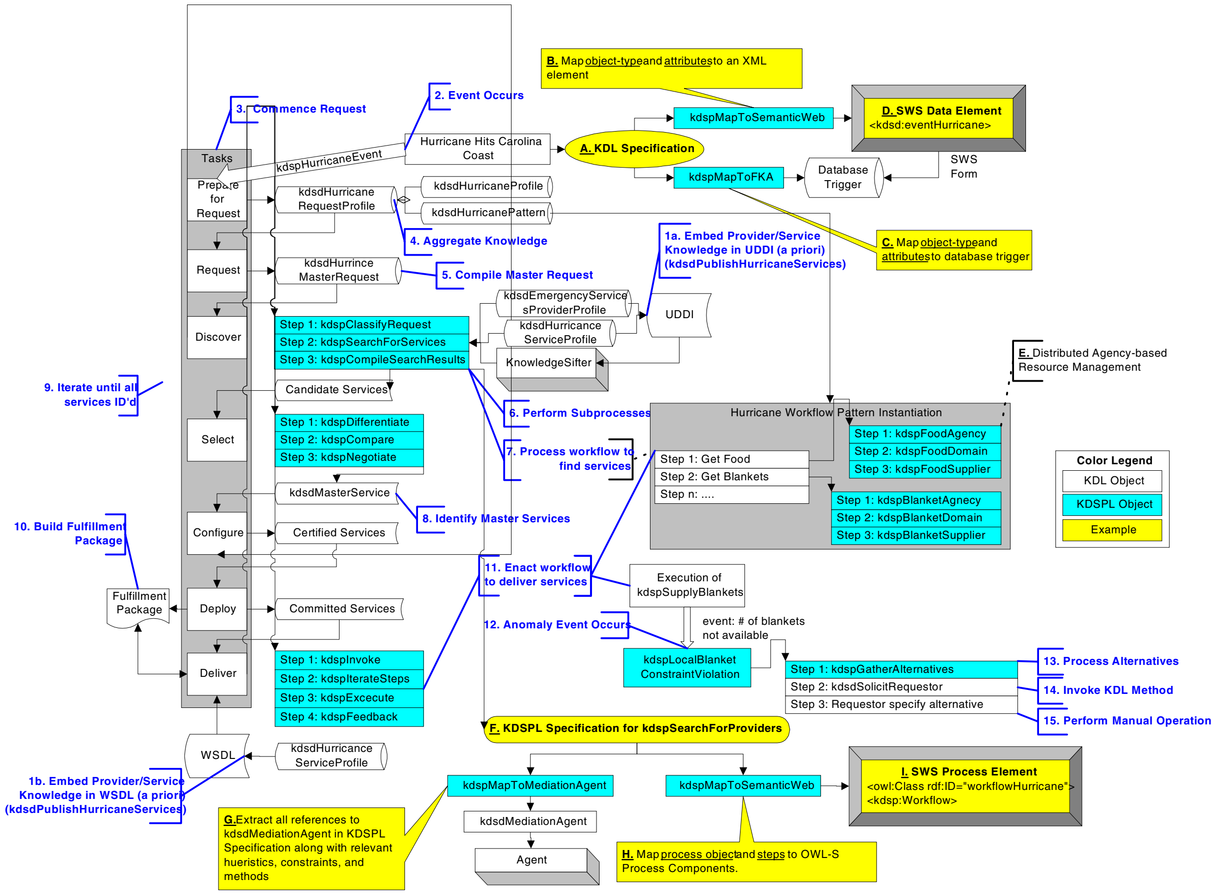
B. Map object-type and attributes to an XML element
D. SWS Data Element <kdsd:eventHurricane>
kdspMapToSemanticWeb
2. Event Occurs
3. Commence Request
Hurricane Hits Carolina Coast
A. KDL Specification
kdspMapToFKA
Database Trigger
SWS Form
Tasks
kdspHurricaneEvent
Prepare for Request
kdsdHurricane RequestProfile
kdsdHurricaneProfile
kdsdHurricanePattern
C. Map object-type and attributes to database trigger
1a. Embed Provider/Service Knowledge in UDDI (a priori) (kdsdPublishHurricaneServices)
4. Aggregate Knowledge
Request
kdsdHurrince MasterRequest
5. Compile Master Request
kdsdEmergencyServicesProviderProfile
UDDI
kdsdHurricance ServiceProfile
Discover
Step 1: kdspClassifyRequest
Step 2: kdspSearchForServices
Step 3: kdspCompileSearchResults
KnowledgeSifter
E. Distributed Agency-based Resource Management
9. Iterate until all services ID'd
Candidate Services
6. Perform Subprocesses
Hurricane Workflow Pattern Instantiation
Select
Step 1: kdspDifferentiate
Step 2: kdspCompare
Step 3: kdspNegotiate
7. Process workflow to find services
Step 1: kdspFoodAgency
Step 2: kdspFoodDomain
Step 3: kdspFoodSupplier
Step 1: Get Food
Step 2: Get Blankets
Step n: ....
Color Legend
KDL Object
KDSPL Object
Example
kdsdMasterService
Step 1: kdspBlanketAgecy
Step 2: kdspBlanketDomain
Step 3: kdspBlanketSupplier
10. Build Fulfillment Package
Configure
Certified Services
8. Identify Master Services
11. Enact workflow to deliver services
Execution of kdspSupplyBlankets
Fulfillment Package
Deploy
Committed Services
12. Anomaly Event Occurs
event: # of blankets not available
kdspLocalBlanket ConstraintViolation
Deliver
Step 1: kdspInvoke
Step 2: kdspIterateSteps
Step 3: kdspExcecute
Step 4: kdspFeedback
Step 1: kdspGatherAlternatives
Step 2: kdsdSolicitRequestor
Step 3: Requestor specify alternative
13. Process Alternatives
14. Invoke KDL Method
15. Perform Manual Operation
F. KDSPL Specification for kdspSearchForProviders
WSDL
kdsdHurricance ServiceProfile
1b. Embed Provider/Service Knowledge in WSDL (a priori) (kdsdPublishHurricaneServices)
kdspMapToMediationAgent
kdspMapToSemanticWeb
I. SWS Process Element <owl:Class rdf:ID="workflowHurricane"> <kdsp:Workflow>
G. Extract all references to kdsdMediationAgent in KDSPL Specification along with relevant hueristics, constraints, and methods
kdsdMediationAgent
Agent
H. Map process object and steps to OWL-S Process Components.

Fig. 11. Scenario process and objects.

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				

Fig. 12. kdsdHurricaneEvent KDL specification.

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				
:STEPS	:STEPNAME	SearchUDDI			
	:SEQUENCE-NUMBER	1			
	:STEP-DESCRIPTION	Search the UDDI registry for acceptable providers and services			
	:STEP-SUCCESSOR-MODE	Decision			
	:STEP-SUCCESSOR-BRANCH	kdspAdjustSearchParameters	:STEP-CONTROL-CONDITION	Insufficient Results	
	:STEP-SUCCESSOR-MODE	Sequential			
	:STEP-SUCCESSOR-BRANCH	kdspRankResults	:STEP-CONTROL-CONDITION	Sufficient Results	
	:DELEGATE	kdsdKnowledgeSifter			
		:DELEGATE-TYPE	AGENT		
		:DELEGATE-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"				

Fig. 13. kdspSearchForProviders KDSPL specification.

PREDECESSOR, kdspClassifyRequest is the previous process in the chain. The next step in the process will be the SUCCESSOR kdspCompileSearchResults.

This sample process object has the single step of SearchUDDI, with kdsdKnowledgeSifter designated as the DELEGATE as well. Notice the branching ability of KDSPL via the STEP-SUCCESSOR-MODE, STEP-CONTROL-CONDITION and STEP-SUCCESSOR-BRANCH arguments. To demonstrate another method to dynamically adjust the workflow, if "Insufficient Results" are delivered, the process kdspAdjustSearchParameters is invoked; otherwise the process continues to execute kdspRankResults. The DELEGATE-TYPE designates a line agent (via the DELEGATE-ROLE). 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.

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. Future Work

This research is still at an abstract level, and a proof-of-concept implementation of the framework is planned to demonstrate and mature the facets of the research. The first step is to create a demonstration of how the components work together to automate Web services. Future work for the framework also includes creating a modeling facility for the unique approach of the multi-dimensional methodology that integrates the meta-model.

Figure 14 shows the architecture with KDSWS Specification and KDSWS Functional Architecture components to support the limited implementation. The three basic types of objects in the specification are atomic, composed and internally-mapped (or mapped for short). The atomic objects are comprised of attributes are not other objects; contrasted with the aggregated objects whose attributes consist of other objects (type = Object). The mapped objects' rules are invoked dynamically to aggregate the object, which is achieved by adding the kdsdMapObject attribute (set to True) and including a test for the Boolean condition in the rules that are involved in the aggregation. Other methods for this dynamic mapping are explained in Sec. 3.

The Import Agent and Export Agent are specialized types of the Mapping Agent that control the mapping in and out of the KDL and KDSPL specifications, and the three agents are examples of internal agents that support the infrastructure of the framework. The implementation imports and maps the specialty engine (e.g. WfMS and Expert Systems) data into the specification. Agent Profiles are exported to specify the purpose and behavior of the agents involved in the implementation; whereas, Profiles are exported to guide the life-cycle activities. The Master Request is exported out of the specification and into an enact-able form that the Request Handling Agent can process to commence the work of the Workflow and Broker. Knowledge Objects are exported to not only to store knowledge, but to specify how to use the knowledge (in this case, the ontologies are the targeted object). We also plan to incorporate the Knowledge Sifter agent into the brokering activities to demonstrate interoperation with existing agents.

The implementation will map to our semantically-enhanced versions (denoted by the "+" suffixes) of UDDI, WSDL, and OWL-S to demonstrate how the framework co-exists with current technology offerings. Recommendations are emerging in the literature for these semantic enhancements. In order to facilitate quicker adoption, we are leaning towards structures that leverage existing functionality within current technologies as much as possible. We are exploring the use of specialized

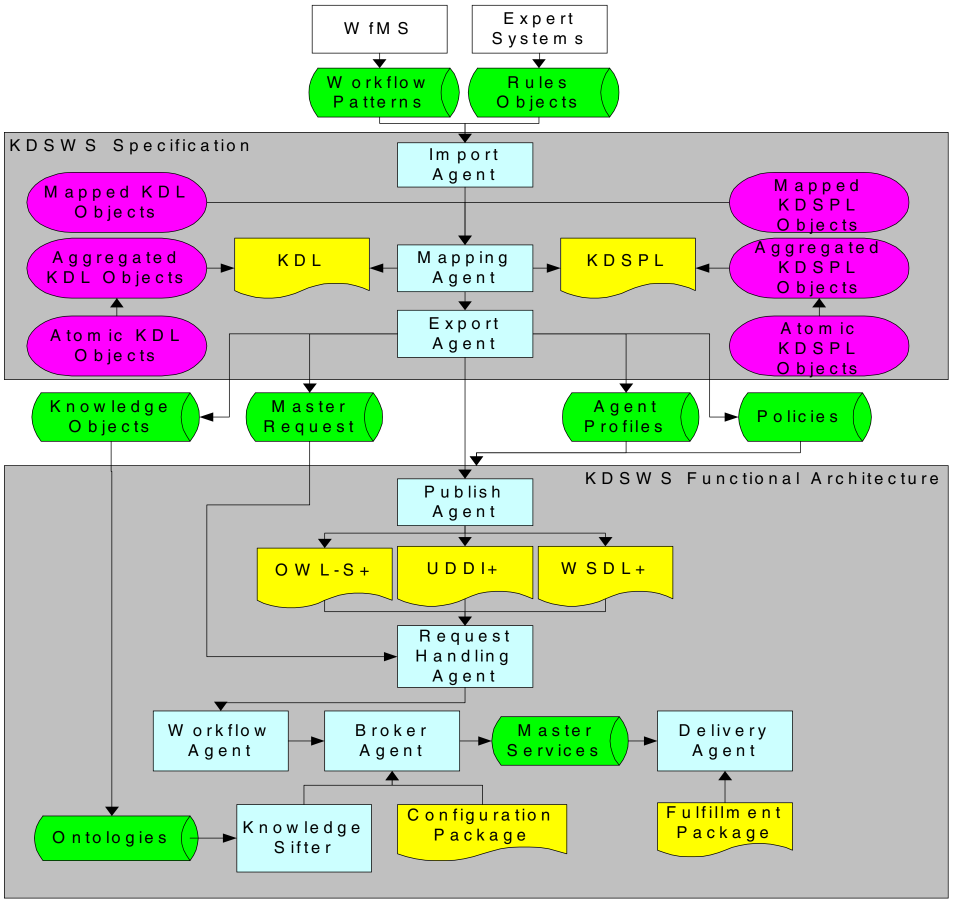

Fig. 14. Scenario implementation architecture.

versions of the root and affiliate registries introduced with UDDI-3.[73] We hope to create specialized data types for the WSDL or possibly establishing the Fulfillment Package as an attachment to the SOAP message. For OWL-S, we feel that new extensions will be necessary to support the implementation of this research. It is also possible that additional concepts with the refining of concepts will be required for this to work with OWL-S.

6. Conclusions

In order to accelerate the introduction of new concepts and systems, we need approaches which automate the entire Web services life-cycle and address the issues related to Virtual Organizations. This paper presents an agent-based approach to managing the brokering of Semantic Web Services for use within a Virtual Organization, and is part of a larger methodology, called the KDSWS

Framework. The framework provides a formal model-based approach to implementing Web services that specifies the modeling, specification, design, implementation and deployment of systems composed of SWS. The meta-meta-model combines KDSWS constructs with RDF and RDFS concepts. The meta-model is couched in UML terminology which can be mapped to OWL.

The goal of the framework is to support the automatic discovery, composition, execution and management of Web services for the Virtual Organization in a protocol-independent manner. This VO is the paradigm we use to denote that the KDSWS can handle both Inter-Enterprise and Intra-Enterprise coordination and interoperation issues.

The framework provides a broader scope because it provides a comprehensive solution by addressing the backend specification for federating enterprise resources, agents and knowledge repositories. It is integrated because the de-coupled data (description) and process (function) specifications, as well as the overarching methodology and process instantiations, are specified by the same foundation — the KDM/KDL. Interoperability is facilitated by the integrated design space and mediation structures. The framework is knowledge-based because it uses heuristics to associate the knowledge to objects and services, and captures usage context as well. This rules-based approach facilitates the dynamic profiling of resources as well as quick adaptation to the rapidly changing Web services standard and protocol base. The term "semantic" denotes the knowledge-based semantic specification of the relevant features and functions provided by the Web Service.

In order to address the multitude of issues in this area, we propose the KDSWS Framework as a way to combine three important, and inter-related, viewpoints:

- The KDSWS Process viewpoint addresses issues related to workflow, transaction-control, security, and interoperation in the form of "threads". The delineation between the resource-based and process-based classes in the meta-model provides crisper concepts for the framework to specify tighter designs.
- The KDSWS Design Specification viewpoint models objects, relationships, constraints, heuristics, and processes using the KDM/KDL and extensions KDSPM/KDSPL to handle special features of Semantic Web Service specifications. The advantage provided by a separate and integrated specification is the VO can adjust to a constantly changing protocol base more rapidly by developing the mappings from the consistent base versus having to recode the affected interfaces.
- 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, 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,[5] 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 semantically-enabled workflow and feedback processes provide a managed approach to the dynamic delivery of Web services. 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.

Acknowledgments

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 US Government.

References

1. A. Bosworth, Developing web services, in *Proc. 17th Int. Conf. Data Engineering*, 2001.
2. UDDI, UDDI version 3 features list, http://www.uddi.org/pubs/uddi_v3_features.htm#_Toc10457162.
3. M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau and H. F. Nielsen, SOAP Version 1.2 Part 1: Messaging framework, 2003, http://www.w3.org/TR/SOAP/.
4. R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer and S. Weerawarana, Web services description language (WSDL Version 2.0 Part 1: Core language, 2002), http://www.w3.org/TR/wsdl20/.
5. C. Bussler, D. Fensel and N. Sadeh, Semantic web services and their role in enterprise application integration and e-commerce, 2003, http://lists.w3.org/Archives/Public/www-rdf-interest/2003Mar/att-0008/ijec_special_issue_cfp.pdf.
6. T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities," *Scientific American* **284** (2001), 34–43, http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.
7. O. Lassila and R. R. Swick, Resource Description Framework (RDF) model and syntax specification, 1999, http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/#intro.

8. J. Hendler, DAML: DARPA agent markup language effort (http://www.daml.org/), 2002, http://www.daml.org/.
9. Ontoknowledge, Description of OIL, http://www.ontoknowledge.org/oil/.
10. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness and P. F. Patel-Schneider, OIL: An ontology infrastructure for the Semantic Web, *IEEE Intelligent Systems* **16** (2001) 38–45.
11. DAML.org, Reference description of the DAML+OIL (March 2001) ontology markup language, 2001, http://www.daml.org/2001/03/reference.html.
12. S. Bechhofer, F. v. Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider and L. A. Stein, "OWL Web ontology language reference (2003), http://www.w3.org/TR/owl-ref/#DatatypeSupport.
13. DAML.org, OWL-S: Semantic markup for web services (2003) http://www.daml.org/services/owl-s/1.0/owl-s.html.
14. M. Paolucci, K. Sycara and T. Kawamura, Delivering Semantic Web services (2003), http://www.ri.cmu.edu/pub_files/pub4/paolucci_massimo_2003_1/ paolucci_massimo_2003_1.pdf.
15. T. Berners-Lee, Web services — Semantic Web (2003), http://www.w3.org/2003/Talks/0521-www-keynote-tbl/Overview.html.
16. E. Cerami, *Web Services Essentials*, 1st edn (O'Reilly, CA, Sebastopol, Beijing, 2002).
17. S. A. McIlraith, T. C. Son and H. Zeng, Semantic web services, *IEEE Intelligent Systems* **16** (2001) 46–53.
18. R. L. Reid, K. J. Rogers, M. E. Johnson and D. H. Liles, Engineering the virtual enterprise, Automation & Robotics Research Institute, http://arri.uta.edu/eif/ve_ie.pdf.
19. I.-Y. Ko and R. Neches, Composing web services for large-scale tasks, *IEEE Internet Computing* (September/October, 2003), pp. 52–59.
20. OASIS, Security Assertion Markup Language (SAML) (2002), http://www.oasis-open.org/committees/security.
21. IBM, Specification: Web Services Security (WS-Security, 2002), ftp://www6.software.ibm.com/software/developer/library/ws-secure.pdf.
22. A.-W. a. N. Scheer, Markus, (*ARIS architecture and reference models for business process management*), 2000.
23. A. Arkin, Business process modeling language (2002), http://www.bpmi.org/bpml-spec.esp.
24. A. Gomez-Perez, R. Gonzalez-Cabero and M. Lama, A framework for design and composition of Semantic Web services, *2004 AAAI Spring Symposium Series*, Stanford University Palo Alto, CA (2004), http://www.daml.ecs.soton.ac.uk/SSS-SWS04/44.pdf.
25. P. Mika, M. Sabou, A. Gangemi and D. Oberle, Foundations for OWL-S: Aligning OWL-S to DOLCE, *2004 AAAI Spring Symposium Series*, Stanford University Palo Alto, CA (2004), http://www.daml.ecs.soton.ac.uk/SSS-SWS04/23.pdf/.
26. M. Paolucci, J. Soudry, N. Srinivasan and K. Sycara, A broker for OWL-S web services, *2004 AAAI Spring Symposium Series*, Stanford University Palo Alto, CA (2004), http://www.daml.ecs.soton.ac.uk/SSS-SWS04/40.pdf.
27. L. Wilkes, The web services protocol stack, CBDI Web Services Roadmap (2004), http://roadmap.cbdiforum.com/reports/protocols/index.php.
28. IBM, Specification: Business process execution language for web services version 1.1 (2003), http://www-106.ibm.com/developerworks/library/ws-bpel/.
29. A. Arkin, S. Askary, S. Fordin, W. Jekeli, K. Kawaguchi, D. Orchard, S. Pogliani, K. Riemer, S. Struble, P. Takacsi-Nagy, I. Trickovic and S. Zimek, Web Service Choreography Interface (WSCI) 1.0 (2002), http://www.w3.org/TR/wsci/.

30. D. Austin, A. Barbir, E. Peters and S. Ross-Talbot, Web services choreography requirements 1.0 (2003), http://www.w3.org/TR/2003/WD-ws-chor-reqs-20030812/.
31. D. Briukhov, L. Kalinichenko and I. Tyurin, Extension of compositional information systems development for the web services platform, *Advances in Databases and Information Systems* **2798/2003** (2003) 16–29.
32. WSMO, Web services modeling ontology (2004), http://www.wsmo.org/index.html.
33. B. Hofreiter, C. Huemer and W. Klas, ebXML: Status, research issues and obstacles, *Research Issues in Data Engineering: Engineering E-Commerce/E-Business Systems*, 2002. RIDE-2EC 2002, Proceedings of the Twelfth International Workshop (2002).
34. ebXML, About ebXML (2004), http://www.ebxml.org/geninfo.htm.
35. S. Patil and E. Newcomer, ebXML and web services, *Internet Computing, IEEE* **7** (2003) 74–82.
36. RosettaNet, RosettaNet PIPs (2004), http://www.rosettanet.org/RosettaNet/Rooms/DisplayPages/LayoutInitial?Container=com.webridge.entity.Entity[OID[279B86B8022CD411841F00C04F689339]].
37. Whatis.techtarget.com, Virtual organization, http://whatis.techtarget.com/definition/0,,sid9_gci213301,00.html/.
38. I. Foster, C. Kesselman and S. Tuecke, The anatomy of the grid: Enabling scalable virtual organizations (2001), http://www.globus.org/research/papers/anatomy.pdf.
39. Grid computing, searchcio.techtarget.com. http://searchcio.techtarget.com/sDefinition/0,,sid19_gci773157,00.html.
40. I. Foster, C. Kesselman, J. M. Nick and S. Tuecke, The physiology of the grid: An open grid services architecture for distributed systems integration (2002), http://www.globus.org/research/papers/physiology.pdf.
41. M. Cannataro, Knowledge discovery and ontology-based services on the grid, in *Proc. Global Grid Forum Semantic Grid Research Group*, 2003, in Chicago IL, USA.
42. A. M. Tjoa, P. Brezany and I. Janciak, Towards grid based intelligent information systems, in *Proc. 2003 Int. Conf. Parallel Processing* (2003).
43. P. Shread, Grid, web services get closer (2004), http://www.gridcomputingplanet.com/news/article.php/3304571.
44. K. Czajkowski, D. F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke and W. Vambenepe, The WS-resource framework (2004), http://www-106.ibm.com/developerworks/library/ws-resource/ws-wsrf.pdf.
45. N. Nayak, K. Bhaskaran and R. Das, Virtual enterprises-building blocks for dynamic e-business VO-, in *Proc. Workshop on Information Technology for Virtual Enterprises (ITVE 2001)* in (2001).
46. D. Booth, H. Haas, F. McCabe and E. Newcomer, Web services architecture (2003), http://www.w3.org/TR/ws-arch/.
47. OASIS, Management using web services: Architecture (2003), http://www.oasis-open.org/committees/documents.php?wg_abbrev=wsdm.
48. S. Bajaj, G. Della-Libera, B. Dixon *et al.*, Web services federation language (WS-Federation) (2003), http://msdn.microsoft.com/webservices/understanding/advancedwebservices/default.aspx?pull=/library/en-us/dnglobspec/html/ws-federation.asp.
49. M. Giovanni Della-Libera, M. Brendan Dixon, M. Mike Dusche *et al.*, Federation of identities in a web services world (2003), http://www-106.ibm.com/developerworks/webservices/library/ws-fedworld/.
50. R. Howard and L. Kerschberg, A framework for dynamic Semantic Web services management, Special Issue on Service Oriented Modeling, *Int. J. Coop. Information Systems*, (Accepted for Publication) (2004), http://www.ise.gmu.edu/techrep/2004/.

51. D. Fensel and C. Bussler, The web service modeling framework WSMF, http://www.swsi.org/resources/wsmf-paper.pdf.
52. W. D. Potter and L. Kerschberg, The knowledge/data model: An integrated approach to modeling knowledge and data, *Data and Knowledge (DS-2)*, eds. R. A. Meersman and A. C. Sernadas (North Holland, Amsterdam, 1988).
53. G. Heidel, Web services standards: Can there be a consensus?, www.wsj2.com (2002), http://classweb.gmu.edu/kersch/infs770/Topics/Web_Services/WebServicesStandards.pdf.
54. D. A. Menasce, QoS issues in Web services, *IEEE Internet Computing* **6** (2002) 72–75.
55. J. A. Miller, W. D. Potter and K. J. Kochut, On the integration of knowledge, data and models (1992), http://citeseer.nj.nec.com/cache/papers/cs/821/http:zSzzSzorion.cs.uga.edu:5080zSz~jamzSzjsimzSz..zSzpaperszSzintegration.pdf/on-the-integration-of.pdf.
56. ObjectManagementGroup, Meta Object Facility (MOF) specification version 1.4 (2003), http://www.omg.org/docs/formal/02-04-03.pdf.
57. A. Sheth, C. Bertram, D. Avant, B. Hammond, K. Kochut, and Y. Warke, Managing semantic content for the web, *IEEE Internet Computing* **6** (2002), 80–87.
58. H. Haas and D. Orchard, Web services architecture usage scenarios (2003), http://www.w3.org/TR/2003/WD-ws-arch-scenarios-20030514/#S201.
59. DublinCore, Dublin core metadata element set, version 1.1: reference description (2003), http://www.dublincore.org/documents/dces/.
60. M. Nodine, A. H. H. Ngu, A. Cassandra and W. G. Bohrer, Scalable semantic brokering over dynamic heterogeneous data sources in InfoSleuth, *IEEE Trans. Knowledge and Data Engineering* **15** (2003) 1082–1098.
61. A. de Moor and W.-J. van den Heuvel, Web service selection in virtual communities, *System Sciences, 2004, Proc. 37th Annual Hawaii Int. Conf.* (2004).
62. L. Kerschberg, M. Chowdhury, A. Damiano, H. Jeong, S. Mitchell, J. Si and S. Smith, Knowledge sifter: Ontology-driven search over heterogeneous databases, *SSDBM 2004, Int. Conf. Scientific and Statistical Database Management*, Santorini Island, Greece, IEEE, 2004 (submitted for publication).
63. L. Kerschberg, Functional approach to in internet-based applications: Enabling the semantic web, e-business, web services and agent-based knowledge management, in *The Functional Approach to Data Management*, eds. P. M. D. Gray, L. Kerschberg, P. King and A. Poulovassilis (Springer, Heidelbeg, 2004) 369–392.
64. L. Kerschberg and D. J. Weishar, Conceptual models and architectures for advanced information systems, *Applied Intelligence* **13** (2000) 149–164.
65. B. Ryder, Formal languages (2002), http://remus.rutgers.edu/cs314/f2002/ryder/lectures/formal-2New.pdf.
66. J. Miller, W. Potter, K. Kochut, A. Keskin and E. Ucar, The active KDL object oriented database system and its application to simulation support, *J. Object-Oriented Programming*, Special Issue on Databases (1991), pp. 30–45. http://citeseer.nj.nec.com/cache/papers/cs/821/http:zSzzSzorion.cs.uga.edu:5080zSz~jamzSzjsimzSz..zSzpaperszSzactivekdl.pdf/miller91active.pdf.
67. P. M. D. Gray, L. Kerschberg, P. King and A. Poulovassilis, *The Functional Approach to Data Management* (Springer, Heidelberg, 2003).
68. A. Brodsky, L. Kerschberg and S. Varas, Resource management in agent-based distributed environments, Lecture Notes in Computer Science, vol. 1652/1999, 1999, pp. 61–85.

69. N. F. Noy and M. A. Musen, Ontology versioning as an element of an ontology-evolution framework, *IEEE Intelligent Systems* (2003), http://smi-web.stanford.edu/pubs/SMI_Abstracts/SMI-2003-0961.html.
70. R. Urro and W. Winiwarter, Specifying ontologies: Linguistic aspects in problem-driven knowledge engineering, in *Proc. Second Int. Conf. Web Information Systems Engineering* (2001).
71. L. Pouchard, L. Cinquini, B. Drach, D. Middleton, D. E. Bernholdt, K. Chanchio, I. T. Foster, V. Nefedova, D. Brown, P. Fox, J. Garcia, G. Strand, D. Williams, A. L. Chervanek, C. Kesselman, A. Shoshani and A. Sim, An ontology for scientific information in a grid environment: The earth system grid, *CCGRID 2003* (2003).
72. L. Kerschberg, M. Chowdhury, A. Damiano, H. Jeong, S. Mitchell, J. Si and S. Smith, Knowledge sifter: Ontology-driven search over heterogeneous databases, *SSDBM 2004, Int. Conf. Scientific and Statistical Database Management*, 2004, Santorini Island, Greece.
73. UDDI, UDDI version 3 Features list (2004), http://uddi.org/pubs/uddi_v3_features.htm.