Discovering and integrating distributed manufacturing services with semantic manufacturing capability profiles
(Running title: Semantic-based manufacturing integration)

Jungho Jang*, Buhwan Jeong*[†], Boonserm Kulvatunyou‡, Jaegyong Chang¶, and Hyunbo Cho*

*Department of Industrial and Management Engineering
Pohang University of Science and Technology (POSTECH)
San 31, Hyoja, Pohang, 790-784, South Korea

‡Manufacturing Systems Integration Division
National Institute of Standards and Technology (NIST)
100 Bureau Dr., Gaithersburg, MD, 20899

¶Korea Institute for Electronic Commerce
6F Textile Center, Daechi, Kangnam, Seoul, 135-713, South Korea

Abstract

Integrating distributed manufacturing systems is a longstanding dream of industrial engineers. The advent of Internet technologies has provided opportunities to fulfill this dream, and has presented new challenges to overcome. Since most current Internet technologies including SOA and web services, originated in business applications, it is difficult to apply them directly to manufacturing systems. The difficulties stem particularly from differences in meaning and usage of manufacturing terms and an inability to express semantic information about manufacturing services. The present paper aims to extend the UDDI registry specification to include semantic descriptions about manufacturing services and to support reasoning about those descriptions for service discovery. Specifically, we provide OWL-based definitions for manufacturing service capability profiles and a DL-based reasoning procedure for matching queries to service descriptions. An illustrative process is presented with a prototype implementation for a discrete part manufacturing case.

Keywords: Distributed manufacturing, manufacturing capability profile, manufacturing ontology, manufacturing service, semantic matchmaking, service discovery, UDDI

[†] Corresponding Author. Email: bjeong@postech.ac.kr

1. Introduction

Within the manufacturing environment, successful implementations of a service-oriented architecture (SOA) are accelerating the distribution of manufacturing facilities, which reduces the physical coupling between designers and factories. Now, that coupling can be established virtually through a process of service advertisement, discovery, composition, and invocation. The key ingredient necessary for supporting such a process is a reliable registry, where every service is advertised and accessed openly. One of the most promising registries for SOA is Universal Description, Discovery, and Integration (UDDI) (Clement *et al*. 2004). UDDI is a remotely searchable registry that maintains information about providers, services, and businesses, as well as technical information for requesting and/or receiving services.

Briefly, UDDI provides a simple data structure based on key-value pairs to express technical specifications about ‘services’. Each UDDI registry entry uses four core data types: *businessEntity*, *businessService*, *bindingTemplate*, and *tModel*. *businessEntity*, which is the top-level element, contains data about businesses and providers; e.g., business or provider name, contact information, business type. It can have one or more *businessService* elements, which capture descriptive information about services. The technical information about a service is contained in the *tModel* element. The *bindingTemplate* links a *businessService* element with one or more *tModel* elements. The *businessEntity* and *businessService* elements can associate directly with *tModel* elements via a *categoryBag* element. Their logical relationship is depicted in Figure 1.

Figure 1

Although the simplicity of the data structure allows users to advertise and search service descriptions easily, the use of UDDI in specific industry domains, like manufacturing, has two main drawbacks. The first arises from the fact that the pre-defined data structures in UDDI are incompatible with those used for manufacturing services[1]. The second arises from UDDI’s inability to express semantic descriptions about manufacturing services. Semantic descriptions may be included in the *description* tag, but they are not computer-interpretable. We conclude, therefore, that UDDI cannot store enough semantic information about manufacturing services to support the advanced inferences required for accurate discovery.

---

[1] Web services are self-contained, self-describing, modular computer components interfaced with XML messages.

This paper addresses this shortcoming of UDDI by (1) enhancing the registry's functionality to encapsulate semantic service profiles with detailed manufacturing capabilities, and (2) proposing semantic matchmaking techniques to discover those capabilities. To this end, the paper proposes an extension to UDDI - *serviceProfil* - that contains OWL (Web Ontology Language)-based service descriptions and a new DL-based reasoner to perform semantic matchmaking based on those descriptions. It also includes a case study in discrete part manufacturing to demonstrate both.

The remainder of the paper is organized as follows. Related works on service advertisement and discovery are summarized briefly in Section 2. Section 3 addresses the characteristics distinguishing a manufacturing service from a general web service. The proposed UDDI extension is outlined in Section 4. Section 5 presents an ontology for describing a manufacturing service and a reasoner for finding such semantic matches. Section 6 describes a case study, encompassing the entire process from ontology design to service discovery. Finally, a summary is given in Section 7.

2. Related work

SOA provides a flexible and powerful means to organize and execute public (web) services as one's native services. The UDDI registry, which can be thought of as an information marketplace, acts as a mediator between service providers and service requestors. Providers advertise their service descriptions in the registry; requestors search the registry to discover services that meet their needs, and invoke those services using a predefined set of messages. In cases where the need cannot be met by one service, the requestor can invoke multiple services and compose them to behave as a single service. Since this paper focuses mainly on service advertisement and discovery, we summarize the state-of-art in these two areas in the remainder of this section.

2.1 Service advertisement

As noted above, UDDI's data structures and associated semantics are not sufficiently rich to support automatic discovery of manufacturing service capabilities. To overcome this, Paolucci *et al*. (2002) and Srinivasan *et al*. (2004) redefined the UDDI specification by using DAML-S and OWL-S, respectively. Specifically, they first represent the service capabilities in one of these languages, and then map the resulting data structures into the corresponding ones in UDDI. They defined additional *tModel* elements for those data structures that have no corresponding elements in UDDI. This redefinition enables service providers to advertise their service profiles the ontology languages, but still use existing UDDI APIs (Application Programming Interfaces). The authors also designed a DAML-S/OWL-S and UDDI matchmaker that matches the advertised service descriptions with service queries. This approach works well for simple services; but, as the services become more complicated, requiring more and more *tModel* elements, the advertising of profiles becomes cumbersome. Pokraev *et al*. (2003) also employed DAML-S to define key-values in *tModel* elements. Although these approaches provide users with richer semantics to describe and locate services, they still do not support the full scope of manufacturing capability profiles. This is because the DAML/OWL-S languages are designed for general web service descriptions rather than for domain-specific semantics, i.e., manufacturing profiles.

Instead of using the UDDI registry, Dogac *et al*. (2004) extended the ebXML Registry Information Model (RIM)[2] with OWL-like data structures and assumed OWL semantics for these structures. In particular, they redefined the RegistryObject field in the ebXML RIM. This redefinition results in a RegistryObject that is a semantics container, thereby allowing the service providers to publish and advertise more semantically precise information. In addition, it allows the service description to be matched through an additional module that can make inferences based on assumed OWL semantics. However, since this approach does not allow OWL ontologies to be stored directly, there exists a risk of losing expressivity and functionality of the ontology language.

Kulvatunyou *et al*. (2005) extended DAML-S to enable the advertising of general manufacturing capabilities. They proposed a semantic markup of manufacturing capabilities using DAML encoding and assumed that manufacturers have a pointer directed to where the definitions of the semantic markups can be retrieved. To describe manufacturing capabilities in SOA registries, they defined a new element, *domainServiceCategory*, as a subtype of the *serviceCategory* of DAML-S. This approach allows manufacturing service providers to represent their services fully and requestors to discover them easily. Despite its merits, however, this approach suffers from the problem that it is difficult to associate the service capabilities described in DAML-S with data structures stored in UDDI.

2.2 Service discovery

Several approaches have been used to discover services in the UDDI registry. ShaikhAli *et al*. (2003) proposed a simple approach that modifies the UDDI registry with an additional component for describing user-defined properties associated with a service. With this

[2] http://www.ebxml.org/specs/ebrim2.pdf

modification, users can discover services by searching on those extended properties. While this modified UDDI registry can coexist with existing UDDI registries, it sill cannot support reasoning based on ontological descriptions of a service and its relations with other services. Luo *et al.* (2005) provides semantic query processing for complex, semantic, service descriptions stored in the UDDI registry. The query processing supports ontology mapping and service-description mapping.

Akkiraju *et al.* (2003) extended the UDDI inquiry APIs to enable requestors to specify the capabilities required of a service, and enhanced the service discovery capability of UDDI via semantic matching, thereby enabling automatic service composition and execution. This approach allows requestors to discover effectively services based on the service interfaces; unfortunately, however, it results in inferior semantic matches. Syncara *et al.* (2003) used the concept of the match degree by using DAML+OIL[3] to reduce the possibility that a requestor cannot find a suitable service due to differences between interfaces. They defined the match degree as the distance between concepts in a taxonomy tree, in terms of an 'exact match', 'more/less general match', and 'mismatch'. This approach may fail in computing the match degrees among complex service descriptions.

The key to semantics-based service discovery is the concept of a semantic match between a service description and a query. Semantic matches are identified by using ontology mapping algorithms that compute semantic similarity between two ontological objects. This computation exploits all types of information about those objects available including lexical, structural, and logical (Jeong *et al*. 2005). Three examples of similarity-based mapping tools are Rondo (Melnik *et al*. 2002), Cupid (Madhaven *et al*. 2001), and PROMPT (Noy and Musen 2000).

3. Manufacturing service

3.1 Characteristics

We define a manufacturing service as any activity that uses physical equipment to operate on raw or in-process materials. There are four important differences between our definition of a manufacturing service and the general web services in common use today.

[3] http://www.daml.org/2001/03/daml+oil-index.html

1. A manufacturing service is associated with equipment that transforms the shape or state of a physical object. A web service is associated with a software application that transforms information entities such as XML messages or documents.
2. For manufacturing services, physical distance and geographical constraints impact the invocation and integration of manufacturing services. For web services they are irrelevant because web services can be invoked virtually anywhere and anytime.
3. The capability profile for a manufacturing service must contain a variety of complex information such as the service outputs (available products), the service inputs (available raw materials or castings), the service operations (available equipment) among others. Furthermore, in many cases such information must be very precise and rich; e.g., machining tolerance, processing capacity, lead-time, and so on.
4. A service requestor must express needs in terms that can be matched against capability profiles stored in the registry. This means the specification and capability profile must be transparently transferred to the requestor. Contrary to business services, where sharing information on service capabilities is a threat to the intellectual assets of the business, revealing the capabilities of manufacturing services to requestors as much and correctly as possible increases the chance to collaborate, thereby providing an opportunity to create real value to both sides.

The above features justify our conclusion that the current methods for advertising web services - UDDI, DAML/OWL-S, and WSDL – are insufficient for advertising manufacturing services. In the following section, we describe a new method that we believe will work.

3.2 Manufacturing service capability profile

Let us now take a further look at the kinds of manufacturing-service information that must be published by a provider in a public registry so that a requestor can find a manufacturing service that meets the requestor's requirements. Here we describe only abstractly the major types of information that should be included in a manufacturing-service capability for a discrete part manufacturer because specific details about those types will vary from one manufacturing domain to another (Kulvatunyou *et al*. 2005, Schlenoff *et al*. 2000, Kevin *et al*. 1996).

The first type includes details about the products that the manufacturer can produce. A product catalog is the typical example of this information type. It should include detailed specifications about the products including name, shape, size, raw material, and a design drawing, among other things.

The second type includes details about the manufacturing processes and other operations the service is capable of providing. It should include detailed specifications about material handling and storage options, material removal processes, assembly operations, inspection operations, and transportation operations.

The third type includes details about the equipment devices - machining tools, cutting tools, jigs and fixtures, robots, AGVs, and AS/RSs - used by the provider to perform the process and operations described above. In addition to such physical devices, it may be desirable to identify certain software applications that influence directly the selection of or the use of those devices. Examples include process planning, NC simulator, NC controller, tool selector, and manufacturing cost estimator (Schlenoff *et al*. 2000).

The fourth type includes details about the geographical location of any factories, warehouses, and other facilities that will be involved in the service. These are important because they heavily influence the delivery costs and lead time.

3.3 UDDI's inability to represent manufacturing services

The UDDI framework provides a powerful mechanism for advertising and discovering services with a predefined data structure (e.g., *categoryBag*, *tModel*); but, it suffers from a lack of semantics. First, in contrast to other service profile languages such as DAML-S and OWL-S, UDDI's predefined data structures are insufficient to represent all of the information types described above. Second, they cannot incorporate structured semantic information about manufacturing service capabilities, except for a plain text description, thereby making it impossible for services to be discovered based on the details of those capabilities. Third, basic service information cannot be easily matched against requestor specifications when they use (1) different terms for the same thing; e.g., name = "Hole Making" vs. name = "Drilling", (2) different metrics for the same measurement; e.g. diameter = "100mm" vs. diameter = "10cm" vs. diameter = "3.94 in", or (3) different classification codes for the same thing: e.g., NAICS 331[4] vs. UNSPSC[5] 73.16.15.09.

4. Extension of UDDI

To cope with these limitations, we have developed an extension of the UDDI data structures to allow for semantics-rich descriptions of a manufacturing service. These descriptions will allow requestors to reason over and to discover the best matches for a given service query.

[4] North American Industrial Classification Scheme, http://www.naics.com
[5] Universal Standards Products and Services Classification, http://www.eccma.org/unspsc

We incorporated three design constraints into our extension. First, the current UDDI structure must be preserved as much as possible to ensure backward compatibility with existing registries. Second, the registry must be able to exploit the additional semantics for accurate service discovery. Third, the registry should be domain- and/or application-independent; that is, we should avoid defining any domain-specific data structures. In particular, for the second constraint, we will represent the service description in a machine-interpretable and semantics-rich form using any ontology description language and also provide ontology-reasoning functionalities.

Our extension of the UDDI structure is depicted conceptually in Figure 2, where we insert a new container for semantic descriptions - the *serviceProfile* element into the *businessService* structure (Jang *et al*. 2005). In the *serviceProfile* element, a provider can include its service description according to a publicly available or privately used semantics description schema, which must be stored in the Profile Schema database. The semantic service description can be encoded in any ontology description language such as DAML[6] and OWL[7]. We decided to use OWL to create our semantic description schemas. We note that the schemas used by a service requestor do not need to be the same as those used by service providers, because the semantic matchmaking tool will reason and reconcile schemas first and then match instance descriptions.

Figure 2

With the extended version of UDDI, ontology matching techniques can be employed to discover services by reasoning over semantic relations between the service query and service descriptions. However, the computational burden increases as the number of service descriptions increases. Therefore, we developed a two-phase matching approach: category-based key-value matching and ontology matching. The key-value matching algorithm rapidly selects a small number of candidate services, and the ontology-matching algorithm precisely determines the best service from among the candidate services. As in the existing UDDI registry, the key-value matching will utilize *tModel* key-values. One advanced feature of our proposed approach is that the algorithm will return all services having related category codes. If a service query, for instance, specifies only ‘NAICS 331’, then all services having ‘NAICS 331xx’ will be candidate services. For the ontology matching, we revise and extend a DL (Description Logic)-based approach (Anicic *et al*. 2005), which computes the logical subsumability and consistency among concepts from a service query and those from

[6] DARPA Agent Markup Language, http://www.daml.org
[7] Web Ontology Language, http://www.w3c.org/TR/wol-ref

candidate service descriptions. The next section details how to define an OWL schema and how to generate a service description for a manufacturing service, and how to reason such descriptions to discover the most appropriate one for a particular query.

5. Manufacturing service advertisement and discovery

5.1 Overview

The overall procedure for advertising a semantic service description and for discovering a service is depicted in Figure 3. The service advertisement procedure involves ontology construction and service publication; the service discovery procedure involves comparisons between concepts in service descriptions and in a query. In particular, when creating a service query, a service requestor may reuse the ontology schemas given by service providers or define his/her own schema.

Figure 3

5.2 Service advertisement using web ontology language (OWL)

We chose OWL to describe the manufacturing service capability profile because (1) it is the standard markup language for expressing semantics for semantics web ontologies and (2) it provides powerful reasoning capability among ontologies. OWL is designed for use in applications in which the content of information must be processed rather than just presented to humans. OWL has two important features not found in XML and RDF(-S): additional vocabulary and formal semantics. Depending on the expressivity and computational complexity, OWL has three increasingly expressive sublanguages: OWL Lite, OWL DL, and OWL Full. OWL Lite, which uses a limited set of OWL constructs, supports users primarily needing a classification hierarchy and simple constraints. OWL Full supports maximum expressiveness and the syntactic freedom of using RDF constructs without computational guarantees. Meanwhile, OWL DL is an intermediate representation that emphasizes computational completeness and decidability while retaining the expressiveness of OWL Full as much as possible. The vocabulary in OWL DL corresponds to that of Description Logics. It uses a full vocabulary, but with restrictions: a pair-wise separation between classes, datatypes, datatype properties, object properties, and so forth. Such constraints are required for DL-based reasoning and inference - subsumption and consistency checking. We chose OWL DL to describe the capability profile of a manufacturing service.

The basic process of defining and generating a manufacturing capability profile consists of ontology design, ontology vocabulary definition, and ontology instantiation. In OWL, vocabulary definition is roughly made up of defining object classes (or resources in RDF), properties such as relations between classes/resources, and statements about those properties. The process for constructing ontological service descriptions is given below.

1. Design conceptual ontology models for manufacturing services using well-known format such as UML diagrams or EXPRESS-G expressions. The ontology model defines the objects used in the profile, specifies a set of properties of for those objects, and designates explicit relations among those objects and properties. It is critical that the objects span a domain of interest – in this case manufacturing - in order to make them reusable. We recommend that existing ontologies be used whenever possible to encourage wide dissemination. Although such ontologies may not cover the full capability of a particular provider, it is recommended to extend them or at least to explicitly reference them with annotations, rather than build a new ontology.
2. Create OWL Schemas (i.e., concepts, properties, and relations) for those ontology models.
    a. Define classes for objects using constructs of *owl*:*Class* and *rdfs*:*subClassOf*. We can also define more complex and expressive classes using other constructs of *owl*:*intersectionOf*, *owl*:*unionOf*, *owl*:*complementOf*, *owl*:*oneOf*, and *owl*:*disjointWith*. As usual, more general classes must be defined first to be inherited by more specific ones.
    b. Define properties for objects using constructs of *owl*:*ObjectProperty*, *owl*:*DatatypeProperty*, *rdfs*:*subPropertyOf*, *rdfs*:*domain*, and *rdfs*:*range*. A property is constructed by a name, and it can have multiple domains and ranges. A *domain* asserts that the *property* only applies to instances of the class expression associated with the *domain*, while a *range* asserts that the *property* only assumes values that are instances of the class expression restricted by the *range*. Each property is characterized by the constructs of *owl*:*TransitiveProperty*, *owl*:*SymmetricProperty*, *owl*:*FunctionalProperty*, *owl*:*inverseOf*, and *owl*:*InverseFunctionalProperty*; and is restricted by the constructs of *owl*:*allValuesFrom*, *owl*:*someValuesFrom*, *owl*:*cardinality*, and *owl*:*hasValue*. Notice that the data types used in OWL are the same as those defined in XML Schema (i.e., predefined XML Schema DataType[8]).

[8] http://www.w3.org/TR/xmlschema11-2

   c. Map those classes and properties using constructs of *owl*:*equivalentClass*, *owl*:*equivalentProperty*, and *owl*:*sameAs* (in OWL Full). The mapping is established by a triple (*Subject*, *Predicate*, *Object*) as a logical formula *Predicate*(*Subject*, *Object*), where the binary predicate *Predicate* (i.e., properties) relates the object/class *Subject* to the object *Object*. In fact, OWL (RDF as well) offers only *binary predicates*(*properties*).

3. Generate an OWL instance/individual for a manufacturing service. The individuals can also map to others using constructs of *owl*:*sameAs*, *owl*:*differentFrom*, and *owl*:*AllDifferent*.

An OWL ontology (both OWL schemas and individuals) has the *rdf*:*RDF* root element that also declares a number of namespaces. For example:

```
     <rdf:RDF
xmlns:owl  = “http://www.w3.org/2002/07/owl#”
xmlns:rdf  = “http://www.w3.org/1999/02/22-rdf-syntax-ns#”
xmlns:rdfs = “http://www.w3.org/2000/01/rdf-schema#”
xmlns:xsd  = “http://www.w3.org/2001/XMLSchema#”
```

In addition, an ontology document starts with an ontology header comprised of a collection of assertions for housekeeping purposes. These assertions are grouped under an *owl*:*Ontology* element that contains comments (i.e., *rdfs*:*comment*), version control (i.e., *owl*:*priorVersion*) and inclusion of other ontologies (i.e. *owl*:*import*). For example:

```
     <owl:Ontology rdf:about="">
<rdfs:comment>A sample OWL ontology</rdfs:comments>
<owl:priorVersion rdf:resource=”http://uni-ns-old”/>
<owl:import rdf:resource=”http://other-ontology”/>
<rdfs:label>Sample Ontology</rdfs:label>
</owl:Ontology>
```

For more detailed syntaxes and their semantics and usage, refer to (Antoniou and van Harmelen 2004)

After generating such OWL schemas and individuals as the capability profile of a manufacturing service, the service provider publishes his/her service on the UDDI registry. First, the provider registers meta-information about his/her service – including service name, service category, and contact information - in the same way as registering them to the original UDDI registry. Then, (s)he uploads that service profile onto the *serviceProfile* element. It is important to make the OWL schema of the service profile accessible to service requestors.

5.3 Service discovery through DL-reasoning

In the previous section, we proposed a two-phase matching approach: category-based key-value matching followed by ontology matching. The key-value matching is analogous to that supported by the original UDDI registry, the ontology matching is new. Here we focus more on the latter ontology matching based on DL-reasoning.

5.3.1 DL-reasoning

The use of OWL DL implies the ability to reason among (possibly different) ontologies with the support of powerful inference logic, namely Description Logic[9]. The intention of OWL DL is both to provide a language that can be used to formalize a domain by defining classes and their properties as well as to define individuals with asserted properties. More importantly, OWL DL supports reasoning, based on subsumption and consistency, about classes and individuals that are instances of classes. Subsumption is the basic inference technique for concept expressions in OWL reasoning, typically written as $C \subseteq D$. Determining subsumption means to verify that a concept denoted by *D* (subsumer) is considered more general than one denoted by *C* (subsumee). In other words, subsumption checks whether the subsumee always denotes a subset of the set denoted by the subsumer. Determining consistency means two things: finding any inconsistencies that might exist among concepts within the ontology, and finding any individuals that belong to a concept whose restrictions conflict with other individuals belonging to that concept.

OWL itself does not have the capability to perform reasoning based on subsumption and consistency checking. To do this, we can use the DL reasoner. DL is a frame-based formalism for representing knowledge. It is based on the notion of concepts and roles, and is characterized mainly by constructors that allow complex concepts and roles to be built from atomic ones. The main benefit of using DL is that there exists sound and complete algorithms to solve subsumption and satisfiability problems. A few DL-reasoners have been studied and implemented in academia, such as Racer (Haarslev and Moller 2001), FaCT++ (Horrocks *et al.* 1999), and Pellet (Sirin and Parsia 2004). Here we employ Pellet, developed at the University of Maryland, as our starting reasoning tool because it is fully capable of processing OWL-DL expressivity (e.g., *owl*:*oneOf*, *owl*:*hasValue*). Pellet is a hybrid DL-

[9] http://dl.kr.org

reasoner that can deal with both non-empty ABox reasoning and TBox reasoning, based on the tableaux algorithm.

5.3.2 Service discovery through semantic matchmaking

The first phase of matchmaking is category-based key-value matching, which selects a small number of candidate services whose classification codes match those contained in the query. We note that ‘matched’ does not imply that two codes are lexically identical. Rather, it means that they are related - one code is more or less general than the other. This feature prevents relevant services from being filtered out due to mere differences in classification code configurations. In the second phase, subsumption reasoning and consistency reasoning are performed successively between a service query and the candidate services selected in the first phase. We use the reasoner to perform the subsumption and consistency checking to determine the actual matches (see Figure 4).

Figure 4

5.4 Implementation of semantic service advertisement and discovery

As part of the present research, we also design the implementation architecture for registering services with their capability profiles and discovering such services based on ontology reasoning (see Figure 5). The left part of the figure shows the service advertisement module; and, the right part shows the service discovery modules. The advertisement module receives manufacturing service capability profiles represented as OWL individuals, validates them against their corresponding OWL schemas, and then stores them in the *serviceProfile* of the UDDI registry. The discovery part consists of the key-value matching module, the communication module with an external DL-reasoner, and the interface module with service requestors to receive a service query and to show the match result.

Figure 5

6. Case study in discrete part manufacturing

This section demonstrates a simple scenario to advertise a manufacturing service in OWL and to discover it through a semantic matchmaking process. In examining this example, we also highlight and reiterate the limitations of the original UDDI registry.

6.1 Illustrative example

Let us consider a scenario in which a manufacturer provides a Hole-Making service. Key parts of the service description and service query are shown in Figure 6. The service provider, a manufacturer, describes and publishes its Hole-Making service; the service requestor seeks a service provider who offers a Drilling service.

Figure 6

The service description for the Hole-Making service can be represented using the original UDDI data structure, as shown in Figure 7. The *businessEntity* element has the *businessKey* attributes as a unique identifier. Since the UDDI specification by itself cannot describe the details of the service capabilities in a computer-interpretable way, the service capabilities are annotated in the *description* element of the *businessEntity* element. Alternatively, it is possible to insert a pointer into the UDDI registry that directs the user to a URL that gives the capability information. The UDDI allows the service type to be specified using a classification code such as NAICS or UNSPSC within the *categoryBag* element. However, no classification code exists for the service type ‘Hole Making’; hence, the service provider would have to use the closest code. For example, NAICS 311, which indicates ‘Primary Metal Manufacturing’ according to NAICS 2002, could be used. The provider may, additionally or alternatively, specify other items in the *categoryBag* element. For example, NAICS 33111 for ‘Iron and Steel Mill and Ferroalloy Manufacturing’ and/or NAICS 33131 for ‘Alumina and Aluminum Production and Processing’. The same situation would apply using the UNSPSC specification codes. This example highlights the inability of the UDDI specification to capture and describe manufacturing service capabilities in sufficient detail. This limitation will affect not just manufacturing services, but any service domain where the business is characterized by specialized service capabilities.

Figure 7

In attempting to discover a service, the service requestors may try to find service providers by using key-value pairs, for instance, either *businessService*/*name*=”Drilling” or *businessService*/*categoryBag*/*keyValue*=”33131”. In the above situation, the service requestor cannot find the service provider ‘ISL Inc.’ unless the provider specifies multiple *categoryBag* elements with NAICS 33131. Thus, even though Drilling is a type of Hole-Making operation, the service requestor cannot find the service provider. We noted further, that service providers and requestors must use the same language in this case the same classification coding scheme. More importantly, since the UDDI framework supports discovery based on high-level service information only, the requestor would fail to find services offering specific service capabilities - such as those related to hole diameter and allowable material type. Therefore, the detailed information about a service contained in *description* is practically useless for automated service discovery.

6.2 Service description in OWL

For demonstration purposes, here we consider only a part of the ontologies necessary for defining the manufacturing service capabilities. As outlined in Section 5.2, the first step is to construct the service capability ontology, as depicted in Figure 8. The ontology shows that the *HoleMaking* process is a subtype of the *MaterialRemoval* process and related to the Drilling process. It consumes *availableMaterial* whose volume is less than *maxMaterialSize* and it can produce a hole with a diameter larger than *minHoleDiameter*. Other information is also augmented in the same way. We note that this ontology is based on the work described in (Kulvatunyou *et al*. 2005).

Figure 8

With the ontology above, we construct an exemplary OWL schema for the service provider, as shown in Figure 9. This schema mainly consists of *classes* (i.e., *owl*:*Class*) and *properties* (i.e., *owl*:*ObjectProperty*). A service requestor may reuse this schema or define his/her own schema.

Figure 9

The OWL schema allows a service requestor to precisely discover a service provider with a specific manufacturing process type and capability. For instance, one could search for a service provider that can handle a work-piece of dimension 500×500×500 mm^3. In addition, because the *Drilling* class is a subclass of *HoleMaking*, a query for the term *Drilling* would discover all providers that advertise themselves as offering a *HoleMaking.* This demonstrates the advantage of using an ontology language to describe the service profile. The instance includes specific capabilities and relationships with other ontologies. Using the OWL Schema (in Figure 9), we generate a sample OWL individual of the *HoleMaking* service description in Figure 6, as shown in Figure 10. This OWL individual is recorded in the *serviceProfile* entity when the *HoleMaking* service is advertised.

Figure 10

6.3 DL-based service discovery

Due to space limitations, here we simplify and/or eliminate unnecessary parts of the XML (OWL) documents. First, the service provider represents the hole-making service using the OWL representation and registers it on the extended UDDI registry. If no agreed-upon OWL schema is publicly available, the provider should encode his/her own schema and make it accessible to others. In the same way, the service requestor also encodes his/her service query in OWL. The service requestor may use the same available schema provided by service providers or define his/her own schema. Figure 11 illustrates the OWL schemas for the *HoleMaking* process (of the service provider) and the *Drilling* process (of the service requestor).

Figure 11

The service provider represents available materials in the *HoleMakingProcess* class as an enumerated data type, which is a constructor for defining a range of data values in OWL. Additional properties of the class are also represented, but it is noted that a *Drilling* class is not defined. In the second schema (for *Drilling*), the service requestor does not explicitly define the *Drilling* class to be an equivalent class to the *HoleMakingProcess* class, nor does he/she define it to be a subclass of the *HoleMakingProcess* class. Without reasoning, the requestor could not determine that the *HoleMakingProcess* subsumes *Drilling* and that these two classes are related semantically. Fortunately, the subsumption reasoning detects their relationship, as listed in Figure 12.

Figure 12

Finally, to check whether the hole-making service satisfies the service query in terms of its capabilities, consistency reasoning between the *HoleMakingProcess* class and an individual of the *Drilling* class should be performed. Figure 13 shows a simple OWL instance for the *Drilling* ontology. The *Drilling1* individual has only one available material, aluminum. We can determine that the individual is consistent with the *HoleMakingProcess* class, because the *Drilling1* individual has only one *availableMaterial* property and the property has a string value of ‘aluminum’, which is consistent with the enumerated data of the *HoleMakingProcess* class. Through the consequent subsumption and consistency reasonings, we can determine that the hole-making service meets the requirements of the service query.

Figure 13

We implemented a prototype system by extending UDDI[10] to be interactive with the Pellet DL-reasoner (Sirin and Parsia 2004) as well as to have a *serviceProfile* container. This prototype implementation brought to light two key limitations of the proposed methodology. First, the computation time for OWL document validation and ontology reasoning is too long. Second, UDDI users must be fully conversant with OWL syntax to achieve efficient discovery. In particular, the computation time required for reasoning a description on the average requires less than 0.01 seconds in our experiments (with an IBM PC equipped with 1.2 GHz CPU and 1 GB RAM), but this matters for a large number of complicated service descriptions. In other words, the time complexity exponentially increases according to the size of service descriptions and the complexity of a service description. These limitations must be resolved if the UDDI extension framework is to enjoy wide popularity.

The presented work is a cumulative one of previous methodologies in the literature, but also a unique and advanced one. We used an extended UDDI registry to mediate contracts between designers and manufacturers, the OWL language to augment manufacturer's capability profiles, and DL-reasoning to discover such profiles. Particularly, for example, the use of the public registry resolves the difficulty to associate service profiles Kulvauntyou *et al.* (2005). The simplified extension (i.e., attachment of *serviceProfile*) keeps inordinate *tModel*s from being defined (c.f., Luo *et al.* (2005) and Paolucci *et al.* (2002)), thereby enabling a DL-reasoner to support straightforward inference, as well as obtains backward compatibility with existing registries. Moreover, the two-phase discovery would provide a practical solution to the long time reasoning.

7. Conclusion

In the last decade, industries throughout the world have been challenged by a new business paradigm – doing business with virtual partners over the Internet. Recent advances in Internet technologies such as XML, SOA, Web Service, and ebXML have made it possible to realize loosely integrated manufacturing. In particular, SOA provides a robust integration model and UDDI acts as a broker between service providers and requestors. However, the weaknesses of the current UDDI registry make it impossible to discover and retrieve services based on semantic information (e.g., manufacturing capabilities).

Many approaches that exploit ontology languages for web services still do not have an ontology that can describe manufacturing service capabilities. We have extended the UDDI registry specification to enable semantics-based service advertisement and discovery.

[10] http://ws.apache.org/juddi

Specifically, we have added a new semantics container, the *serviceProfile*, and described how to represent and to reason about manufacturing service capabilities in a semantic manner; in particular, we used OWL and DL for the latter purpose. The extensions to the UDDI specification and implementation described here could be used to advertise and discover services (and providers as well) in domains other than manufacturing.

In future work, we plan to further extend UDDI for dynamic interaction and invocation of web services. As indicated, both service providers and requestors must have full knowledge about the ontology description language (i.e., OWL), and the efficiency (i.e., discovery time) must be improved. In addition, there is an urgent need to categorize formally the information necessary to construct manufacturing service capability ontologies. Moreover, the naïve extension to the UDDI registry should be further standardized.

Disclaimer

Certain commercial software products are identified in this paper. These products were used only for demonstration purposes. This use does not imply approval or endorsement by NIST, nor does it imply that these products are necessarily the best available for the purpose.

Reference

Akkiraju, R. Goodwin, R., Doshi, P., and Roeder, S., A method for semantically enhancing the service discovery capabilities of UDDI. Paper presented at the Workshop on Information Integration on the Web, Mexico, August 9-10, 2003.

Anicic, N., Ivezic, N., and Jones, A., An architecture for semantic enterprise application integration standards. Paper presented at the 1st International Conference on the Interoperability of Enterprise Software and Applications (I-ESA'05), Geneva, Switzerland, February 23-25, 2005.

Antoniou, G. and van Harmele, F., A Semantic Web Primer, 2004 (Tarrytown, NY: MIT Press).

Clement, L, Hately, A., von Riegen, C., and Rogers, T., Universal description discovery and integration (UDDI) Ver. 3.0.2. Available online at: http://uddi.org/pubs/uddi v3.htm (accessed March 2005).

Dogac, A., Kabak, Y., and Laleci, G., Enriching ebXML registries with OWL ontologies for efficient service discovery. Paper presented at Research Issues in Data Engineering (RIDE2004), Boston, MA, March 2004.

Haarslev, V. and Moller, R., Description of the RACER system and its applications. Paper presented at the International Workshop in Description Logics, San Francisco, CA, August 1-3, 2001.

Horrocks, I., Sattler, U., and Tobies, S., Practical reasoning for expressive description logics. Paper presented at the 6th International Conference on Logic Programming and Automated Reasoning, Tbilisi, GA, September 1999.

Jang, J., Jeong, B., Cho, H., and Lee, J., Capability and extension of UDDI framework for semantic enterprise integration. Paper presented at International Conference on Advances in Production Management Systems, Rockville, MD, September 2005.

Jeong, B., Kulvatunyou, B., Ivezic, N., Cho, H., and Jones, A., Enhance reuse of standard e-business XML schema documents. Paper presented at International Workshop on Contexts and Ontology: Theory, Practice and Applications (C&O'05) in the 20th National Conference on Artificial Intelligence (AAAI'05), Pittsburgh, PA, July 9, 2005.

Kevin, K., James, E., and Mary E.A., Modeling of manufacturing resource information. 1995, NISTIR 5707, National Institute of Standards and Technology (NIST).

Kulvatunyou, B., Cho, H., and Son, Y., A semantic web service framework to support intelligent distributed manufacturing. *International Journal of Knowledgebased and Intelligent Engineering Systems*, 2005, **9**, 107-127.

Luo, J., Montrose, B., and Kang, M., An approach for semantic query processing with UDDI. Paper presented at On the Move to Meaningful Internet Systems 2005: OTM Workshops, LNCS 3762, Agia Napa, Cyprus, October 31 - November 4, 2005.

Madhavan, J., Bernstein, P.A., and Rahm, E., Generic schema matching with Cupid. Paper presented at the 27th International Conference on Very Large Data Base (VLDB), Rome, Italy, September 11-14, 2001.

Melnik, S., Garcia-Molina, H., and Rahm, E., Similarity flooding: A versatile graph matching algorithm. Paper presented at the 18th International Conference on Data Engineering (ICDE), San Jose, CA, 2002.

Noy, N.F. and Musen, M.A., PROMP T: Algorithm and tool for automated ontology merging and alignment. Paper presented at the 17th National Conference on Artificial Intelligence (AAAI-2000), Austin, TX, 2000.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K., Importing the semantic web in UDDI. Paper presented at Web Services, e-Business and Semantic Web Workshop (WES2002), tronto, Canada, May 2002.

Pokraev, S., Koolwaaij, J., and Wibbels, M., Extending UDDI with context-aware features based on semantic service descriptions. Paper presented at International Conference on Web Services (ICWS2003), Las Vegas, NV, June 2003.

Schlenoff, C., Gruninger, M., Tissot, F., Valois, J., Lubell, J, and Lee, J., The process specification language (PSL): Overview and version 1.0 specifications., 2000, NISTIR 6759, National Institute of Standards and Technology (NIST).

ShaikhAli, A., Rana, O., Al-Ali, R., and Walker, D., UDDIe: An extended registry for web services. Paper presented at the Workshop on Service Oriented Computing: Models, Architectures and Applications at SAINT Conference, Orlando, FL, January 27-31, 2003.

Sirin, E. and Parsia, B., Pellet: An OWL DL reasoner. Paper presented at the 3rd International Semantic Web Conference (ISWC2004), Hiroshima, Japan, November 2004.

Srinivasan, N., Paolucci, M., and Sycara, K., Adding OWL-S to UDDI, implementation and throughput. Paper presented at the 1st International Workshop on Semantic Web Services and Web Process Composition (SWSWPC2004), San Diego, CA, July 2004.

Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N., Automated discovery, interaction and composition of semantic web services. *Journal of Web Semantics: Science, Services and Agents on the World Wide Web*, **1**(1), 2003.