

Dynamic Integration of Distributed Semantic Services

Guofei Jiang, Wayne Chung and George Cybenko
 Institute for Security Technology Studies and
 Thayer School of Engineering, Dartmouth College
 Hanover, NH 03755
 Email: *firstname.lastname@dartmouth.edu*

Abstract- *Global networking is changing the way that we think about and perform computation. Network-based computing may link tens or hundreds of distributed devices, sensors and computing resources to support an application. Therefore, a critical challenge is how dynamically to discover and integrate these distributed services seamlessly for various applications. In this paper, many cutting-edge technologies including semantic web, web services, peer-to-peer network and content-based routing, are used to address this challenge. With these technologies, we propose a new framework for dynamic integration of distributed services. Moreover, a prototype system with basic capabilities is implemented as a proof-of-concept to demonstrate the potential of this framework and its constituent technologies in network-based computing.*

systems to be dynamically interfaced in a networked environment. With these cutting-edge technologies, we developed a new framework for the dynamic integration of distributed services. Semantic Web [4] technology is used to describe, locate and compose distributed semantic services in the framework. Distributed computing and data resources are organized with a peer-to-peer (P2P) network for scalability and reliability. Web services and content-based routing are employed to dynamically integrate distributed services across the network. Moreover, we implemented an extensible prototype system with basic capabilities as a proof-of-concept to demonstrate the potential of this framework and its constituent technologies in network-based computing. This prototype system is able to dynamically discover, compose, and integrate these distributed services into new applications.

1. INTRODUCTION

The immense success of the Internet has led to great interests in network-based computing such as Grid [1] and Pervasive [2] computing. Global networking is changing the way that we think about and perform computation. Grid computing refers to large-scale computation in a distributed environment where computing and data resources are located throughout a network. Pervasive computing is about a physical environment where ubiquitous devices are networked, anthropocentric and interacting with each other to improve the quality of human life. Both Grid computing and Pervasive computing may link tens or hundreds of distributed devices, sensors and computing resources to support an application cooperatively. Therefore, one critical challenge is how dynamically to discover and integrate these distributed resources seamlessly for various applications.

In recent years, many distributed computing technologies are developed to support such network-based computing. To improve semantic interoperability among distributed systems, markup languages such as eXtensible Markup Language (XML) and DARPA Agent Markup Language (DAML) [3] are used to create ontologies and describe the semantics of data. Loosely coupled distributed computing technologies such as web services enable distributed

2. DISTRIBUTED SERVICES

In this section, we first discuss some fundamental capabilities needed in network-based computing. Then we introduce the prototype system and its components that support these capabilities.

2.1 Semantic Services

Distributed services include data services and computing services. Distributed data services are sensors, databases and other data resources. Distributed computing services refer to the data processing and computing resources such as fuselets in our framework. Fuselets are lightweight data fusion algorithms that can be deployed in an environment as computing services. A distributed service network consists of these services and can be viewed as an overlay network built on Internet. Since these services are not centrally administrated and may come and leave the network asynchronously, peer-to-peer networking is used to organize the service network for scalability and reliability. In order to be dynamically discovered during composition, both data and computing services' capabilities have to be semantically described with certain ontologies and be published in a knowledge base or directory service for service brokering. Additionally, service interfaces such as communication protocols and input parameters have to be explicitly described for dynamic service invocation. In our framework, distributed services are implemented as web services, which use Web Service Description Language (WSDL) to describe their programming interfaces. However, WSDL does not include any semantic

information on service, parameters or conditions in its description. For example, what does a parameter constitute for that service? In our prototype, DAML-S [4] ontology is employed to describe the semantics of services using WSDL as service grounding.

DAML-S is a set of service ontologies to establish a framework within which the semantics of web services may be described. DAML-S partitions a semantic web service description into three components: service profile, process model, and service grounding. Service profile describes what the service does by specifying the input and output types, preconditions, and effects. Process model describes how the service works. Each service is either an AtomicProcess that is executed directly or a CompositeProcess that is a combination of other sub-processes. Service grounding contains the details of how an agent can access a service by specifying the communications protocol, parameters to be used in the protocol, and the serialization techniques to be employed for communications. We regard the distributed services with semantic description as distributed semantic services. The use of semantics leaves the environment open to expansion, without having to create interaction models specific for each service [5].

2.2 Prototype System

We implemented a prototype system and a ground vehicle tracking application as a proof-of-concept to demonstrate how these technologies can be applied in dynamic integration of distributed services. The architecture of our prototype system is shown in Figure 1.

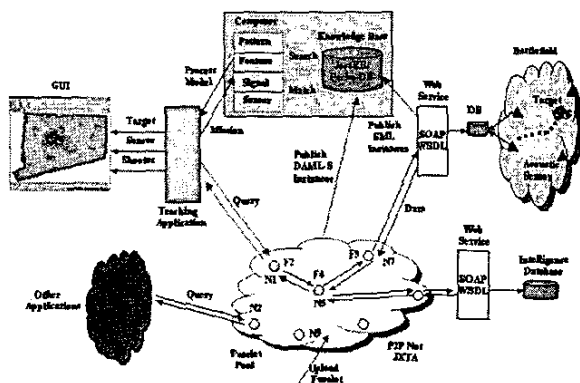


Figure 1: Prototype System Architecture

Distributed data services: Several acoustic sensors are implemented to support our ground-vehicle tracking scenario. A hierarchical sensor ontology and acoustic sensor ontology are created with DAML to describe sensor properties. Sensors' capabilities are marked up with these ontologies and can be dynamically discovered by non-functional property searches during service brokering. Sensor web services are used as "wrappers" to interface

other services with sensors and are regarded as distributed data services in our system. With Simple Object Access Protocol (SOAP) and WSDL technologies, sensor web services can be dynamically invoked by other services in the network. In addition, sensor web services are marked up with the DAML-S ontology using WSDL as the service grounding. Sensor's non-functional properties are included in the "service profile" part of DAML-S ontology and their DAML-S instances are published in a knowledge base for service brokering.

Distributed computing services: Fuselets are reusable and shareable computing services developed for data fusion specifically. Fuselets are hosted on distributed computing servers across the network and these servers are organized as a peer-to-peer service network in order to achieve reliability and scalability. The P2P network includes both computing services and data services in our system and it is implemented on Sun Microsystem's JXTA platform. The content-based routing mechanism is designed to enable dynamic and reliable query routing in this P2P network. Several fuselets are developed to support our vehicle tracking application such as Band-pass filter and Root-Mean-Square (RMS) power calculator. These fuselets are described with DAML-S service ontology and their markup instances are published in the Knowledge Base (KB) for service brokering.

Composer and KB: The service composition system serves as a human-machine interface to combine a human's professional knowledge with a machine's searching and matching ability. It includes both a composer and an inference engine. The composer is the user interface that handles the communication between the human operator and the engine. The inference engine stores the information about the services in its knowledge base and has the capability to find matching services. An operator can use the composer to view the available services, browse their detailed information, filter the services by entering constraints on the non-functional properties and compose different services by chaining them.

Network-based Applications: A ground vehicle tracking application is implemented as an example to demonstrate the potential of our framework and its constituent technologies in network-based computing. The application can dynamically discover and assemble distributed services in the network to fulfill a target-tracking mission. In the following sections, we will use the context of this example to discuss related technologies.

3. SERVICE DECLARATION

The ability to interoperate services based on semantic description depends on the ability to create, use and manage shared ontologies of concepts and their relationships. Specifically, distributed services have to achieve a level of semantic agreement for service brokering and data processing. Without a common vocabulary, a machine has

no way to understand the terminology in the structured data. Building a system capable of semantic brokering requires a standard ontology to describe the resources in the network. If distributed services use different ontologies for service description, ontology mapping is needed to translate datasets represented by disparate ontologies.

In our system, we created a hierarchical sensor ontology using DAML to describe the capabilities of sensors. The general sensor ontology includes major properties such as location and sensing mechanism. Each major property can be specified with low-level properties. For example, the following DAML class describes a property named sensing mechanism. Every sensor has to choose at least one of eight sensing mechanisms to describe its property.

```
<daml:Class rdf:ID="SensingMechanismType">
  <daml:oneOf rdf:parseType="daml:collection">
    <SensingMechanismType rdf:ID="ElectricMagnetic"/>
    <SensingMechanismType rdf:ID="Mechanical"/>
    <SensingMechanismType rdf:ID="Biological"/>
    <SensingMechanismType rdf:ID="Chemical"/>
    <SensingMechanismType rdf:ID="Radioactive"/>
    <SensingMechanismType rdf:ID="Cyber"/>
    <SensingMechanismType rdf:ID="Optical"/>
    <SensingMechanismType rdf:ID="Other"/>
  </daml:oneOf>
</daml:Class>
```

The acoustic sensor ontology is a sub-class of sensor ontology and it includes many specific properties of acoustic sensors such as the following PickupPatternType property.

```
<daml:Class rdf:ID="AcousticSensor">
<daml:subClassOf rdf:resource=".../sensor.daml#Sensor"/>
</daml:Class>
.....
<daml:Class rdf:ID="PickupPatternType">
  <daml:oneOf rdf:parseType="daml:collection">
    <PickupPatternType rdf:ID="Omnidirectional"/>
    <PickupPatternType rdf:ID="Cardioid"/>
    <PickupPatternType rdf:ID="Hypercardioid"/>
    <PickupPatternType rdf:ID="Shotgun"/>
  </daml:oneOf>
</daml:Class>
.....
```

Distributed services are implemented as web services in our framework and WSDL is used to describe the programming interface of a service. In our prototype, the DAML-S ontology is employed to describe the semantics of services. As we mentioned above, sensor properties are marked up with a hierarchical sensor ontology and their instances are included in the "Service Profile" part of DAML-S. Sensor web service's WSDL is included in the "Grounding" part of DAML-S as the service grounding. The DAML-S instances of web services are published in the KB for service brokering.

Fuselets are the computing services in our system. Several fuselets are implemented for our tracking application, such as Band-pass filter and Root Mean Square (RMS) power calculator. These fuselets are described with the DAML-S ontology using WSDL as the service grounding and their DAML-S instances are published in the KB for service brokering. The following text is a part of the filter's DAML-S instance.

```
<!-- Service description -->
<service:Service rdf:ID="FilterService">
  <service:presents rdf:resource="#FilterProfile" />
  <service:describedBy rdf:resource="#FilterProcessModel" />
  <service:supports rdf:resource="#FilterGrounding" />
</service:Service>
.....
<!-- Profile description -->
<profile:input>
  <profile:ParameterDescription rdf:ID="SoundInput">
    <profile:parameterName>SoundInput
  </profile:parameterName>
  <profile:restrictedTo
    rdf:resource="http://localhost/sensor/dt.daml#wav"/>
  <profile:refersTo rdf:resource="#soundInput"/>
  </profile:ParameterDescription>
</profile:input>
.....
```

Though DAML-S and WSDL can describe the inputs and outputs of a service explicitly, it is hard to describe some implicit preconditions of a complicated service, such as the temporal and spatial relationships of the input parameters. For example, our multi-sensor fusion algorithm uses four inputs from four different acoustic sensors. These four inputs have to be time-synchronized and normalized in the same way. The sensors also have to be deployed with a specific topology. Moreover, service providers and service users may require different description granularity for the same service. Service providers want the service ontology to describe the low-level properties such as "Pickup Pattern Type". Their services are to be located and used for various purposes. Conversely, end users want the ontology to describe the high-level capabilities of services such as "Vehicle Tracking". As they are interested in what the service can do instead of their low-level properties. To assist the user, we have created a graphical service composition engine that hides much of the complexity in matching the services.

4. SERVICE COMPOSITION

A network-based computing environment may include thousands of distributed services. The semantic description of services could facilitate automatic discovery and composition by a software agent. However, for complicated tasks, we believe that a human needs to be in the loop to guide the composition process with specific domain knowledge. The service composition system in our framework provides a human-machine interface to combine a human's professional knowledge with a machine's

searching and matching ability. This system was implemented at the University of Maryland as a joint research effort [6].

The service composition system has two basic components: composer and inference engine. The inference engine stores the information about the services in its KB and has the capability to find matching services. The composer is the user interface that handles the communication between the human operator and the engine, which is shown in Figure 2. The inference engine is a DAML reasoner built on Prolog. Ontological information written in DAML is converted to RDF triples and loaded to the KB. The engine has the built-in axioms for DAML inferencing rules. These axioms are applied to the facts in the KB to find the entailments such as the class inheritance relation between two classes that are separated several steps away in the hierarchy tree. This system supports searching and matching of both functional and non-functional properties.

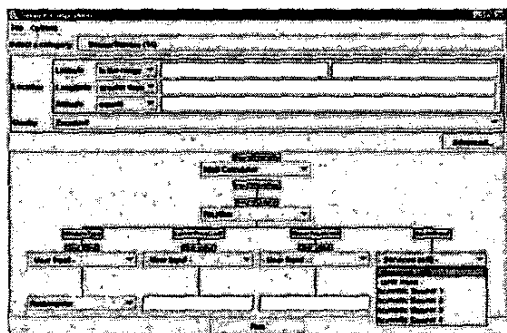


Figure 2: Service composer

The composition of semantic services in our framework is accomplished as follows: sensor and fuselet DAML-S descriptions are loaded to the inference engine and the composer program presents a user interface where an operator can view the available services, browse their detailed information, compose different services by chaining them and filtering the services by entering constraints on the non-functional attributes of the service such as the properties of its corresponding sensor. The composition process can start from top computing services to bottom data services or vice versa.

During the composition process, the composer allows the user to create a composition of services by presenting the available choices at each step. The user is first presented with all the available services registered with the engine. When a service is selected from the list, a query is sent to the KB to retrieve the information about the inputs of the service if any exist. For each of the inputs, a new query is run to get the list of the possible services that can supply the data for this input. The matching of chained services' input/output is done using the information from the service profiles. Each service profile describes its inputs, outputs and the range of these parameters. The composer also shows different types of services available in the system

and filters the results based on the constraints defined by the user on the attributes of a service. The matching accuracy of the inference engine relies on the granularity of ontologies.

After the composition, the composer outputs a query that outlines the sequence of chained services and their parameters. The query is submitted to the P2P network that uses content-based routing to deliver the query message. At each node along the route to the end sensors, the service information and parameter information is parsed from the query message and used to locate dynamically and load the expected service. The following is an example of the query message:

```
<s:Envelope
xmlns:s=http://schemas.xmlsoap.org/soap/envelope>
<s:Body>
<passMessage xmlns="urn:xml-soap-demo">
  <classname NameofClass="Service1">
    Method="Service1_Invoke">
      <Param>Service1_Param</Param>
      <classname NameofClass="Service2">
        Method="Service2_Invoke">
          <Param>Service2_Param</Param>
        </classname>
      </classname>
    </passMessage>
  </s:Body>
</s:Envelope>
```

The query is a SOAP request message that lists the sequence of chained services. SOAP defines the layout of the message and provides a simple method for query delivery. Each service consists of a tag specifying the service identifier and its parameters, with successive services embedded as input parameters to the previous service. In fact the service chain represents the recursive function calling process of services. As we see in the above message, the query itself only specifies the requested service names not the physical nodes running the services. Instead, these nodes are dynamically located during the routing process. The composed queries can be saved and reused by other users as a composite process in DAML-S.

While the current service composition system can greatly aid operators to compose semantic services, we believe that a high-level integrated development environment (IDE) needs to be developed for service composition. Service discovery, service composition, service validation, service monitoring and debugging etc. should be integrated in this IDE. Moreover, in some cases, the composer may only be able to compose an incomplete workflow based on existing services and the operator has to insert his own logic. Therefore the IDE needs to integrate developer's programming environment with high-level service composition.

5. SERVICE EXECUTION

All of the services in our framework are implemented as web services and are hosted on distributed servers. These web services are described with DAML-S and WSDL and can be remotely invoked by SOAP request messages. The distributed servers are organized with a P2P network to achieve scalability and reliability. After the composer sends the query to the P2P service network, it is the content-based routing mechanism that drives the query across the network and invokes service execution sequentially.

The P2P network is implemented with the JXTA platform. Each server node in the network includes three major components: the gateway module, the service module and the routing module. The gateway module is responsible for receiving queries, parsing queries, forwarding queries and data. The service module is responsible for dynamic loading, invoking and managing services on the server. The routing module is responsible for discovering new peers, updating its routing table and maintaining the P2P network. Once a new service is to be deployed in the environment, its DAML-S instance is published in the Knowledge Base for service brokering. The code for this service is uploaded to multiple nodes in the environment. The routing module of these nodes distribute the indexing information of this service to their peers and then every node's routing table is updated. The content-based routing table maps each service name to a list of nodes that host the service. For each service, the routing module uses specific routing policies to select one best node from the list to run that service. The following is an abstract format of the routing table:

```
Service1: Node1, Node4, Node6
Service2: Node2, Node3
.....
ServiceN: Node10, Node81, Node132, Node4
```

As we mentioned in Section 4, the composite query consists of a series of specially formatted XML tags and is passed using SOAP messages. As shown in Figure 3, the query routing process is like the recursive function calling process in a program. S represents services and N represents physical nodes in the network. The dot line represents the query forwarding process in the network while the solid line represents the data processing and return process. The composer can submit a query to any node in the network. After an application node receives the composite query, it searches its content-based routing table to discover which node has the first service in the query. If the current node has the service, it removes the first service from the composite query and forwards the rest of the query to a next node, and waits for a response. If the current node does not have the next requested service, it forwards the query directly to a next node that has the service. The next node continues the same routing logic. Once the response is received, it is used as a parameter to invoke the service. The computing result is enclosed in a SOAP message as an attachment and sent back to the previous calling node. At

one point, if one node splits the query to several parallel sub-queries and forwards them to several different nodes, multi-threads are created to wait for the responses from these nodes. Only after all responses are received as parameters, the service on the current node is invoked.

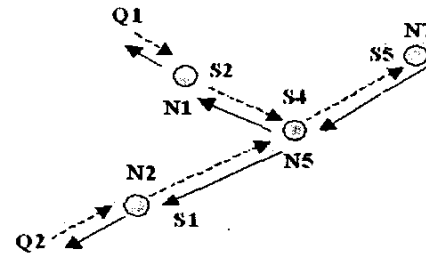


Figure 3: query routing process

Queries with common sub-expressions can be dynamically merged along the route to the sensors so that the data stream will not be pulled and processed multiple times along the route back to the clients. For example, in Figure 3, we have query Q1: $s2 \rightarrow s4 \rightarrow s5$ and query Q2: $s1 \rightarrow s4 \rightarrow s5$. Their sub-query $s4 \rightarrow s5$ can be dynamically merged at the node N5 during the routing process. This dynamic content-based routing process can dramatically reduce both network traffic and computational loads in the network. See details on query routing in [7].

The service module at each node consists of a dynamic loader and a pool of services. The dynamic loader relies on Java reflection and is able to load classes on demand. After a node receives a query, it uses the service identifier, in our case the name of the requested service, to invoke service execution. The dynamic loader is responsible for loading the service code and invoking the service execution. With reflection, the dynamic loader casts a generic class object and a generic method list to match the class and method lists specified in the query respectively. Once the class is instantiated and the method is determined, reflection is invoked and the necessary parameters are passed to the call. As long as the inputs and outputs of a fuselet are kept consistent, the underlying methods and implementation can be modified at will.

Services are not centrally administrated and may join and leave the network asynchronously. The query itself only specifies the requested service names but not the physical nodes running the services. Instead, the requested services are dynamically assigned to active physical nodes along the routing paths. If one node is offline, it can be replaced by another node that has the same service. This content-based routing mechanism increases the reliability and scalability of our framework. However, we believe that an advanced query language is needed to describe the composite process as opposed to using the specific, rigid and uninformative tags. Meanwhile, since query latency originates from both data communication and service computation along the route, an advanced routing algorithm is needed to optimize the routing process [7].

6. APPLICATIONS

In the above sections, we discussed the technologies used for service declaration, service composition and service execution. In this section, we analyze how these technologies can be applied to integrate distributed services into various applications. With multiple acoustic sensors, a ground vehicle tracking application was implemented as a proof-of-concept in order to demonstrate the feasibility of our framework and its constituent technologies.

For a specific application, an operator uses the composer to search the knowledge base and discover the sensors capable of sensing the event. For example, in our ground vehicle tracking application, acoustic sensors are discovered in the desired location. With his specific expertise, the operator knows that he can accomplish the tracking goal by fusing the data from several sensors in that field. The raw data from these sensors has to be processed via signal processing services in the P2P network. With the composer, the operator searches fuselet services in the network and creates a composite query by chaining services. After the composition, the composer outputs a SOAP message that lists the chained services, parameters and their sequences.

The composer submits the SOAP message to the P2P service network. Content-based routing drives the query message across the service network and the sensor web services are invoked at the end. A data stream is pulled from the sensor and enclosed in a SOAP message as a data attachment. Then the returned data message is sequentially routed back through signal processing fuselets for data processing. Each service processes the data and then delivers the result to the next service. Eventually the result is routed back to the operator's application. In this way, distributed semantic services can be dynamically discovered, composed and executed to support a specific application.

In our tracking application, multiple acoustic sensors acquire the sound signal of a ground vehicle in the field. Several distributed signal-processing services were developed to process sensor's data, such as Band-pass filter and RMS power calculator. The moving target's position is tracked based on the acquired signal power and the sensor's location. In our experiments, 12 distributed nodes and 4 sensor assets were successfully integrated within our framework to support the tracking application. Since the sensor's data can be used by various applications, it needs to have multiple data resolutions to satisfy different needs in the environment. Meanwhile, for network-based computing, the sensor's sampling rate must be considered along with the network's bandwidth and latency. High sampling rate could generate a large amount of data that cannot be sent across the network to the end applications in time.

7. CONCLUSIONS

This paper proposes a promising framework for dynamic integration of distributed semantic services in network-based computing. The framework includes cutting-edge technologies for service description, discovery, composition and execution. Distributed services are implemented as web services and are described with Semantic Web technology for service brokering. Distributed servers are organized with peer-to-peer overlay network in order to achieve scalability and reliability. Content-based routing is used to drive the query across the network to invoke and execute services sequentially. While these technologies are independently researched by many literatures, we have not seen many similar works on how to integrate these technologies together for network-based computing.

Acknowledgements: Evren Sirin and Jim Hendler at UMD implemented the composer. At Dartmouth, Annarita Giani, Yong Sheng, Glenn T. Nofsinger, Han Li, Diego Hernando, Paul Thompson participated in the implementation of the prototype system. The authors are grateful for their contributions. This research was partially supported by: DARPA projects F30602-00-2-0585 and F30602-98-2-0107; National Institute of Justice, Department of Justice award number 2000-DT-CX-K001.

REFERENCES

- [1] I. Foster and C. Kesselman, *The Grid Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, San Francisco, California, 1998.
- [2] W. S. Ark and T. Selker, *A look at human interaction with pervasive computers*, IBM Systems Journal, Vol. 38, No. 4, 1999.
- [3] DARPA Agent Markup Language, <http://www.daml.org>
- [4] Semantic Web, <http://www.w3.org/2001/sw/>
- [5] M. Blay-Fornnarino, A. Pinna-Dery, and M. Riveill, "Towards Dynamic Configuration of Distributed Applications", *2002 IEEE International Conference on Distributed Computing Systems Workshop Proceedings*, June 1002
- [6] E. Sirin, J. Hendler, B. Parsia, "Semi-automatic Composition of Web Services using Semantic Descriptions." Accepted to "Web Services: Modeling, Architecture and Infrastructure" workshop in conjunction with ICEIS2003, 2002.
- [7] G. Jiang and G. Cybenko, "Query Routing Optimization in Sensor Communication Networks", *IEEE 41st Conference on Decision and Control*, Las Vegas, December, 2002.