Enhanced Service Discovery in Bluetooth

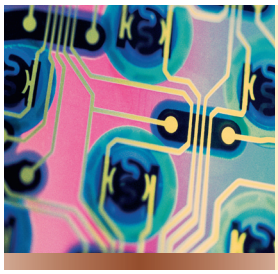
Sasikanth Avancha, Anupam Joshi, and Timothy Finin
University of Maryland, Baltimore County

The rapid evolution and expansion of wireless-enabled environments have increased the need for sophisticated service discovery protocols (SDPs). Typically, service discovery involves a client, service provider, and lookup or directory server. Service registration and lookup, or matching, are important components of most common SDPs including Jini, the service location protocol (SLP), Salutation, UPnP (universal plug and play), and UDDI (universal description, discovery, and integration).

To discover a service, a client uses one of these protocols to issue a query to a central server or an individual service provider. The service description in the query may contain a specific name and set of one or more attributes. The server or provider attempts to match the query's pattern with the pattern of a service its database contains, then it returns the appropriate response to the client.

These SDPs assume the existence of

- continuous and robust network connectivity,
- support from network protocols such as IP,
- support from transport protocols such as TCP (transmission-control protocol) and UDP (user datagram protocol), and
- network layer mechanisms such as multicasting.



Semantic matching improves the quality of service discovery by ensuring more positive responses to requests.

However, some or all of these requirements may not be met in wireless networks, especially those formed in an ad hoc manner. For example, temporary disconnections occur frequently in such networks, and the negative impact on TCP performance in turn decreases SDP performance.

Bluetooth (http://www.bluetooth.com) short-range wireless technology operates in the globally available 2.4-GHz ISM (industrial, scientific, and medical) frequency band and provides data rates of up to 432 Kbps (symmetric) and 721 Kbps (unsymmetric). The Bluetooth protocol stack currently includes specifications that define the SDP, RFCOMM (for cable replacement), the logical link control and adaptation protocol (L2CAP), a host controller interface (HCI), the link manager protocol (LMP), the baseband protocol, and a radio frequency (RF) protocol.

The Bluetooth specification defines a lower power level that covers the personal area within a room and a higher power level designed to cover a medium range, such as within a home. The specification supports both point-to-point and point-to-multipoint connections. Up to seven slave devices can be set to communicate with a master radio in one configuration. Ad hoc scatternets can link several such piconets together to allow communication among continually flexible configurations.

UNIVERSALLY UNIQUE IDENTIFIERS

The Bluetooth SDP is optimized for ad hoc networks and resource-constrained devices. In place of large strings that other SDPs use for pattern-matching service requests, the Bluetooth SDP uses 128-bit universally unique identifiers (UUIDs), one of which is associated with every service and attribute of that service.

The UUID-based approach is well suited to service discovery in ad hoc networks formed between, for example, a headset and cellular phone. The headset service is simple and straightforward enough, in terms of possible attributes, to be associated with a UUID. The phone can specify the headset service's UUID in its service discovery request and expect to receive either a positive or negative response—no inexact matching is possible or required.

However, UUID-based description and matching of services are often inadequate. For example, consider a wireless hotspot such as an airport terminal or shopping mall where patrons use handheld devices to discover information about available services from one another as well as a central business directory. A shopper equipped

with a PDA is searching for a luggage store that carries 18-, 22-, and 24-inch leather suitcases costing less than $500; she prefers the largest size but would take either of the other two if they are the only sizes available.

Using regular Bluetooth SDP, the PDA would specify a series of UUIDs associated with the luggage store service in its request—one each for the service and its attributes. If the store determines that it carries 18- and 22-inch suitcases costing less than $500, but not 24-inch ones, the request will fail. Also, because the current version of Bluetooth SDP does not support service registration, the luggage store would likely not be able to register its services with the mall directory, denying the user this useful facility.

Mall, airport terminal, and other hotspot services are associated with a larger number of more complex attributes than a simple headset service. Specifying requests in terms of UUIDs, even if possible, would lead to a situation in which a simple positive or negative response is meaningless.

SEMANTIC SERVICE DISCOVERY

To address this problem and increase the quality of service discovery, we have enhanced the Bluetooth SDP matching mechanism to use semantic information associated with services rather than simple UUIDs in hotspot environments. This includes priorities, expected values of service attributes, and some index of a match's closeness—for example, matching two out of three attributes is deemed a success.

To support this matching mechanism and allow more efficient service discovery, we have introduced a service ontology described in a semantic language and a Prolog-based reasoning engine that uses the ontology.

Traditionally, the assumption is that sophisticated matching mechanisms impose heavy computational and memory burdens that only server class systems can handle. However, our experiments reveal that devices such as Compaq's iPAQ Pocket PC, which supports at least 32 Mbytes of RAM, could handle even a heavyweight engine with no visible performance degradation.

SYSTEM DESIGN

The process of semantic matching is dependent on a well-defined ontology. Using the ontology effectively and reasoning about the information it provides require a powerful engine. We implemented the reasoning engine in XSB, a research-oriented logic programming system for Unix- and Windows-based systems.

Using an ontology to describe services can facilitate inexact matching by specifying rules and constraints on attributes.

Service ontology

Describing services ontologically is superior to UUID-based descriptions because it provides a structure for reasoning about and deriving knowledge from the given descriptions. An ontology also describes relationships between different entities in the system more clearly. Further, the ontology can facilitate inexact matching by specifying rules and constraints on attributes—for example, by imposing a priority on each option.

We use the semantically rich DAML+OIL—Darpa Agent Markup Language and Ontology Inference Layer (http://www.daml.org/2001/03/daml+oil-index.html)—to describe our ontology. The World Wide Web Consortium (http://www.w3c.org) and the US Defense Advanced Research Projects Agency (http://www.darpa.mil/) are developing this language as an extension to the Resource Description Framework and Extensible Markup Language (XML).

Using DAML+OIL as our description language offers two main advantages. First, it's easier to use the syntax and rules of an existing language than to create a new one. Second, because DAML+OIL is becoming a standard for use in the semantic Web, services developed for the Internet using this language can easily be installed on Bluetooth-enabled devices and disseminated in Bluetooth networks.

Reasoning engine

The key to correct functioning of our reasoning engine is a knowledge base with sufficient and complete information about service instances. Large amounts of factual data must be loaded into the knowledge base before the program can use them. XSB, a variant of Prolog that also interfaces easily with C, offers a powerful method for asserting clauses that can improve program speed and indexing for compiled code. In addition, XSB can run on a small-footprint device such as the iPAQ because its overall memory requirement on Linux is less than 2 Mbytes.

DAML allows description of services in a highly structured, object-oriented manner, but it cannot take the place of a reasoning engine that derives meaning from this structured data. The XSB engine splits available data into extensional knowledge (facts present in the description) and intentional knowledge (facts derived from other facts). It thus extracts the relationships that DAML describes and uses them to arrive at a solution to a given service discovery query.

The engine first performs pattern matching to determine whether it can answer the query directly. Upon failure, it evaluates other possible solutions, which match the requested query attribute values within an error range based on the specified closeness index.

Figure 1 illustrates our system's service registration and semantic matching mechanisms. We use the Bluetooth stack for Linux developed by Axis Communications to implement the enhancements. The regular SDP in this stack uses XML files to describe services and associate them with UUIDs. Thus, the matching process involves parsing

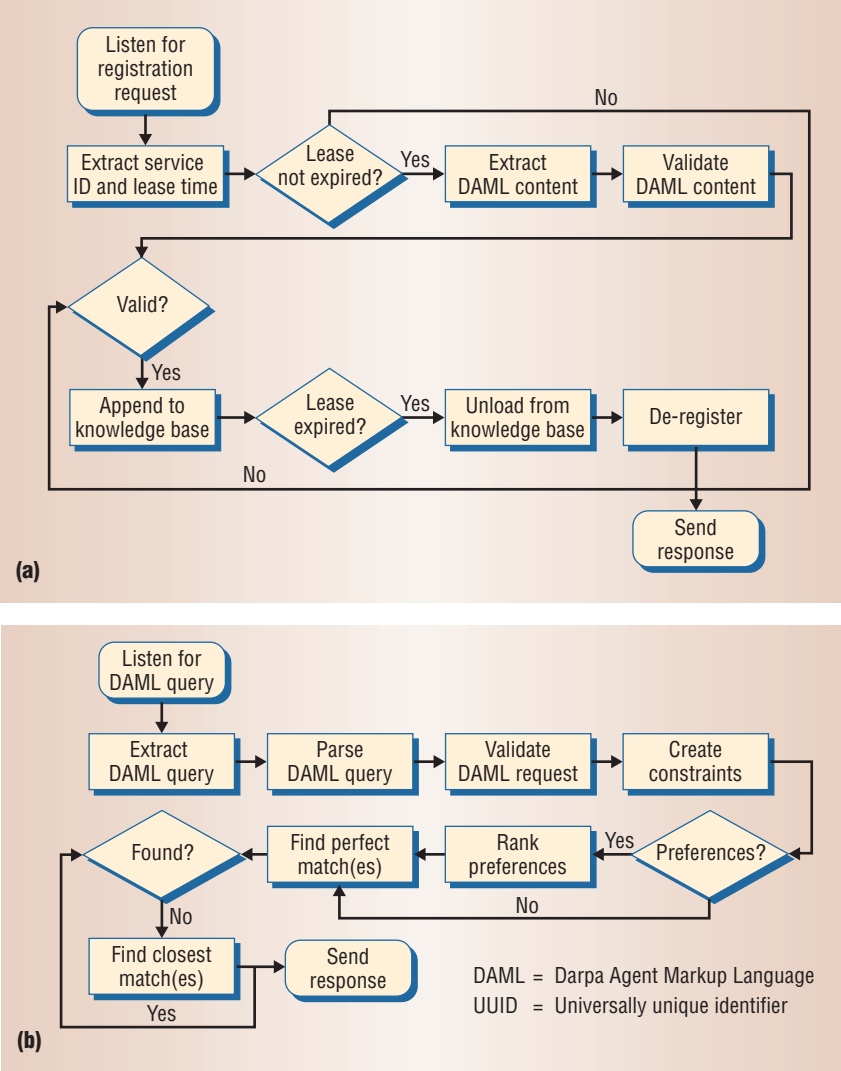


Figure 1. Service registration and semantic matching mechanisms. (a) The service registration process involves parsing a request and validating its content and lease time, resulting in either addition or deletion of the request to or from the knowledge base. (b) The semantic matching process parses and validates the DAML request, creates a ranked list of attributes, and determines the closest value that matches each attribute's specified value.

and validating the request before comparing requested and available UUIDs.

PERFORMANCE EVALUATION

To evaluate service registration and semantic matching performance, we used two Linux-based laptops to compare the response and processing times for service discovery queries in the enhanced Bluetooth SDP system with those in the regular system. We created a discovery request in DAML+OIL for a printer service associated with eight attributes.

Response time

In the first experiment, a client sent 100 service discovery requests associated with a single attribute to the server, and we measured and calculated the average response time. The client then associated two attributes with the printer service in a second set of 100 messages. We repeated this procedure until the request contained all eight attributes.

The results showed that

- the enhanced SDP system's response time is three to 14 times greater than that of the regular system for requests containing one to eight attributes, respectively. This is primarily due to the transmission time because request messages in the enhanced SDP system are 40 to 50 times larger than corresponding requests in the regular system.
- the increased response time is not unacceptable from the user's perspective. In absolute terms, regular SDP response time ranges from approximately 0.05 to 0.1 seconds, whereas enhanced SDP response time ranges from 0.2 to 1.2 seconds. Considering that the largest request size in the enhanced SDP was 1,724 bytes, a 1.2-second response time seems reasonable.

In addition, we believe that the quality of results in the enhanced system ensures that clients will have to repeat few, if any, requests due to negative responses from the server.

Processing time

The second experiment followed the same procedure except that we captured only the time the server took to process a service discovery request. This is important because in regular Bluetooth SDP the parsing and matching mechanisms are fairly simple, whereas both mechanisms are complex in the enhanced system.

The results demonstrated that

- the processing time for simple and enhanced SDP is comparable for a given set of attributes in a request, and
- the processing time for enhanced SDP is nearly constant, irrespective of the number of attributes in the request.

The penalty paid for using a complex matching technique with a heavy reasoning engine such as XSB appears to be insignificant compared to the relatively lightweight mechanism of parsing an XML file and comparing UUIDs.

Because we used DAML and Prolog, which are independent of the underlying wireless technology, to design and implement these enhancements, they can easily be packaged into a separate module and plugged into any other protocol stack at the appropriate layer. We envision our semantic service discovery solution being introduced into wireless LAN, IEEE 802.15.1, and wide-area wireless networks such as Cellular Digital Packet Data.

Given the rapid deployment of wireless technologies in hotspots such as cafes, shopping malls, and restaurants throughout the world, we believe that semantic service discovery will play an important role in future mobile-commerce applications.

In peer-to-peer (P2P) environments involving the sale and exchange of small items, devices need to discover services or information that others are willing to provide or sell. This discovery cannot be based solely on fixed values such as UUIDs or service names because the peers have formed the network in an ad hoc manner.

Semantic service discovery is ideal in such a situation—a peer can simply issue a semantic query and expect to receive a positive response. We are currently designing and developing a P2P m-commerce application using semantic service discovery in which users can buy and sell information such as news clips, weather reports, and stock quotes.

Other future work includes addressing energy consumption issues by reducing the amount of data transmitted and received in service discovery and registration requests, while maintaining all semantic information completely. We also plan to investigate quantifying improvement in the quality of service discovery. ■

Sasikanth Avancha *is a graduate student in the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County. Contact him at savanc1@csee.umbc.edu.*

Anupam Joshi *is an associate professor in the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County. Contact him at joshi@csee.umbc.edu.*

Timothy Finin *is a professor in the Department of Computer Science and Electrical Engineering at the University of Maryland, Baltimore County. Contact him at finin@csee.umbc.edu.*