Comparison of Service Discovery Protocols

M. Bratoev, B. Bontchev

Key Words: *Service discovery protocols; SOA; UPnP; Jini; SLP.*

Abstract. *The advent of wireless, mobile and ubiquitous computing has made it necessary to develop a highly-dynamic infrastructure that enables devices to advertise their services and clients to easily locate and utilize a particular service out of hundreds of accessible services. There must be no need for complex configuration, administration or device driver installation. Service discovery is an important element in these intelligent computer networks and contributes for automated discovery, seamless information exchange and remote control between devices. This paper is represents a survey of several prominent service discovery technologies, compares their major features and outlines the challenges and trends in development of service discovery protocols during next years.*

1. Introduction

During the last years a large part of science research interests and efforts have been directed toward the invention of technologies that will enable construction of new-generation, intelligent, self-configuring computer networks. The goal is to build *zero-configuration* network architectures with automatic discovery of services, where a device can dynamically join the network, use available services, negotiate required parameters and, in the end, smoothly leave the network. Traditional software communication platforms (RPC, Java RMI, DCOM, CORBA, and MOM) are considered unsuitable for the above scenario because they are designed and implemented to work in networks with relatively static structure. Their usage in dynamically changing networks is irrational since manual change of their working parameters is required to reflect joining or leaving of a device (service). Under these circumstances, the idea for creation of new technology instruments was born. That technology should help automatic advertisement, registration discovery and utilization of services and devices in computer networks.

The software of modern, dynamic, adaptive, distributed systems is implemented according to principles of loosely-coupled, service oriented architectures (SOA) [12,15]. SOA defines conceptual software platform where information exchange between system modules and communication with external systems is based on interaction of services. In the context of SOA, the service is defined as an object that can provide an information, perform an action, or control a resource on behalf of other objects. The service has five basic characteristics:

- A service is a strategic encapsulation of program logic and interface (API) to that logic. This interface defines the messages to interact with the service.
- Access to service functionality is possible only through its API.
- Each service has an address. The concrete format of this address is dependent on particular communication protocol being used
- The service accepts messages, performs relevant actions and returns a result to the client.
- The service itself is responsible to establish and enforce its security policy and should itself perform client authentication and authorization by itself.

SOA defines a set of rules, mechanisms and protocols to describe, advertise and discover services as well as to communicate between services and their clients (*figure* 1). A Service Discovery Protocol (SDP) is a specification of procedural interactions between a device and other devices in the network and defines the syntax and semantics of messages exchanged in order to advertise and discover services [1,2]. In addition, SDP prescribes expected behavior of communicating entities as a result of arrival of specific message, i.e. it defines the state chart diagram for each entity involved in the service discovery process.

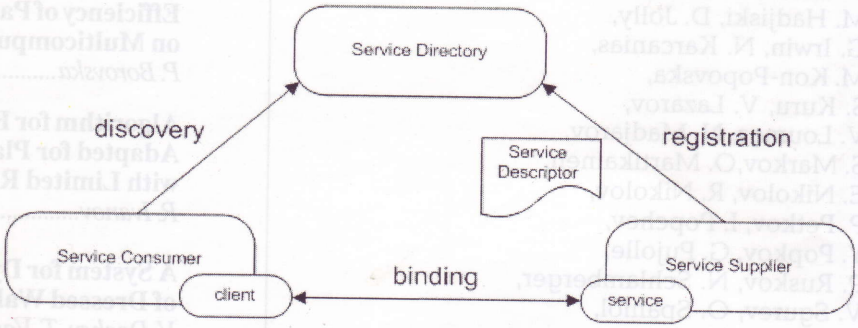


Figure 1. SOA based architecture

This paper provides an overview and comparison of several prominent service discovery mechanisms – SLP [3,4,19], UPnP [7,8] and Jini [9], which are currently used for building self-configured networks. Components that constitute each of these technologies are described and some of important interactions of these components are examined. A special attention is given on SDPs used since they are fundamental instrument to minimize the user involvement in system configuration and, more importantly, they have different implementation which serves as foundation for their comparison. In order to make the presentation of service discovery mechanisms more systematical and easy to understand we propose a unified structure for their description that consist of the following elements:

- *Components* – introduces the different functional entities that constitute the inspected service discovery system and identifies their roles and responsibilities.

• *Architecture* – outlines the interaction of the individual components and explains how the execution focus moves from one component to another during operation of the given protocol.

• *Mechanism for service discovery* – explains the process and of the service discovery - how the service is described, how it is advertised and what the service clients have to do in order to discover a particular service of interest.

This standard approach for description of service discovery techniques makes the intended comparison easier, facilitating the identification of their similarities and differences, and contributes to more profound understanding of fundamental concepts and special features behind them.

2. Service Discovery Protocols

2.1. Service Location Protocol

Service Location Protocol (*SLP*) is an *IETF* (*Internet Engineering Task Force*) standard that provides for network applications means for service discovery in distributed environment. SLP is an instrument for service discovery and advertisement in IP networks and makes heavy use of TCP and UDP protocols. SLP allows applications to discover existence, location, and characteristics of desired services and enables services to advertise their capabilities.

2.1.1. Components

In SLP there are three basic software objects (*agents*) [4]:

• *User Agent* (*UA*) – performs service discovery on behalf of client software.

• *Service Agent* (*SA*) – advertises the location and attributes on behalf of services.

• *Directory Agent* (*DA*) – aggregates information for available services in the network.

2.1.2. Architecture

SLP defines two modes of system operation - with DA and without DA [4]. Depending on presence or absence of DA in SLP, two separate architectures are possible whose models are shown on *figure 2*.

When DA appears, it collects all service information for services in the network. All SA components must advertise in DA before possible access to these services. When UA searches particular service it sends to DA request for information about this service. Later the received information is used for access to this service. In larger networks environments, several DA may be used to increase overall system performance, scalability and fault tolerance. Because SAs register in all DAs they detect, all DA will contain the same service information (the presumption is that all SAs can find them all). Since UAs can choose any available DA to issue request to the load will be shared among DAs. In the case, when no DA responds to UA request, UA just multicasts an request message in the network hoping that any SA with coinciding group will respond.

When there are no DAs, UA components periodically multicast the same requests to the available SAs that it would unicast to the DA if such were present. This service request includes a query that the SAs process against the attributes of services it advertises. If the multicast request fails in matching, the SA simply discards it.

The presence of DA reduces the traffic in the network, since the multicast messages are avoided. Disadvantage of an architecture with DA is the more complicated infrastructure, increased cost and degraded system reliability due to the dependence on introduced DAs.

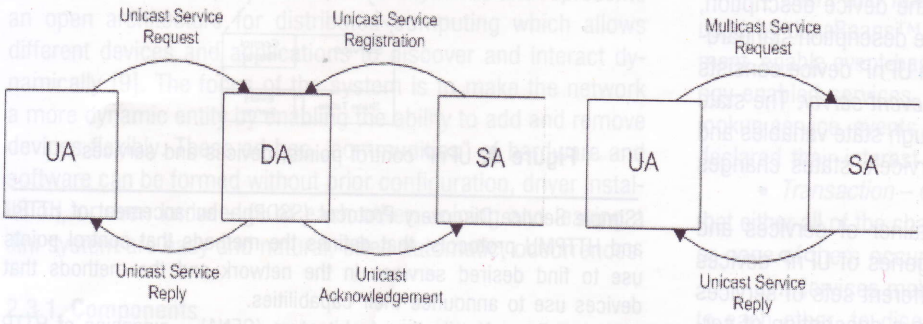


Figure 2. SLP architectures

2.1.3. Mechanism for Service Discovery

Services in SLP are described with unique identifiers (*ServiceURL*), which contains the type of service and the address where more information about the service could be found [6]. Client applications that obtain this URL have all the information they need to connect to the advertised service. The actual protocol the client uses to communicate with the service is independent of SLP.

At startup, UAs and SAs first determine whether there are any DAs on the network. DA addresses can be configured statically or dynamically obtained form DHCP server [5]. In these cases, there is no need to perform DA discovery. In all other cases, UAs and SAs first need to find available DAs. The goal of discovery process is to acquire *ServiceURL*, scope and attributes of each DA in the network. There are two ways to discover DAs – active and passive [3].

• In the case of active discovery, UA and SA send *Service Request* (*SrvRqst*) messages on group SLP address 293.255.255.253 specifying the desired scope of DA. When DA receives these messages it checks if the specified scope is the same as its own. If a match occurs, DA returns *Service Reply* (*SrvRply*) to the sender.

• In passive DA discovery, DA periodically pumps multicast messages into the network presenting itself to available UAs and SAs. All UAs and SAs that receive these messages can extract and memorize DA's address and scopes.

After the DA has been discovered, SA can register its services by issuing Service Registration messages to appropriate DA. DA then returns acknowledgement for successful or unsuccessful registration. UA searching for particular service sends a

SrvRqst message to DA and will receive back a *SrvRply* message. In *figure* 3 it is shown how the client application uses SLP API to send service request (*SrvRqst* message) and the SA replies with a *SrvRply* message.

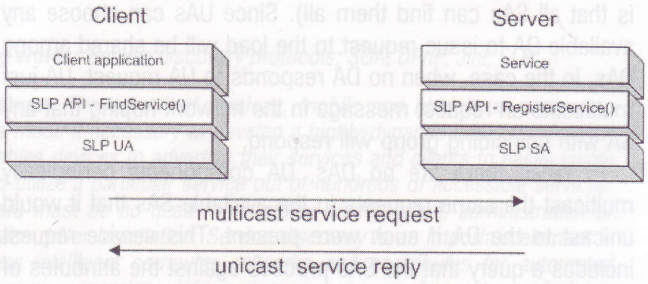


Figure 3. Service discovery with SLP API without DA

It should be emphasized that SLP provides mechanism only to discover information for services but not for their access and control. To connect to a given service the developers have to use or implement additional means for that. That's why SLP is not a complete solution for service discovery.

2.2. Universal Plug and Play

Universal Plug and Play (*UPnP*) is a technology developed by Microsoft and a consortium of other organizations which includes IBM, 3Com, Alcatel Telecom, Compaq and Dell. According to the UPnP specification, it is architecture for pervasive peer-to-peer network connectivity of intelligent appliances, wireless devices, and PCs of all form factors. The basic goal of UPnP is to provide distributed, open networking architecture that relies on IP, TCP, UDP, HTTP and XML to enable seamless proximity networking where a device can dynamically join a network, obtain an IP address, convey its capabilities, and learn about the presence and capabilities of other devices. After that the device can directly interact with other devices in the network (for example, perform a remote control or data exchange). Finally, a device can leave a network smoothly and automatically without leaving any unwanted state behind.

2.2.1. Components

The main components of UPnP are *a service, a device* and *a controller* (*control point*). The smallest unit of control in the UPnP network is a service. A service exposes actions and models its state with state variables. Similar to the device description, this information is a part of an XML service description standardized by the UPnP forum [13]. A service in UPnP device consists of a *state table*, a *control server* and an *event server*. The state table models the state of the service through state variables and updates them dynamically when the service's status changes during working process.

UPnP device (*figure* 4) is a container of services and nested devices. There are different categories of UPnP devices and each category is associated with different sets of services and embedded devices. Each device has a description of services it provides, embedded devices and a list of additional properties (name, for example). All this information is captured in an XML document that the device delivers to control points on demand.

A control point in UPnP is a component capable of discovering and controlling other devices through invoking services exposed by them. After a device discovery, the control point could retrieve device description and get a list of associated services, retrieve services descriptions, invoke operations to control a service and subscribe to events generated by service when its state changes.

2.2.2. Architecture

For exchange of data and control messages, the UPnP uses a protocol stack, which consists of three upper layers responsible for service and device description in UPnP network [8]. *The UPnP Device Architecture* defines a template (XML schema) for construction and description of each device or service. The required information for a given advertising or discovering message is extracted from description of service or device and is included in the message before being formatted by SSDP[1], GENA[2] and SOAP protocols. The lower layers in the UPnP protocol profile are composed of widely distributed, standard protocols with proved efficiency (SOAP, HTTP, UDP, TCP, and IP). This is a big advantage of the technology, since it helps for seamless integration of the technology in legacy computer networks that internally use these protocols. Regardless of the convenience that is the usage of the HTTP based protocols SSDP and GENA, this requires a control point and services to implement a Web server or to rely on services of an existing one.

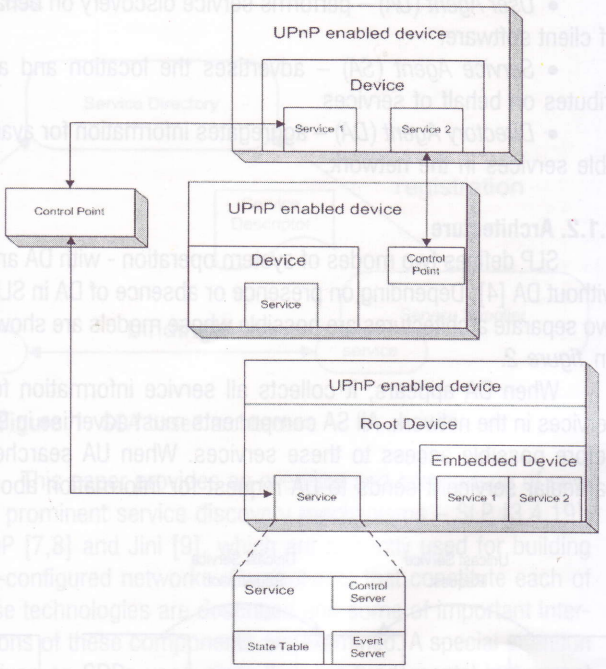


Figure 4. UPnP control points, devices and services

[1]Simple Service Discovery Protocol (SSDP) – enhancement of HTTPU and HTTPMU protocols, that defines the methods that control points use to find desired services in the network and the methods that devices use to announce their capabilities.

[2] General Event Notification Architecture (GENA) – extension of HTTP with additional methods and headers, that contributes for realization of mechanism for event notification of control points when the state of a UPnP service changes.

2.2.3. Mechanism for Service Discovery

The UPnP working process consists of the following six steps:

1. Addressing – the device is configured with static IP address or receives a dynamic address by the DHCP server.

2. Discovery – based on packet multicasting. When a device is added to the network, the UPnP discovery protocol allows this device to advertise by IP its services to control points on the network using *advertisement messages*. Similarly, when a control point is added to the network, the UPnP discovery protocol allows that control point to search for devices of interest on the network using *discovery messages*. When a device is removed from the network, it should notify other members of the network for that, multicasting a number of messages revoking its earlier announcements.

3. Description – to learn more about the device and its capabilities or to interact with the device, the control point must retrieve (by doing HTTP GET request) a description (XML document) of the device and its capabilities from the URL provided by the device in the discovery message.

4. Control – in contrast to SLP, UPnP provides a means for remote service control. The control points construct messages containing description of the invoked operations and values of input parameters, send them to the remote service and when the action is completed (or failed), the service returns the results or any errors encountered during operation.

5. Eventing – on every service state change, the service (in the part of *publisher*) checks for control points subscribed to receive notification for this particular event occurrence and sends the corresponding messages as XML document containing the names and the new values of service state variables.

6. Presentation – offers a graphical user interface (Web page) to control points for control and state monitoring of services and devices.

UPnP is more resistant in respect to individual device failure than SLP. On the other hand, the resources required by UPnP devices and control points are significantly more compared to SLP, which comes from the fact that UPnP uses XML and SOAP. Another disadvantage of this technology is that it allows discovery only by service or device type. It does not permit discovery by attributes as in SLP [6].

2.3. Jini

Jini is a development of *Sun Misrosystems* and represents an open architecture for distributed computing which allows different devices and applications to discover and interact dynamically [9]. The focus of the system is to make the network a more dynamic entity by enabling the ability to add and remove devices flexibly. These ad hoc „communities" of hardware and software can be formed without prior configuration, driver installation, or even knowledge of each other. Joining and leaving a Jini system are easy and natural, often automatic, occurrences.

2.3.1. Components

In Jini all components are called services. A service can be implemented as either hardware device, software program, or a combination of the two. A network of Jini services is called *Jini federation*. The most important part of Jini architecture is the *lookup service*. Every service should be registered with at least one lookup service in order to be accessible by other services. The lookup service acts as a central repository for all services in the network and is analogous to Directory Agent (DA) in SLP. One lookup service can contain other lookup services which makes possible to build a hierarchy of lookup services.

2.3.2. Architecture

Jini architecture can be segmented into three categories [9]:

- *Infrastructure* – the set of components that enables building a federated Jini system.
- *Programming model* – The programming model is a set of interfaces that enables the construction of services (including those that are part of the infrastructure and those that join into the federation).
- *Services* – can be or are part of Jini federation and offers some functionality to every other member of the federation.

A Jini system can be seen as a network extension of the infrastructure, programming model, and services that constitute traditional Java technology which allows the application of its main concepts in computer network.

The Jini infrastructure consists of:

- A *distributed security* integrated into RMI (*Remote Method Invocation – RMI*), that shifts the Java platform's security model to the field of distributed systems;
- *Discovery/join* protocol – defines the rules governing the processes for discovering existing services and for attaching and detaching services from the Jini system. Services in a Jini federation communicate with each other by using a *service protocol*, which is a set of interfaces written in the Java programming language;
- *Lookup service* – the entries recorded in it are serialized Java objects, which can be downloaded by potential clients and work as local proxies between the client and the service registered the object.

The Jini programming model is composed of interfaces that support the interaction between services and Jini infrastructure. Some of those interfaces are:

- *Leasing* – defines a way of allocating and freeing resources using a renewable, duration-based approach for obtaining object references.
- *Event and notification* – extension of the event model used by JavaBeans™ components to the distributed environment, enable event-based communication between Jini technology-enabled services. When the services register or leave the lookup service, events are generated and objects that have been declared their interest for these events are notified.
- *Transaction* – enable entities to cooperate in such a way that either all of the changes made to the group occur atomically or none of them occur.

Jini services make use of the infrastructure to make calls to each other, to discover each other, and to announce their presence to other services and users. Services appear programmatically as objects written in Java programming language, perhaps made up of other objects. A service has an interface

that defines the operations that can be requested of that service.

2.3.3. Mechanism for Service Ddiscovery

The Jini operation can be summarized in the following three steps, represented in *figure 5*, which illustrates registration, discovery and invocation of the service. The first step in the procedure is the discovery of the lookup service. This step is similar to the discovery of DA in SLP and can be done in one of three ways:

- Sending a message to the previously configured static address. In response to this message the lookup service will acknowledge its existence in the network.
- Multicasting of UDP datagrams, that will force the lookup services to reply.
- Lookup service actively announces its presence as well as their groups by periodic message distribution to all devices in the network.

For discovery of lookup service, the *discovery* protocol from the pair *discovery/join* is used. After the lookup service is discovered, any Jini service which aims to register in the network sends *Service Registration* message. This message includes the proxy object of the service and a request for some leasing period (indicating how long the service will be registered). The response of the request includes the real lease time and the unique identifier of the service (*ServiceID*) that is used for discovery purposes and service identification in the context of the lookup service that returned it. The procedural interactions for service registration are described in *join* protocol.

The next step describes a Jini service that wants to find another service. After this service discovers the lookup service it sends a request to concrete *ServiceID* or *ServiceType*. When the lookup service finds a service with corresponding parameters it returns the proxy object of this service. The proxy object can contain the entire service implementation or just provide stubs that redirect (using RMI or other mechanism) the client request to a remote service implementation.

The ability of the service to dynamically download and execute Java code is essential for most Jini functions. It should be noted, however, that Jini architecture is dependent on Java application environment not on the Java language itself. Jini supports any programming language that have Java byte code compatible compiler.

3. Protocol Comparison

Service discovery protocols are proposed to facilitate dynamic cooperation among devices (services) with minimal administration and human intervention in normal system operation. This survey of SLP, UPnP, and Jini shows that these technologies address similar issues of service discovery and have identical architectural and functional characteristics. Each one of the inspected protocols provides means or contributes to service announcement, discovery and control. At the same time, however, there are some differences due to the fact that these protocols stress on different aspects of their functionality and put different weight to its components. This section proposes a

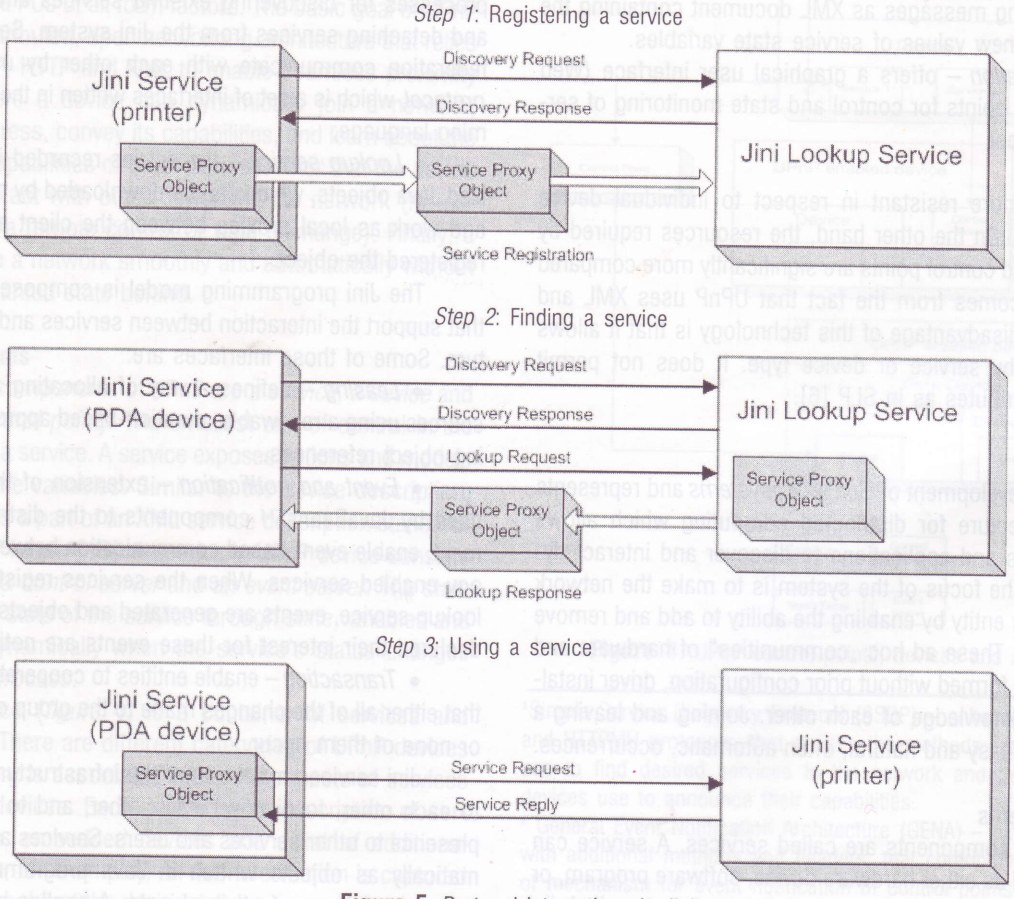


Figure 5. Protocol interactions in Jini

comparison of the presented SDPs that makes possible the identification of unresolved problems and helps outlining fields where more research needs to be done.

3.1. Comparison Criteria Selection

The selection of criteria for comparison of SLP, UPnP and Jini is essential since it is directly related to the consequent results and their usefulness. These criteria should be comprehensive enough to embrace the most important characteristics of considered SDP solutions. Following this principle of criteria selection we define four mutually orthogonal perspectives to these technologies:

- *Architectural* – represents different components, depict their roles in the system and describes the internal organization of the given SDP technology.
- *Service Description* – examines the techniques and methods for construction of service descriptors in the context of a specific SDP.
- *Operational* – provides a view to various dynamical characteristics of the SDP, such as leasing, event notification, active or passive service discovery and ability for service control;
- *Interoperability* – defines the dependability of the SDP system on specific operating system, hardware platform, network protocol or programming language.

Each of these perspectives reveals a different aspect of the nature of these protocols and gives a separate view to a given SDP solution. This view represents a projection of the SDP technology in the space of all characteristics that are used for description from a concrete viewpoint.

3.2. Comparison Results

The following table systematizes major distinguishing features of the presented protocols grouping them with respect to the proposed perspectives as identified in section 3.1.

From the information in this table we can make the following inferences:

- SLP is standardized and well documented through IETF. Since it is able to operate with or without DA, it is suitable for networks of different sizes, ranging from very small ad hoc connectivity to large enterprise networks. Unfortunately, SLP do not define a protocol for communication between clients and discovered services and lacks the event notification mechanism which is available in other two protocols.
- Jini technology is independent of any hardware platform and communication protocol, but it presumes that there is Java virtual machine (*JVM – Java Virtual Machine*) installed on every network device or that every device can use a surrogate [10] to represent it in Jini network. In addition, according to standard, *CLDC* (*Connected Limited Device Configuration*) configuration of *J2ME* platform does not support RMI. This forces developers themselves to implement similar mechanism when the application have to work on devices as cellular phones, PDA or embedded devices.
- UPnP relies on the wide-spread IP and Web technologies. It's usage of XML for service description is unique among examined protocols. This approach ensures a powerful instrument for device description of device attributes, service control commands and events that can occur during system functioning. Since UPnP does not use a registry, it is likely to generate significantly more network traffic than centralized variants of SLP and Jini.

Each presented service discovery technology has advantages and disadvantages. None of these technologies is a superset of the others and none is mature enough to dominate the market. A great problem with the currently available SDP solutions is that they are virtually incompatible. This means that services available in one platform can not be easily discovered and accessed from services based on another platform. The tight bindings of most SDP mechanisms to a particular communication protocols and hardware platform make it difficult to rapidly prototype ubiquitous computing applications spanning a wide variety of devices and services. In addition, from this SDP platform's comparison it is evident that each of them has its own method to specify service characteristics. The ability to make transition between two different service representations is critical for solving integration problems.

Currently, interoperability efforts are perhaps the most important force in service discovery, since it is very unlikely that device manufacturers will embrace multiple service discovery technologies on low-cost mobile devices.

4. Conclusion

Service discovery represents one of foundations that intelligent computer networks are built. On it suppresses the need for preliminary device configuration and helps for seamless information exchange between devices in order to enable remote control of one device from another.

The presented service discovery protocols have a number of functional similarities mainly regarding their orientation to local network services and methods used to describe and discover services. However, as a whole, they remain incompatible because of their dependence on operating platform, communication protocol or instrumental programming language in most of them. There are also many essential differences, such as the absence of event notification and mechanisms for service access in some protocols. Moreover, it is fairly apparent the lack of a common open standard that makes use of the strongest features of examined protocols and that can be used for construction of architectures not only with local but with both local and global access to resources.

The shown incompatibility defines the challenges and trends in development of service discovery protocols during next years. They could be generalized in the following way:

- further development and improvement of existing SDPs with a view to creation of complete solution for discovery and remote control of services and convergence of their characteristics;
- modeling and realization of communication bridges (such as *uMiddle* [15] and event based parsing solution described in [18]) between several SDP systems in order to allow construction of multi-platform SDP-enabled applications;
- proposing new SDPs in various application areas – for example, in [11] a Service Accumulator Agent protocol for Grid architecture is proposed;
- building of an open standard for discovery, local and

Table 1. Comparison of service discovery protocols

	Feature	SLP	UPnP	Jini
Architecture	Components	Service Agent (SA) Directory Agent (DA) User Agent (UA)	Devices Control points Services	Services
	Central repository	Directory Agent (DA)	No	Lookup service
	Network model	client-server (with DA)/peer-to-peer	Peer-to-peer	Client-server (when using lookup service)/peer-to-peer (when peer lookup is used)
	Service advertisement	Registration in DA	Multicast of NOTIFY message with NTS: *ssdp:alive*	Discovery/Join protocol
Service Description	Standard for service types	IANA – Service Template	UPnP Forum Working Commitee – device/service template	No
	Service discovery	By service type (plus attributes if LDAP filters are used), through request to DA or multicast to all SAs	By type or identifier of device (service), after the control point reads service advertisements	Request to lookup service contains the service template interface and attributes)
	Service description	Service type, attributes (in respect to corresponding template)	XML document, conforming to XML schema for the concrete device/service type	Interface type and attributes
Operation	Service control	No (the result of discovery process is the service URL)	SOAP requests to service URL	Service proxy object that implements the service or redirects (using RMI or other mechanism) requests to the remote service
	Leasing concept	The specified expiration time during service registration	max-age = *expiration_time* should be set in the header block of the NOTIFY message	Jini leasing
	Event notification	No	The service publishes events (GENA), when the value of a status variable changes	Remote Events
	Service discovery model (active/passive)	Active and passive (only active without DA).	Active and passive (using discovery and advertisement messages)	Active and passive (only active when peer lookup technique is used)
Interoperability	Hardware Platform/ Operating system	Dependent	Dependent	Independent
	Communication protocol	Dependent (works only in IP-based networks and uses TCP and UDP transport protocols)	Dependent (tightly bound to Web protocols - IP, TCP, UDP and HTTP)	Independent (uses Java RMI)
	Programming language	Independent	Independent	Java (or language with compiler capable to produce Java compatible byte code)

global access to loosely-coupled services in the light of service oriented architectures.

Since service discovery mechanisms currently in development are still in their infant stage, there are many areas of improvements still needing research. Another promising research challenge is the work on quality analysis of existing SDPs and architectural modeling of their operation using the appropriate modeling instruments (for example, *UML* or some kind of architecture description language (ADL) like *Rapide* [16,20]) for description of events and dynamic behavior of *„plug-and-play"* device communities. The construction of such architectural models is intended to closely examine dynamic characteristics of SDPs, like consistency, accessibility, latency, scalability, delays, dead locks and other synchronization problems. The results obtained form this analysis would be of great value for further comparison of various SDPs and can be used to predict the behavior of distributed systems in dynamic conditions and to fuel the development of new, reliable mechanisms to guarantee quality of service (QoS) in spontaneous SDP-enabled networks.

We have shown that one of the reasons for incompatibility of different SDP platforms is the absence of a common standard method for service description in these platforms. To help solving this integration problem, some means to translate between different service descriptions are needed. Our future work is concerned with proposing a common description language that provides uniform set of abstractions for representing the services in platform-neutral format. Finally, we plan to build a tool that translates from this abstract format to a native service description and vise versa.

References

1. Richard, G. G. Service Advertisement and Discovery: Enabling Universal Device Cooperation.– *IEEE Internet Computing*, Sept./Oct. 2000, 18-26.

2. Helal, S. and C. Lee. Protocols for Service Discovery in Dynamic and Mobile Networks.– *International Journal of Computer Research*, 22, 1-12.

3. Guttman, E., C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. IETF RFC 2608, June 1999.

4. Guttman, E. Service Location Protocol: Automatic Discovery of IP Network Services.– *IEEE Internet Computing*, 3 1999, No. 4, 71-80.

5. Perkins, C. and E. Guttman. DHCP Options for Service Location Protocol. IETF RFC 2610, June 1999.

6. Guttman, E., C. Perkins, and J. Kempf. Service Templates and Service:Scheme, IETF RFC 2609 June 1999.

7. UPnP™ Device Architecture. Version 1.0, 2000 http://upnp.org/resources/documents.asp.

8. Microsoft, Corp. Universal Plug and Play Device Architecture Reference Specification.

9. Sun Microsystems Inc. Jini Specifications Version 2.0. http://wwws.sun.com/software/jini/specs/.

10. Sun Microsystems Inc. JiniTM Technology Surrogate Architecture Specification, v 1.0 DraftStandard http://surrogate.jini.org.

11. Heaton, J. B. Enabling the Grid for Pervasive Computing. MSc Theses, Dep. of Distributed Systems Engineering, Lancaster University, 2004.

12. Microsoft, Corp. Service Oriented Architecture. http://msdn.microsoft.com/architecture/soa/.

13. UPnP Forum. http://www.upnp.org/.

14. Oasis Consortium. Reference Model for Service Oriented Architecture. Committee Draft 1.0, February 2006.

15. Nakazawa, J., J. Yura, and H. Tokuda. uMiddle: A Universal Framework for Bridging Diverse Middleware Platforms.

16. Dabrowski, C. and K. Mills. Analyzing Properties and Behavior of Service Discovery Protocols Using an Architecture-Based Approach. In Proceedings of Working Conference on Complex and Dynamic Systems Architecture, Brisbane, Australia, December 2001.

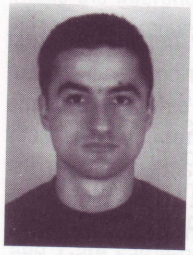
17. Capra, L., W. Emmerich, and C. Mascolo. Middleware for Mobile Computing. In proceedings of the International Conference on Networking 2002, Pisa, Italy, May 2001.

18. Yйrom-David Bromberg and Valйrie Issarny. Service Discovery Protocol Interoperability in the Mobile Environment. In Proceedings of the International Workshop Software Engineering and Middleware (SEM), September 2004, 64-77.

19. Bettstetter, C. and C. Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. In proceedings of the 6th EUNICE Open European Summer School: Innovative Internet Applications, 2000.

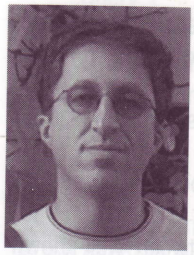
20. Luckham, D. Rapide: A Language and Toolset for Simulation of Distributed Systems by Partial Ordering of Events. DIMACS Partial Order Methods Workshop IV, July 1996, August 1996.

Manuscript received on 30.03.2006



Mihail Bratoev *received a MS Degree in Computer Science at the Technical University of Sofia, Bulgaria, in 2006. He is now finishing his PhD Thesis in the field of service discovery in distributed software environments. Dr. Bratoev is Microsoft Certified Solution Developer (MCSD) and has participated as senior developer in many industrial projects. He is an author and coauthor of several publications in international conferences and magazines. His present scientific interests are in the field of pervasive computing, zero-configuration computer networks, and semantic service discovery systems.*

Contacts:
Sofia University St. Kliment Ohridski
Faculty of Mathematics and Informatics
Department of Information Technologies
e-mail: mihail.bratoev@rila.com



Boyan Bontchev *received a MS Degree in Electronics and Automation Engineering at the Technical University of Sofia, Bulgaria, in 1988. He got his PhD in Computer Science at the Center of Informatics and Computer Technology (Bulgarian Academy of Sciences) in 1993, investigating hybrid data flow architectures. Dr. Bontchev has participated and conducted many research and industrial projects in the area of parallel processing, object-oriented software, mobile systems and computer modeling and simulation, for several entities in Portugal, Spain, Italy, Bulgaria and EC FP5/6. He is author and coauthor of more than forty research papers published at international conferences and in magazines. Since 2004 Dr. Bontchev is Associated Professor at Chair of Information Technologies, Sofia University. His current scientific interests are in the field of service sciences, software modeling and design, and adaptive systems.*

Contacts:
Sofia University St. Kliment Ohridski
Faculty of Mathematics and Informatics
Department of Information Technologies
e-mail: bbontchev@fmi.uni-sofia.bg