

Dynamic Device Access for Mobile Users

Dirk Kutscher and Jörg Ott

Technologiezentrum Informatik (TZI), Universität Bremen,
Postfach 330440, 28334 Bremen, Germany,
{dku|jo}@tzi.uni-bremen.de

Abstract. The Dynamic Device Association (DDA) framework presented in this paper provides service discovery and secure device access for mobile users. We discuss scenarios and requirements for enabling mobile users to find and access co-located (mobile) multimedia services on demand. As example, we use the dynamic discovery of IP telephony services in a network that an application on a mobile device can locate, personalize and control on behalf of a user. We examine existing protocols such as SLP and UPnP and derive a different, more general solution that addresses the issues of scalability for service announcements and security for device access. We outline the DDA approach with its five phases: service discovery, selection, device association, application protocol operation, and device dissociation. We show how the DDA approach can be used to bootstrap arbitrary application protocol sessions and present a concrete protocol specification and its implementation.¹

1 Introduction

Mobile users increasingly rely on the ubiquitous availability of applications ranging from paging and voice communications to Internet access and web browsing to full-fledged multimedia collaboration and entertainment. However, the typical mobile device is not able to implement all the applications of potential interest to a user, at least not equally well. This mostly stems from its form factors and the respective incompatible requirements for different types of applications (e.g. text processing with a large screen and keyboard vs. voice communications with minimal weight and size). Hence, typical users carry numerous devices, that overlap in some parts of their functionality and are complementary in others: notebook computers, tablet PCs, PDAs, and cellular phones comprise just a subset of the variety of devices available today.

To overcome the inherent limitations of individual mobile devices, they may be combined with one another to create a more powerful working environment. And they may be complemented by functionally richer,

¹ A long version of this paper is available from <http://www.dmn.tzi.org/dda/>.

more powerful, or just specialized ones: “personal” appliances such as desktop computers and telephones in the office space or at home as well as “public” devices such as PCs in Internet cafés, speaker phones in conference rooms, or video presentation and communication equipment in dedicated meeting rooms. One notable example is the use of IP telephones in conjunction with PDAs, where the latter support configuring and controlling the former: by means of some infrared interface [ABL02] or through third party call control.² Users may log in to any such IP phone device within their domain and take control of it; i.e. they become able to place calls with their privileges under their identity and may receive calls placed to their phone number or URI and have all their personal settings loaded into the phone upon login.

In this IP telephony scenario, an employee is able to move around in the office and take control of an arbitrary phone by means of a portable device (provided she possesses the necessary permissions). This concept can be generalized in several ways: 1) IP telephony can be extended to arbitrary services, i.e. a framework for device access should not be restricted to a certain class of applications and associated protocols. 2) Devices providing certain services do not need to be found “manually” by the user but can rather be discovered automatically. 3) While numerous service discovery protocols are available today, those mostly address rather closed and static deployment environments (see also section 3). The closed service location environment is opened up to support “external” users and even enable public services.

In this paper, we combine ideas from auto-configuration, service discovery, and component integration to design and implement a protocol framework for Dynamic Device Association (DDA). DDA includes discovering, potentially locating, and associating with devices for authorized users; allows for bootstrapping application protocols; and provides means for dissociation after use. All DDA components are based upon standardized protocols as far as possible. In section 2, we present a general outline of the problem and identify the various steps of a generic solution. In section 3, we review related work for each of the necessary steps based upon which we devise a solution and present our system design in section 4. We conclude this paper with a summary and a discussion of open issues as well as future work in section 5.

² E.g., by using CTI solutions such as TAPI [MS99] or SIP third party call control [RPSC02].

2 Conceptual Overview

In a sample mobile user scenario, Alice and her visitor Bob meet in a conference room of Alice's company. Alice brings her laptop, Bob his PDA, both use the available WLAN. The conference room is equipped with a SIP-based conference phone and a regular SIP phone. Alice's laptop and Bob's PDA find two devices offering telephony services and obtain their locations and labels. Alice attaches to both devices and uses the conference phone to place a call to Carol with whom they are supposed to have a tele-chat. Alice and Bob use the speakerphone for the voice conversation and they use Alice's laptop to add a slide presentation to the call. During the teleconference the speakerphone is not accessible for others. Bob is only allowed to access the regular SIP phone to which he also attaches. Both may receive incoming calls through the regular SIP phone, but only Alice may place outgoing ones. After the conference call completes, Alice and Bob dissociate from the two phones and leave.

From this scenario, we can roughly identify the following five phases of dynamic device association:

1. Service and Device Discovery

Initially, the mobile devices need to find services offered by other devices. The discovery obviously involves an identification of the services, their availability (free vs. in use), their location information (to determine physical proximity), and a rendezvous address.

2. Device selection

Once a set of suitable devices has been found, the user may select a particular device. This selection process may also be fully automated (by user preference, physical proximity, or some other algorithm).

3. Service and device association

As soon as the user has picked a particular device, an association process is invoked by the mobile device. The selected service is contacted and an authentication procedure is carried out, after which the application protocol is bootstrapped: all necessary configuration parameters are exchanged and the application protocol is initialized.

4. Application protocol operation

The application protocol is run in the context of the dynamically established association. This may involve all kinds of interactions between the mobile and the associated device.

5. Service and device dissociation

When the associated device is no longer needed, the user's device dissociates from the device, freeing all allocated resources, and potentially making the device fully available again to the public.

Different protocols may be employed to implement the necessary mechanisms for each of the above phases. The following section discusses related work in this area, particularly regarding phases 1 and 3.

3 Related Work

Service discovery and the establishment of communication session between peers is addressed by multiple protocols and architectures, including but not limited to the *Service Location Protocol* (SLP, [GPVD99]), *Universal Plug and Play* (UPnP, [MS00]), *Salutation* [SAL99] and *Jini* [SUN01]. We will compare SLP and UPnP in the following.

3.1 Service Location Protocol

The *Service Location Protocol* (SLP, [GPVD99]) is a framework for service discovery and service selection in IP networks. It is intended for automating the configuration of applications that want to use services such as network printers or remote file systems in an administrative domain.

SLP is a very lightweight protocol: Essentially, an SLP user agent (a client application that is looking for a service) sends a request for a service and receives responses from service agents (entities that advertise services) or directory agents (entities that aggregate service advertisements from multiple service agents and can answer requests on their behalf). The request can either be directed to a service agent or directory agent or can be sent via multicast to a standard multicast group using a well-known port. Service agents never advertise their service using multicast or broadcast announcements. In order to increase scalability for multicast requests, a user agent can add the addresses of service agents that have answered before to a *previous responders list*. A service agent that sees its address in this list of a multicast request does not answer the request. Directory agents are intended for larger networks in order to enhance scalability.

User agents can specify three types of information in service requests: scope identifiers, a service type³ and a query predicate. A filter predicate can optionally be specified to query a service with respect to its attributes. After receiving responses to service requests a user agent can query the service's attribute explicitly by sending an attribute request. SLP also provides optional authentication but does not provide confidentiality.

SLP provides a solution to locating services, not to associating with the services. Service association (i.e., allocating a service resource, exchanging confidential access credentials) has to be done in a second step, with a different protocol.

3.2 Universal Plug and Play

Universal Plug and Play (UPnP, [MS00]) is an architecture for device control in local networks and relies on different protocols for functions such as service discovery and description, transmission of control commands and event notifications. It provides peer-to-peer communication between different types of devices without the need for manual configuration.⁴ To allow these devices to interwork, they do not only need to locate each other but must also dynamically learn their capabilities and be enabled to exchange information without knowing in advance which devices are present. The following functions can be distinguished:

Device discovery: UPnP's *Simple Service Discovery Protocol* (SSDP, [GCL⁺99]) relies on periodic unsolicited service advertisements from devices *and* on explicit service requests from clients (*control points*). Devices periodically send service advertisements containing information about the device type, an identifier, a URI for more information about the device and a duration, for which the advertisement is valid. There is no mechanism in SSDP to scale the announcement interval with the number of UPnP devices.

Device description: A UPnP service description is an XML document that describes the service's interface (in terms of methods and variables that can be inspected and modified. After discovering the de-

³ The service type is a fragment of an SLP *service URI*, a URI that can be used to specify the type and the address of a service, e.g., `service:printer:lpr://printer.example.com`.

⁴ E.g., in home networks where devices from different vendors are connected to an IP network in an ad-hoc fashion.

vice, a control point retrieves the corresponding service description by sending an HTTP GET request.

Device control: UPnP relies on the *Simple Object Access Protocol* (SOAP, [BEK⁺02]) for remote procedure calls and variable manipulation.⁵ Neither SOAP-1.1 or UPnP in general define any security mechanisms to authenticate control points and to provide integrity and confidentiality of the communication.

Event notification: UPnP allows control points to subscribe to certain state variables of a service relying on the *General Event Notification Architecture Base* (GENA, [CaYG00]) that specifies the methods SUBSCRIBE, UNSUBSCRIBE and NOTIFY.

While SLP carefully addresses the issue of scalability, e.g., by minimizing the use of multicast messages, UPnP has obvious deficits: The service advertisement rules do not allow UPnP to scale to larger environments with many services. In addition, the control features of UPnP are completely insecure, lacking authentication of control points, message integrity and confidentiality. This makes UPnP difficult to use except for controlled environments.

4 System Design

When we compare SLP and UPnP we can state that SLP provides mechanisms for service discovery and selection in static enterprise networks without addressing the service association problem. It focuses on simplicity and scalability with respect to the number of user agents. We believe that SLP's exclusive usage of service requests by user agents can lead to sub-optimal behavior in the case of ad-hoc communication and mobility. UPnP is intended as a complete solution for service discovery and device control through SOAP. It is targeted at service discovery in "dynamic" environments where devices can be connected dynamically, hence relying on periodic service announcements, however without properly addressing scalability.

These observations have led us to develop a new design for a dynamic device association framework for dynamic ad-hoc environments that addresses the necessary scalability and security issues and is not tied

⁵ SOAP is a specification of how to represent remote procedure calls, including their parameters and responses, in XML documents, which are (usually) transmitted using HTTP.

to any particular application protocol. Figure 1 depicts the DDA process schematically. The concepts we present in the following refer to the five phases of device association that we have discussed in section 2.

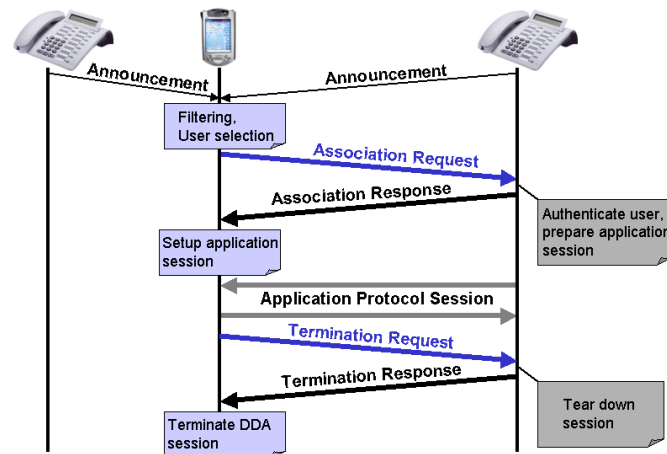


Fig. 1. The DDA process

4.1 Service and Device Discovery

When discovering services a user agent is interested in the following information: the existence of devices (in range) and their availability, the characteristics of the services they offer and their rendezvous parameters.

In principle, the discovery of existing and available services, is addressed by SLP and SSDP. However, to enable mobile user agents to discover (possibly dynamically changing) services offered by (potentially mobile) devices, the SLP mechanisms are not directly applicable:

For mobile devices that roam between networks, we cannot assume a static enterprise network with constant connectivity. A mobile device can enter and leave the scope of services quite frequently requiring the user agent to initiate new queries every time it enters a new network. Relying on the query-response model would also prevent the user agent from timely detecting that a service is no longer reachable. In cases of intermittent connectivity a user agent would thus have to send out queries periodically to validate its view of the available services in a network.

A query-response model does not allow a mobile device to build up and maintain an ephemeral “directory” of available services: this would require either iterating through all conceivable service names (which appears infeasible) or using wildcard queries at the risk of response implosions. Finally, relying on directory agents announcing their availability is not a suitable fallback: some wireless ad-hoc networks may be affected by the *hidden terminal problem*: this could lead to situations where a mobile device can communicate with the directory agent but not with the corresponding service agents.

As a solution, we propose an announcement-based scheme, where services actively announce their availability and user agents receive and filter announcements depending on the service description. This approach allows user agents to discover services more easily, especially in environments with mobile user agents and ad-hoc-networking characteristics. SSDP also relies on an active announcement model but does not provide a rate-adaptation scheme and thus does not scale to large numbers of services. In our approach service agents announce themselves by multicasting advertisements, and service agents participate in a *rate adaptation process*: Each service agent observes the announcements of other service agents per time interval and adapts its own transmission rate so that the total announcement data rate roughly remains constant, regardless of the number of service agents. Service agents also announce discontinuation of their service. User agents receive service announcements but can optionally also send explicit service requests, and service agents include their current estimate of the group size in their announcements in order to inform user agents and other agents about their view of the group.

Using active announcements with a well-defined rate-adaptation scheme has several advantages: User agents do not have to poll to detect the availability of services, instead they receive announcements automatically when connecting to a network. User agents can also detect when services are no longer available by monitoring the group size and calculating the retransmission interval themselves, automatically expiring services that cease sending announcements after some time.

Instead of static service descriptions, we propose the use of a *soft-state* approach, where the latest announcement is conveying the current service description, which is more practical for also accommodating dynamic parts in the service description such as service availability.

In order to provide user agents with enough meaningful information as a basis for the service selection, the service descriptions should include at least the following information in addition to the service type:

- the session protocol that is used to access the service;
- the current availability of the service and its geographic location;
- service attributes that describe the capabilities of the services; and
- a service URI (the “rendezvous point”) to associate with a service.

The DDA framework provides all this information in the periodic service announcements as well as in responses to explicit service queries, thus differing from SLP’s two-stage request-response procedure. For implementing the announcement-based, scalable DDA service discovery procedure, we have selected the *Session Announcement Protocol* (SAP, [HPW00]).⁶ SAP announcements carry a description of a communication session, including transport parameters that are required to join the session. Today, SAP is almost always used with the Session Description Protocol (SDP, [HJ98]): a simple text-based format that can be described as a key-value scheme, where a key is a single character.

Figure 2 is a sample DDA service announcement.⁷ The *session level* section (up to the first ‘m=’ line) contains general information about the device; each *media level* section describes a service and provides details how to access it. The media field indicates the media type `dda-control`, and the protocol field specifies the used application session protocol, e.g., HTTP, HTTPS, SIP, or MBUS. Figure 2 illustrates the use of SDP for HTTPS and Mbus. The service URI is specified in the attribute `dda-connect`. The following session and media attributes are used (the attributes are always prefixed with `dda`): `device-type` to specify the announcement type and the session protocol, `device-id` to distinguish multiple devices of the same type, `device-location` to allow for device selection based on geographic locations, `stats` to indicate the number of visible service agents in a given scope, and `device-status` to specify the availability status of a device. The example in figure 2 also provides an application specific attribute “`iphone`” that is used to specify parameters for this special device type.

⁶ SAP is a multicast-based protocol for the advertisement of multicast multimedia sessions. The protocol provides scalability with respect to the number of sessions that are announced in total and the amount of bandwidth that is used, which is achieved by adapting the announcement interval to the total number of SAP announcements.

⁷ Some lines that are not related to DDA have been omitted.

```

a=app:dda mbus
a=dda-device-id:32778be73ef9823097d22957b3e5809a
a=dda-device-location:MZH%205160
a=ip-phone:SIP dku's%20phone Example.com%20Phone-X
a=dda-stats Local 42
m=dda-control:443 HTTPS -
a=dda-connect:https://10.1.2.3/connect
a=dda-device-type:ip-phone
a=dda-device-status:AVAILABLE

```

Fig. 2. Sample DDA service announcement (abbreviated)

4.2 Device Selection

The actual device selection can be realized in different ways, depending on the application, the network characteristics, and user settings: The user agent can be configured with a filter expression to consider only services of a specified type and with certain attributes. A preference-based ranking may be included as well, and physical proximity may be used as a criteria for automated device selection.

If the device selection process cannot be automated, the list of currently available services is presented to the user. Mobile devices may continuously monitor their environment and present a (structured) service directory to the user from which she can select services to access.

4.3 Service and Device Association

After a user agent has selected an appropriate service, the actual device association takes place. The user contacts the service URI, authenticates itself to the service and requests the service session parameters. The following requirements can be identified for these steps:

Authentication: Some services such as telephony and printing services may only be made available to authorized users. Therefore, DDA must support authentication of user identities. The specific mechanisms depend on the available infrastructure and the application scenario.

For corporate environments with a set of well-known users and an appropriate security infrastructure, public-key based mechanisms as

well as shared secrets (i.e. passwords) may be used. Guests may be authenticated using one-time (or one-day) credentials which may take either shape. With an inter-domain public-key infrastructure, guests may also be authenticated as individuals.

Our DDA scheme supports certificate-based authentication as well as password-based authentication. If no personalized authentication is possible, access passwords can be provided, e.g., at a help desk.

Confidentiality and integrity: After the user has been authenticated the parameters for the actual session protocol have to be negotiated. Since this may include sensitive information such as transport parameters that should not be disclosed and keying material for securing the service session itself, this data exchange may need to be secured with respect to confidentiality and message integrity. Confidentiality is achieved by encrypting the communication; integrity is accomplished by relying on hashed message authentication codes (HMACs).

Session protocol parameter description: One or more services can be described in a session description. The DDA framework is not tied to any specific session protocol, and hence it must be possible to describe session parameters for different protocols such as SIP [RSC⁺02], Mbus [OPK02], HTTP [FGM⁺97], and SOAP [BEK⁺02]. Independent of the specific protocol in use, the following parameters are described: an explicit lease duration, a service URI for re-associating and a service URI for dissociating.

We have mapped the service association step to an HTTP GET request: The user agent requests the session configuration, possibly providing authentication credentials for the user. After the user has been authenticated, the service agent provides the configuration data in the HTTP response. We have considered two mechanisms for user authentication: The user agent can either connect to the server using HTTP or HTTP/TLS. In case of HTTP/TLS, the user agent and the service agent can authenticate themselves using certificates. Alternatively, e.g., if a public key infrastructure is not available and certificates cannot be validated, the user can be authenticated through HTTP digest authentication.

Figure 3 depicts a sample session description.⁸ The parameters for the session are described in a media level section of the SDP

⁸ Some lines that are not related to DDA have been omitted.

```

a=dda-connect:https://10.1.2.3/connect/4367-abc4-9786
a=dda-disconnect:https://10.1.2.3/disconnect/4367-abc4-9786
a=dda-lease:600
m=control 47000 MBUS -
c=IN IP4 224.224.224.224
a=mbus:HASHKEY=(HMAC-MD5-96,T21/U6/0RLxKF/0a)
a=mbus:ENCRYPTIONKEY=(NOENCR)
a=mbus:SCOPE=LINKLOCAL

```

Fig. 3. Sample DDA session description (abbreviated)

description. The `m=` line must specify the type `control`, followed by a port number (if applicable) and a protocol identifier. The session level section includes the lease time (`a=dda-lease`), a URI for lease-renewal (`a=dda-connect`), and may also provide a URI (`a=dda-disconnect`) to explicitly terminate the application session. All other parameters are given as session protocol-specific attributes. The session description uses the attribute `mbus` that is used multiple times to specify different Mbus communication parameters.⁹

Figure 4 shows the complete operation of a DDA process: the user selects an IP phone, connects to it using TLS, thereby authenticating the device and establishing a secure communication link. HTTP digest authentication is used to verify the mobile user's identity and the application session parameters are conveyed in the HTTP body. The application protocol operates and, when the lease expiration time nears, the DDA process is re-invoked to refresh the lease. Eventually, the application session and the device association are terminated.

4.4 Application Protocol Operation

After the mobile device has obtained the session description it can start using the corresponding service. Depending on the application protocol in use, the two associated devices may continue operation in more or less tight coupling: Some protocols (such as Mbus) support liveness monitoring of peers: this allows one device to notice that the other has disappeared and terminate the application session as well as the device association. Other protocols, e.g., HTTP based protocols, do not support

⁹ We have defined DDA fields for the required Mbus parameters (see [OPK02]).

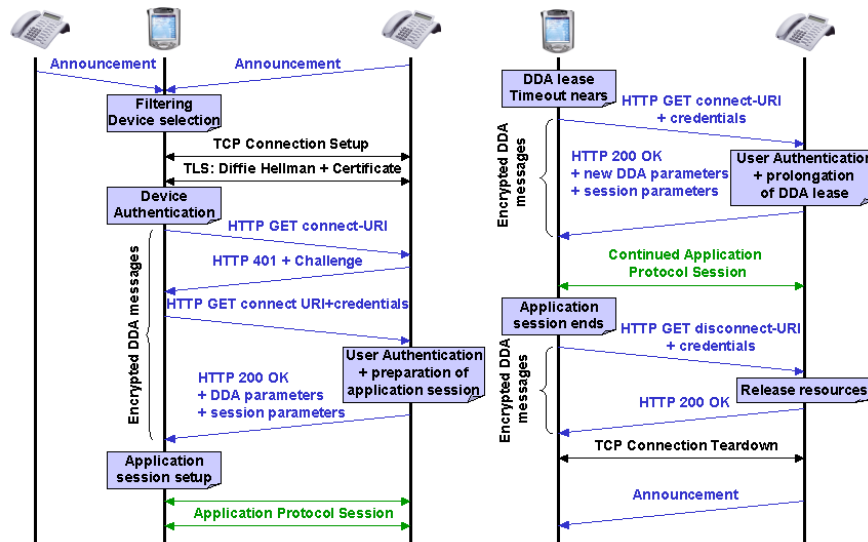


Fig. 4. Full sequence of implemented DDA interactions

liveness detection, which means that services could be blocked indefinitely if no other mechanisms exist to terminate application sessions and device associations.

We therefore propose the usage of *leases*, i.e., explicitly limited usage durations: a service agent specifies a maximum lease time and, if a user agent wishes to use the service beyond this time, it has to re-associate with the service agent using the corresponding URI and obtain a new lease (with repeated authentication and authorization and potentially new parameters). If the user agent does not re-new the lease the service agent can assume the session has terminated when the lease expires.¹⁰

4.5 Service and Device Dissociation

For service dissociation, we have to consider similar issues as discussed in the previous section: If an application protocol supports the notion of application sessions and provides mechanisms for their tear-down, those mechanisms may be used as a hint to also terminate the device asso-

¹⁰ The concept of leases is used by other protocols as well such as DHCP [Dro97] and the Jini architecture [SUN01].

ciation.¹¹ For other protocols, DDA-based mechanisms for dissociation and for dealing with failure situations are required. The DDA support for terminating device associations is hence twofold: first, the service agent provides a dissociation URI to be used by the client for explicit termination. This is complemented by the lease concept which ensures that service agents eventually may terminate sessions: implicitly in case the user agent has disappeared or explicitly by denying a re-association attempt of a user agent.

5 Conclusion

The Dynamic Device Association concept generalizes the ideas of service location and bootstrapping of communication sessions. The service location and selection functionality is usable in different network environments: statically configured enterprise networks, ad-hoc-networks, and networks with both mobile user agents and mobile service agents. The device association is not limited to specific application protocols and provides secure authentication to dynamically located services. We have presented some sample scenarios for which DDA can be useful, e.g., the dynamic association and control of “public” IP telephony systems.

DDA builds on some concepts that have been implemented in other protocols before – we have especially considered SLP and UPnP and have observed some shortcomings. These observations have led to a different approach with a scalable service announcement mechanism and a generic, secure service association procedure.

We have implemented these concepts in a protocol specification and have applied the DDA protocol to the dynamic association of IP phones using PDAs and laptops with “DDA browsers”. The PDA applications can locate available IP phones, personalize and control these devices. The combination of SAP and HTTP with SDP turned out to be a good compromise yielding a scalable, secure, and extensible solution that meets the requirements of our target application scenarios.

For future enhancements, we are considering to develop suitable mechanisms for scoping and automatically selecting devices of interest (e.g. using physical proximity), and we would like to investigate the practical issue of segmented networks and firewalls within enterprises and

¹¹ Examples are the `mbus .bye ()` command and the RTCP BYE packet.

their influence on dynamic device association. Finally, support for additional application protocols will be added.

References

- [ABL02] V. Azondekon, M. Barbeau, and R. Liscano. Service selection in networks based on proximity confirmation using infrared. International Conference on Telecommunications (ICT), Beijing, China, 2002.
- [BEK⁺02] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. *Simple Object Access Protocol (SOAP) 1.1*, May 2002.
- [CaYG00] Josh Cohen and Sonu Aggarwal and Yaron Y. Goland. General event notification architecture base: Client to arbiter. available online at <http://www.upnp.org/>, September 2000. Internet Draft, Work in Progress.
- [Dro97] Ralph Droms. Dynamic host configuration protocol. RFC 2131, March 1997.
- [FGM⁺97] Roy T. Fielding, Jim Gettys, Jeffrey C. Mogul, Henry Frystyk Nielsen, and Tim Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2068, January 1997.
- [GCL⁺99] Yaron Y. Goland, Ting Cai, Paul Leach, Ye Gu, and Shivaun Albright. Simple service discovery protocol/1.0, operating without an arbiter. available online at <http://www.upnp.org/>, October 1999. Internet Draft, Work in Progress.
- [GPVD99] Erik Guttman, Charles Perkins, John Veizades, and Michael Day. Service location protocol, version 2. RFC 2608, June 1999.
- [HJ98] Mark Handley and Van Jacobsen. Session description protocol. RFC 2327, April 1998.
- [HPW00] Mark Handley, Colin Perkins, and Edmund Whelan. Session announcement protocol. RFC 2974, October 2000.
- [MS99] Microsoft Corporation. *IP Telephony with TAPI 3.0*, 1999.
- [MS00] Microsoft Corporation. *Universal Plug and Play Device Architecture*, June 2000.
- [OPK02] Jörg Ott, Colin Perkins, and Dirk Kutscher. A message bus for local coordination. RFC 3259, April 2002.
- [RPSC02] Jonathan Rosenberg, Jon Peterson, Henning Schulzrinne, and Gonzalo Camarillo. Best current practices for third party call control in the session initiation protocol. Internet Draft draft-ietf-sipping-3pcc-02.txt, Work in progress, December 2002.
- [RSC⁺02] Jonathan Rosenberg, Henning Schulzrinne, Gonzalo Camarillo, Alan Johnston, Jon Peterson, Robert Sparks, Mark Handley, and Eve Schooler. Sip: Session initiation protocol. RFC 3261, June 2002.
- [SAL99] The Salutation Consortium. *Salutation Architecture Specification V2.1*, 1999.
- [SUN01] Sun Microsystems. *Jini Architecture Specification Version 1.2*, December 2001.