Semantic-based discovery to support mobile context-aware service access

Alessandra Toninelli *, Antonio Corradi, Rebecca Montanari

Dipartimento di Elettronica, Informatica e Sistemistica, University of Bologna, Viale Risorgimento, 2-40136 Bologna, Italy

Available online 3 January 2008

Abstract

The increasing diffusion of portable devices with wireless connectivity enables new pervasive scenarios, where users require tailored service access according to their needs, position, and execution/environment conditions (context-aware services). A crucial requirement for the context-aware service provisioning is the dynamic retrieval and interaction with local resources, i.e., resource discovery. The high degree of dynamicity and heterogeneity of mobile environments requires to rethink and/or extend traditional discovery solutions to support more intelligent service search and retrieval, personalized to user context conditions. Several research efforts have recently emerged in the field of service discovery that, based on semantic data representation and technologies, allow flexible matching between user requirements and service capabilities in open and dynamic deployment scenarios. This paper proposes a middleware-level approach to support user-centric semantic service discovery. The presented middleware, called AIDAS, exploits context-awareness based on user/device/service profile metadata to provide personalized views on services of interest, and supports semantic-based matchmaking between requested and offered service capabilities. In addition, AIDAS addresses the crucial management issue of providing resource-constrained portable devices with needed semantic support features. Semantic support services, such as ontology repositories and inference engines, typically require a large amount of computational and memory resources that might not fit the properties of all mobile devices. AIDAS addresses this issue by transparently and dynamically adapting semantic-based discovery support to the properties of different access devices.

© 2007 Elsevier B.V. All rights reserved.

Keywords: Discovery; Context; Semantic web

1. Introduction

Recent advances in the computational capabilities of portable devices, such as cellular phones and palmtops, together with their increased wireless connectivity have favored the emergence of pervasive service provisioning scenarios. Mobile users can access needed services from ubiquitous attachment points and even when changing physical locations, e.g., at their workplaces, at home or at publicly accessible places, such as airports and shopping malls. Moreover, users can increasingly benefit from services whose results adapt to the changing context, such as variations in users’ position, preferences and requirements, and in locally available resources (context-aware services) [1]. Context is a complex notion that has many definitions. Here, we consider context as any key information characterizing the user, e.g., user preferences, needs, location, and any useful information about the environment where he operates, e.g., date, time, on-going activities and interactions with services and/or other users.

In these context-aware service provisioning scenarios, it is crucial to enable the dynamic retrieval of available services in the nearby of the user’s current point of attachment, while minimizing user involvement in service selection, configuration and binding. Service discovery in pervasive environments, however, is a complex task as it requires to face several technical challenges at the state of
the art, such as user/device mobility, variations (possibly unpredictable) in service availability and environment conditions, and terminal heterogeneity. Users might need to discover services whose names and specific implementation attributes cannot be known in advance, while service providers need to advertise services to clients whose technical capabilities and conditions at interaction time might be mostly unpredictable beforehand. In addition, service providers cannot exactly define and code at design time all possible configurations of devices accessing the service, e.g., by including any possible discovery protocol and data format.

In the past few years, industry and academia have investigated and proposed various discovery solutions and protocols, such as the Jini discovery architecture [5], the Bluetooth Discovery Protocol [2], the UPnP Simple Service Discovery Protocol [3], the IETF Service Location Protocol [4]. Other discovery protocols have also emerged within the Web Services research community, such as ebXML [6] and UDDI [7]. All these protocols rely on exact syntactic-based matching techniques based on unique identifiers, interfaces, names or XML-based keywords generally defined within fixed, standard taxonomies. However, the exact matching of patterns or keywords might be too restrictive when applied to dynamic and heterogeneous pervasive environments where little or no prior agreement exists on service description and user request format. Service discovery might possibly fail due to the syntactical mismatching between service names. For instance, if a user is looking for a “News” service, a service called “Information” would not match with the request, although their respective meanings are clearly compatible.

To overcome the expressivity limitations of traditional discovery models and improve discovery effectiveness, the emerging design guideline for novel discovery solutions is the adoption of semantic languages. Semantic languages permit explicit representation of interacting entities, e.g., services, resources and users, at a high level of abstraction while enabling automated reasoning about this representation, thus favoring interoperability and mutual understanding between entities sharing little or no prior knowledge about each other. Several research efforts have also emerged in the field of Web Services to enhance current discovery solutions with semantic-based descriptions (mainly RDF- or OWL-based) and advanced matching techniques, ranging from logic-based to similarity-based strategies [8,9,29–32].

However, we believe that discovery solutions for pervasive environments can be further improved and brought to their full potential. Primarily, there is the need for a paradigm shift from an “administratively-scoped” to a “user-centric” discovery approach [33]. Current discovery approaches typically locate services within administratively defined network domains, which might correspond to logical or physical environments. However, from the end user’s perspective, it is often necessary to locate and retrieve services not only based on logical and/or physical closeness, but also taking the user’s context into account.

Therefore, service searching scope needs to be filtered according to finer-grained criteria other than administrative or network grouping. All and only those services that are semantically compatible with the user’s context should be automatically and transparently made visible to him. The exploitation of user’s context-awareness in service discovery helps mobile clients saving time and efforts in service selection. Current semantic-based approaches, such as [8,9,29–32], provide powerful and flexible matching strategies but are mainly targeted at supporting automated service composition rather than personalized user-centric discovery. In addition, they generally require complex semantic service descriptions and matching algorithms that might be too complicated if compared with the average needs of typical mobile usage scenarios. For instance, discovery often occurs in situations where users with variable degrees of technical expertise cannot devote much attention to service retrieval and selection operations. In addition, non technical users might feel more comfortable in specifying service requests in terms of expected capabilities rather than required service inputs and outputs.

On the other hand, the potential of semantic-based discovery has not been fully exploited yet because of various management issues, which seem to be still open. Access terminals usually exhibit relevant differences in resource capabilities, such as display size and resolution, computing power, memory, network bandwidth, and battery. A crucial management crucial issue remains how to provide support for semantic-based discovery to mobile devices with limited capabilities. Semantic support services, e.g., ontology repositories, inference engines and knowledge management tools, typically require a large amount of computational/memory resources that may not fit the properties of mobile devices. In particular, strict limitations exist about the kind of semantic support facilities that can be hosted on resource-constrained devices. For example, executing a reasoning process on board of a resource-limited device, such as a smart phone, might not only consume battery, but more important, it would probably monopolize all available memory resources, thus making the execution of other applications very difficult.

This paper describes the implementation of the above discussed design guidelines within the AIDAS (Adaptable Intelligent Discovery of context-Aware Services) framework that supports personalized, user-centric and semantic-based discovery for mobile users. In particular, AIDAS adopts semantic languages to model the properties of interacting entities and the environment, and it relies on DL-based subsumption reasoning for matching user requests and service offers. As a key feature, AIDAS integrates semantic data representation and matchmaking support with facilities for context management and context-based service filtering. Context visibility allows AIDAS to narrow the range of discovery: service selection filters out all services that could/would not be accessed by the user, either because of user device’s technical limitations, or because the user is not interested in that kind of service.
Another distinctive feature of AIDAS is its wide set of mechanisms for making semantic-based discovery viable even to resource-constrained portable devices, for example by allowing a device to install on-board or download or remotely access needed semantic services depending on device properties. AIDAS extends the model presented in [19] and [28] in its metadata model and, more notably, in its support for dynamic configuration of semantic support services to fit different technical properties of heterogeneous devices.

The paper is organized as follows. Section 2 describes the configuration model adopted in our framework. Section 3 presents an overview of the middleware architecture, while some implementation insights are provided in Section 4. The usability and effectiveness of the proposed middleware is shown through the discussion of a case study in Section 5. Section 6 provides some evaluations about the performance and accuracy of the prototype implementation. Related research work is presented in Section 7. Concluding remarks and future research directions are given in Section 8.

2. AIDAS metadata model

AIDAS adopts semantic-based metadata (profiles) to describe properties and characteristics of the entities involved in the discovery process, i.e., services and clients. AIDAS models the concept of service as a “black box” providing some functionalities, while a client represents any entity wishing to access those functionalities. A client might be a human user or a software component, such as a service broker in charge of composing single service instances into a complex one. AIDAS associates each client with its context, which defines any client-related information relevant to the discovery process. In case of a user client, context might include personal information about the user, such as spoken languages, as well as technical features of the user’s device.

AIDAS metadata model is specifically designed to support personalized user-centric discovery. In particular, the model focuses on the representation of service/user/device capabilities and requirements, whereas other parameters, e.g., service preconditions, postconditions, effects and assumptions taken into account in other solutions [39,23], are not considered relevant since they mainly serve the purpose of supporting automatic service composition. As a key feature, AIDAS metadata model provides fine-grained profile modularization to improve precision and effectiveness in service selection. The comparison between service request and offer is performed by considering single capabilities and requirements of interest rather than the entire service profile. In addition, device profiles are exploited to refine the searching scope.

Finally, AIDAS allows users to further personalize service discovery by specifying preferences. In particular, users can define in advance a priority order amongst requested service capabilities, thus being relieved from the burden of manually selecting and ordering discovery results. Furthermore, users can specify which requirements to possibly relax in order to extend the set of potentially interesting services.

2.1. Service metadata

Each service is described by a static and a dynamic profile. The static profile collects data that are relatively stable over time, such as service name and functions, or do not depend on dynamic operating conditions. Conversely, the dynamic profile includes information that might frequently change, e.g., location and state of the application.

2.1.1. Static profile

Each static profile consists of four sub-parts, namely: identification, service capabilities, service requirements, and service interface. Identification information provides information to name a service and identify its location. Service capabilities basically represent logical units of service functionality. For example, a service capability might be “updating software via SSL from a remote site”. In our model a capability might be either core, i.e., describing the service specific activity, or functional, i.e., concerning properties that do not describe the service activity, but how this activity is performed. Referring to the previous example, the service has the core capability of “updating software” and the functional capability of “communicating over SSL”. Let us note that the latter is not specifically related to the service main activity since a software update might be performed regardless of any SSL support. A capability example, which provides news from newspapers, is shown in Fig. 1. This capability consists of different blocks, each pertaining to a specific concept (denoted with the has_info_entity property), such as available newspapers and supported languages. Service capabilities allow service providers or advertisers to describe their services in terms of offered functionalities. Conversely, users express their requests in terms of requested capabilities.

Service requirements represent access conditions to a service. In particular, each requirement is modeled as a required capability, i.e., it defines which kind of capability a client should possess to be allowed/able to access the service. For example, a remote software update service might require that the client device runs a previous version of the software needing update. We distinguish between hard and soft requirements. While hard requirements are mandatory, soft requirements might be defined with variable degrees of priority and importance, for example by means of a scoring function. An example of service requirement is shown in Fig. 1. The req_strength property might assume either the value hard or soft, while the requires property defines the required capability. Different kinds of specific requirements, like for instance security or technical requirements, might be subclassed from the req_strength class.

To represent service capabilities and requirements, we adopt a semantic approach. AIDAS data model defines a
base ontology for capabilities/requirements, while application-specific ontologies can be defined for each application scenario. Ontologies are specified in OWL-DL [15].

The third part of the static profile, i.e., the Service interface includes or points to needed information for actual service invocation, such as input/output description or invocation endpoint. Let us note that such information might be provided according to different specifications, depending on the interface implemented by the service, e.g., a method signature for a Java object or a WSDL profile.

2.1.2. Dynamic profile

The dynamic profile describes service properties that might vary over time. In particular, it includes information about the service operating conditions (service state). Those conditions are dynamically retrieved via external information sources that provide, for example, information about service availability and load, or the average response time to service invocation.

2.2. User metadata

Clients are described in terms of profiles and preferences. User profiles are composed of both dynamic and static metadata. Dynamic properties include, for example, user locality and state, while static properties are grouped into three categories (analogously to the case of a service): identification, capabilities and requirements. Identification information, such as an ID code, a string or an URL, is used to name and identify the user. Capabilities represent the user’s abilities and capacities, such as speaking a language. User requirements describe user-defined constraints that must be always satisfied during service provisioning. Fig. 2 shows an example of user static profile.

Preferences allow the user to express desired choices about service results and to relax constraints on service requests [21,24]. AIDAS model defines two types of preference: value preferences and priority preferences. Value preferences specify the preferred value for a service property. Depending on the property value type, a preference might be either object or data type. Value preferences are modeled using an ontological approach and expressed in OWL. In particular, object preferences are modeled by means of the OWL ObjectProperty construct, while data type preferences are modeled by means of the OWL DatatypeProperty construct. The basic preference ontology is represented in Fig. 3 [27]. As shown in the diagram, a value preference defines the desired property value for a given capability, one or more explicit alternative values, and the kind of acceptable constraint relaxation. Let us note that, in case of a datatype preference, such relaxation defines an interval of acceptable values, based on the preferred numerical value, while in case of an object preference
we define three possible semantic relations between the ideal value expressed in the request and other acceptable values, namely: exact (default), plug-in, subsumes [23]. “Exact” represents the case of a perfect match between required and offered capability for the property constrained by the preference. In the “plug-in” case, request subsumes offer, therefore the user may consider acceptable a service providing a more specific value than the requested one. For example, a user looking for a service providing news might accept a service providing news only provided by newspapers. In the “subsumes” case, offer subsumes request, hence the user may decide to accept a service whose capability is more general than the requested one.

Since a request for service may include several capabilities and a capability might have multiple properties, AIDAS allows the user to define a priority order amongst capabilities/properties by means of priority preferences. For example, let us consider the capability of providing news. If a preference assigns higher priority to “language” rather than “topics”, the former will be tested first when performing matching and a service with a good value for this property will be considered more compatible than another having a better value on the “topics” capability.

2.3. Device metadata

Device metadata describe the technical characteristics and operating conditions of a user device. Similarly to service and user profile, the device profile includes static and dynamic metadata, and is composed of the identification, capabilities and requirement parts. The identification part includes device category and type, as well as names and parameters that allow device identification within a network, such as the MAC address or the Bluetooth ID number. Device capabilities describe technical characteristics, supported functions and resource state, such as memory
storage size, secure socket layer support, and battery level. Finally, device requirements specify technical requirements that other devices must meet to access services hosted on that device. For example, if a device is able to connect to another device only via Bluetooth, then Bluetooth connectivity represents a requirement for that device.

3. Middleware architecture

AIDAS provides various middleware services organized into two different logical sets, with each set targeted at addressing specific management functions. The discovery management set provides the needed functionalities to support service discovery and selection based on user’s context information and preferences, as expressed in service queries. The configuration management set provides the needed facilities to allow each portable device to advertise hosted semantic functionalities to co-located devices, to discover locally available discovery management facilities, and to choose whether to download on-board or remotely access needed semantic services based on device properties. AIDAS logical architecture is shown in Fig. 4.

3.1. Discovery management services

The Metadata Manager (MM) provides support for the specification, modification, checking for correctness, installation and evaluation of different types of semantic metadata. MM provides templates to support the user in the task of specifying user/device/service profiles. The use of templates ensures that metadata are encoded in the correct format, i.e., compliant to AIDAS profile ontology, and offers to non technical users a friendly interface for profile specification. Let us note that MM does not perform semantic reasoning, but only syntactic compliance checking.

The Discovery Manager (DM) is responsible for determining and maintaining the list of services that are visible/accessible to the user based on her context. In particular, among all services available in the user’s network locality, DM selects the ones whose profiles are semantically compatible with user/device profile. The degree of compatibility between service and user/device profile is determined by applying to service capabilities and user/device requirements the semantic matching algorithm implemented by PME. For example, let us consider the case of a user unwilling to use her credit card online. In this case, DM would not include in the user’s personal view any service requiring payment via credit card. Similarly, if the user device does not support the Bluetooth protocol, DM would not include in the service view services accessible only via Bluetooth.

The Context Manager (CM) is responsible for creating user contexts when AIDAS users initiate their discovery sessions, for monitoring changes in both created user contexts and in relevant external environment conditions, e.g., the addition of new services, for notifying changes to interested entities, and for updating user contexts accordingly.

The Query Processor (QP) is in charge of collecting and processing user requests for service. QP interacts with the user via the User Proxy (described in Section 3.2) to specify required service capabilities and user preferences. In particular, QPE translates value preferences expressed by the user at access request time (see Fig. 7) into property restrictions. Let us note that QPE can also express disjunctive queries, which are specified by means of the OWL UnionOf construct.

The Profile Matching Engine (PME) is responsible for performing a matching algorithm between user/device requirements and service capabilities, taking user preferences into account. PME is requested to perform its algo-

![Fig. 4. AIDAS middleware architecture.](image)
rithm in two cases, i.e., when DM needs to determine the list of visible services for each user (i.e., the services whose profile is semantically compatible with user/device profile) and when QP needs to resolve a specific user query. In the first case, PME receives from CM user/device profiles and from DM the profiles of locally available services. In the second case, PME interacts with DM to be provided with the list of visible services and their profiles. In both cases, the static profile is used to perform direct matching, i.e., between user and device requirements and service capabilities. In particular, for each capability required by the user, PME verifies if the service profile contains one or more compatible capabilities. The matching algorithm is described in detail in the next section. The same algorithm is re-applied to the output of the direct matching to perform inverse matching, i.e., to match service requirements against user/device capabilities.

When determining the list of user visible services, PME performs the matching algorithm on all locally available services. On the contrary, when processing a specific request, PME may either stop at the first occurrence of a compatible service, or perform the algorithm on each visible service. In the first case, PME returns to CM a reference to a single service, while in the latter case it returns a list of services, ordered by compatibility degree.

3.2. Configuration management services

The User Proxy is an application-independent middleware component that represents a portable device on the fixed network [1]. We adopt the general emerging design guideline of dynamically extending the Internet infrastructure with proxies acting on behalf of (possibly disconnected) limited devices [1,35,36]. Proxies can handle several complex management tasks transparently to client devices by hiding network heterogeneity and managing network dynamicity. AIDAS associates one UP with each portable device, with a 1-to-1 mapping. UP covers various management roles, including retrieving the profiles of its companion user/device and coordinating with discovery management services during a discovery session. In addition, UP has the crucial responsibility of configuring AIDAS discovery management facilities on the device. Based on the device profile, UP decides where to allocate the various discovery management services, either completely/partially on board of its companion device or remotely on other trusted devices. In particular, UP broadcasts its device static profile to nearby devices specifying which discovery management services its companion device hosts on-board, and receives similar advertisements from nearby devices. UP behaves differently in the case of rich- or resource-constrained devices. In the case of rich devices, UP runs management functions and instantiates all discovery management services (MM, DM, CM, QP and PME) on-board. Having completely on-board discovery management facilities allows to better preserve user privacy since requirements/capabilities and personalized service view are locally stored. In addition, on-board reasoning may be useful in case the user’s device experiences frequent disconnections or poor bandwidth connectivity. On the contrary, in the case of resource-constrained devices, UP’s behavior is a tradeoff between technical device characteristics and privacy needs. In particular, UP decides whether to perform its management operations on-board or remotely, and which discovery management services to instantiate on-board, based on dynamic device profile data, such as CPU, memory size and battery level. Discovery facilities that cannot be hosted on-board are retrieved by querying nearby devices. It is worth mentioning that privacy issues might arise when user data are transmitted to foreign nodes. The choice about the desired tradeoff between privacy and discovery efficiency is left to the user. AIDAS metadata model allows the user to specify desired configuration choices in the user requirement profile part.

4. Implementation details

We have developed a prototype implementation of the AIDAS middleware to be deployed in a wireless Internet scenario, i.e., a computing environment where wireless solutions extend the accessibility of the fixed Internet infrastructure via access points, working as bridges between fixed and mobile devices. We have started implementing AIDAS in the wireless Internet scenario as we consider it the most significant deployment setting for our user-centric, semantic-based discovery. Other implementations of AIDAS for mobile devices connected via low range protocols, such as Bluetooth, are currently under consideration. In the following, we use the term network locality to identify a LAN via access points, working as bridges between wired and wireless devices. As shown in Fig. 4, AIDAS prototype is built on top of the Java-based CARMEN system that supports the provisioning of context-dependent services to portable devices [1]. CARMEN provides any portable device with a companion middleware proxy (shadow proxy) that, in case the device is resource-constrained, autonomously acts on its behalf over the fixed network, follows user/device movements among network localities, and properly configures AIDAS service allocation. CARMEN implements shadow proxies by exploiting the Mobile Agent programming paradigm [17]. In particular, CARMEN provides proxies with execution environments, called places, that typically model nodes. Places can be grouped into domains that correspond to network localities, e.g., either Ethernet-based LANs or IEEE 802.11-based wireless LANs. In the AIDAS deployment scenario, mobile devices entering a locality exploit AIDAS facilities to perform semantically-enhanced discovery of available services, according to the user’s current context conditions and explicit preferences.

4.1. Prototype implementation

The Discovery Manager is in charge of determining a personalized view on services when a user starts a discovery
session. DM includes a service registry component that allows service providers to advertise services and users to look for them via a publish/subscribe mechanism. In particular, service providers publish their service descriptions by filling in service templates provided by a DM-dedicated MM instance. This MM instance only provides checking for syntactic correctness and compliance to the AIDAS profile representation model. Static service profiles are stored in the registry at start-up time and associated with a service identification code to be referenced by other middleware components. Dynamic property values are not stored in the registry, but calculated at service access time via appropriate methods provided by the service interface. DM relies on PME matching algorithm to create a personalized user view on services based on the user’s context. In particular, user/device requirements and service capability metadata are first used as PME input parameters to reduce the set of potentially compatible services. PME matching is re-applied to this subset of services with service requirements and user/device capabilities as input parameters for further context-based pruning. In addition, DM updates discovery scopes for on-going discovery sessions when CM notifies that new services have been added in the locality or have changed their profiles. In this case, DM applies PME matching again to those services to verify whether to include them in the discovery scope.

The Context Manager (CM) is responsible for creating and managing user contexts. CM can update user context according to different strategies: at pre-defined time intervals, or upon any context change detection, or upon explicit user request. The adopted strategy is decided at middleware configuration time and depends on several factors, from user requirements to the desired trade-off between the need for fresh context information and the limitation of update overhead. CM has been developed by exploiting the context-awareness infrastructure and programming APIs of the Java Context Awareness Framework (http://www.daimi.au.dk/bardram/jcaf/). The current CM prototype implementation fully supports acquisition and management of static context information and of relatively simple dynamic information, e.g., time, location, and standardized monitoring indicators about device state, while we are still working on managing dynamic information harder to access in an open way, such as service load and expected network bandwidth/jitter.

The Query Processor (available at http://www.lia.deis.unibo.it/Research/MIDAS) is implemented in two sub-components: the Query Processing Engine (QPE), which is the core module performing automatic translation of user/service requirements into required capabilities processable by the Profile Matching Engine, and the Query Processing Interface (QPI), which is the interface module interacting with the user to define service requests and collect preferences. QPI interacts with the user via a graphical interface that guides the specification of required service capabilities and preferences. Once the user has specified service requests and preferences, these data are forwarded to QPE to be translated into OWL-based required service capabilities and related preferences. QPE then forwards the data to PME for semantic matching.

The Profile Matching Engine performs matchmaking between offered and requested capabilities to determine the degree of semantic compatibility between user/device and service profiles. The details of the matching algorithm are provided in the following section. PME (available at http://www.lia.deis.unibo.it/Research/MIDAS) exploits the reasoning features of the DL-based reasoner Pellet [22] and the framework Jena [18] to acquire and manage ontologies.

4.2. Matching algorithm

This section describes the matching algorithm we have implemented to perform preference-driven semantic selection of services.

As shown in Fig. 5, the algorithm takes an offered capability and a required capability and it returns the degree of semantic compatibility between them. Each capability is characterized by its properties. Let us note that offered capabilities are individuals, i.e., specific instances of a class, whereas requested capabilities are classes defined by restrictions, with restrictions on service properties determined by user-specified value preferences. The algorithm works on one capability at a time. For each required capability, it is able to recognize three possible subsumption relations with the offered capability, namely: the offered capability may be an instance of the requested capability class (case exact), or an instance of a class that subsumes it (case subsumes) or an instance of a class that is subsumed by it (case plug-in) [23]. These semantic relations are determined by performing subsumption reasoning over the property values and class types of offered and required capabilities. In case of exact match for all service capabilities, the offered service is compatible with the user’s request. In case the matching is not exact, compatibility is evaluated depending on value preferences: if there exists a preference allowing to relax the constraint over that property, a plug-in or subsumes case might be considered compatible. In particular, let $V_S$ be the vocabulary describing the service ontology specified in OWL-DL ($SHOIN(D)$). Let $C_S$ be a class in $V_S$ describing a concept capability. Let $C'_S$ be a subclass of $C_S$, i.e., $C_S$ logically subsumes $C'_S$ in the ontology interpretation. Let $C''_S$ be a superconcept of $C_S$, i.e., $C_S$ is logically subsumed by $C''_S$. Let the user requested capability be specified as a set of restrictions on the properties of class $C_S$.

- **Case exact.** The offered capability is an instance of $C_S$. For example, the required capability class is of type Info_Capability, the offered capability is an instance of Info_Capability and the values of its characterizing properties satisfy the restrictions defined in the required capability class.
Case plug-in. The offered capability is an instance of $C_5$. For example, the required capability class is of type Info_Capability and the offered capability is an instance of Newspaper_Capability, which is a subclass of Info_Capability, and the values of its characterizing properties satisfy the restrictions defined in the required capability class. Let us note that, being the offer more specific than the request, it might also happen the case that the offered capability is an instance of a subclass of the required capability (such as the News_Capability with respect to the Info_Capability), but one or more properties constrained in the request by means of value preference specification are not defined in the offer (such as "video broadcasting quality", which is not meaningful for newspapers). In this case, the algorithm might behave differently depending on priority preferences. If the considered property has low priority or is optional, the service might be considered compatible with the user’s request.

Case subsumes. The offered capability is an instance of $C_5$. For example, the required capability class is of type Info_Capability and the offered capability is an instance of a generic service capability, which is the superclass of all service capabilities, and the values of its characterizing properties satisfy the restrictions defined in the required capability class. Let us note that, being the offer more generic than the request, any property constrained in the request will have a value in the offered capability instance.

Priority preferences are exploited to evaluate the degree of relevance of a specific property when determining service compatibility. In particular, priority preferences might be applied to determine the order according to which the algorithm should check compatibility of offered/required capabilities and properties. As a key feature of AIDAS, the prioritization of capabilities and properties allows to evaluate service properties according to the importance they are given in the user’s request, thus enhancing service search with both efficiency and personalized precision. It is also worth noticing that the adoption of value preferences allows AIDAS not only to provide the user with desired service features, but also to adapt the flexibility of matching according to user’s preferences. This ability enables flexible, yet fine-tuned service filtering and provides the user with a personalized view on available services.

5. Case study

We have tested AIDAS in the design and implementation of a News Discovery Assistant (NDA) that enables mobile users to access and retrieve news from information services on a local basis. NDA retrieves information services available in the user’s nearby, providing mobile users with news they might be interested in. NDA can be exploited, for example, to provide tourists with local news and information about ongoing events in the place where they are on vacation. Instead of manually browsing web portals and/or looking for printed information bulletins, tourists can define their personal interests and be automatically forwarded news and information on their device display.

Our setting for NDA consists of a wireless metropolitan network composed by several 802.11 network localities, with each locality modeled as a CARMEN domain. Each domain provides execution environments (places) for sha-
dow proxies on each physical node, offers CARMEN and AIDAS middleware facilities and hosts service components providing information about locally available resources. Resources, entities and services are described by means of semantic metadata. Each place hosts an instance of AIDAS Metadata, Context Manager, and Discovery Manager, whereas the Profile Matching Engine is implemented as a centralized element residing on one predefined place within the domain. The QPE component also resides on a predefined place with PME, while an instance of QPI executes as part of a device-specific application running on the client device. The NDA server-side component executes on each CARMEN domain and relies on the AIDAS framework to provide its functionalities.

The NDA service is accessible from wireless devices through several access points, which may be located, for example, at the harbor in a vacation locality, at the airport, at the railway station, or in the hall of a hotel. NDA users interact with the NDA server-side application via device-specific clients running on their wireless access devices. Client-side applications enable users to subscribe to NDA by filling in a form with user profile and to authenticate themselves to the service before starting any NDA session. When a user first accesses the service, NDA instantiates a shadow proxy in the domain where the user is currently attached. At service provision time, client applications are only in charge of forwarding user requests via QPI (and of visualizing received service results) to (from) their responsible proxies. Whenever a new user makes a request for service, AIDAS middleware initiates a discovery process by interacting with both the user proxy and the Context Manager. In particular, the user proxy forwards a request for service to AIDAS, which performs automated reasoning over service profiles and user request, to find which information services are compatible with the user request. CM is then in charge of providing the user with desired news.

Let us suppose that Alice arrives in Vienna for the 2008 European Soccer Championship and wishes to be provided with information about sport-related events taking place in town or in the local surroundings. Once in her hotel room, Alice accesses the NDA service via the wireless connection provided by the hotel. The interaction flow between Alice and the various middleware components is shown in Fig. 6.

As Alice first connects from the hotel network area, NDA instantiates a shadow proxy within the hotel CARMEN domain. The device-specific client allows Alice to specify her profile and her device’s profile, which will be
retrieved at the beginning of any new NDA session from any CARMEN domain, and to set the desired configuration settings for the discovery process. Based on profile metadata and Alice’s location, the Context Manager determines Alice’s context. The Discovery Manager dynamically determines Alice’s visibility on services based on her context by exploiting PME matching algorithm. For instance, as Alice states in her profile that she can speak English and Italian, DM includes in the service view only services that are provided in one of these languages. In addition, the device profile is used to select services that are technically compatible with Alice palmtop. For example, services not supporting 802.11 are not included in Alice’s view.

After the initialization phase, Alice is enabled to forward to NDA a service request via the QPI interface. In the NDA application, QPI shows service relevant properties and options for news and information services. In addition, QPI takes Alice’s profile into account to reduce the set of displayed properties. In particular, the only possible values displayed for the language property are the ones Alice is able to speak, as stated in her profile. Such filtering of service properties and options based on user profile provides the user with a personalized view on available services, thus making user interaction with the system easier and more manageable, even in case of limited devices and/or attention.

In addition, Alice can express her value preferences. For instance, Alice specifies that she is mostly interested in local news about soccer, but would like to check weather forecasts, too. MM translates these preferences into the corresponding property restrictions (shown in Fig. 7). Alice also agrees to relax her preferences: in case no exactly compatible service is found, AIDAS is allowed to look for services with more generic properties, such as a service providing any kind of sport news. Let us note that Alice is not forced to choose between possible alternative values, but simply accepts to loosen her request to obtain more flexible discovery results. As far as priority preferences are concerned, NDA adopts a simple ranking classification, i.e., a property can be required or optional. In this case, Alice sets the sport preference as mandatory, while the preference on weather forecasts is optional.

The information acquired by QPI is translated by the QPE parser into a service request (shown in Fig. 7), whereas value preferences are translated into the corresponding property restrictions.

This service request is forwarded by QPE to PME, which performs the matching algorithm. Let us consider the case of a local Mobile News Service (MNS) that provides browsing and RSS feeding of local newspapers, i.e., it has the “providing newspapers” capability. Fig. 1 shows an excerpt of MNS static profile. MNS is an English-speaking service providing news about several topics, including sport. However, MNS does not provide news about “soccer”, but, more generally, about sport. In addition, it does not provide information about weather forecasts. Since Alice has agreed to relax her matching requirements and her preference about weather is optional, MNS is compatible with her request. After the direct matching phase, PME matches MNS requirements against Alice capabilities. In particular, the MNS service requires that the client application has a browser (hard requirement) since news are visualized via a browser interface. Since Alice’s device has a browser (see profile), MNS is considered compatible with Alice’s request. PME now passes to CM the identifier for the service and suspends its task. CM automatically retrieves MNS grounding information in the service repository and checks if the service is currently available. If MNS service is currently available, PME terminates the execution of the matching algorithm and CM gives to the user proxy a reference to the service from which Alice will obtain the searched news.

```
<rdf:RDF>
  <query:ServiceRequest rdf:ID="ExampleRequest"> 
    <owl:Class rdf:ID="ExampleCapability_1"> 
      <owl:intersectionOf rdf:parseType="Collection"> 
        <owl:Restriction> 
          <owl:onProperty rdf:resource="Info-ont:InfoCapability"/> 
          <owl:Restriction> 
            <owl:onProperty rdf:resource="Info-ont:hasTopic"/> 
            <owl:someValuesFrom rdf:resource="&d3;Sport"/> 
          </owl:Restriction> 
        </owl:Restriction> 
      </owl:intersectionOf> 
    </owl:Class> 
    <query:preferences> 
      <pref:ValuePreference rdf:ID="ExamplePreference" 
        pref:over rdf:resource="Info-ont:hasTopic"/> 
      <pref:relaxation rdf:resource="&pref:ont;Plugin-Subsumes"/> 
    </query:preferences> 
  </query:ServiceRequest> 
</rdf:RDF>
```

Fig. 7. NDA example user request.
6. Performance evaluation

The exploitation of a context-aware semantic middleware for service discovery, such as AIDAS, introduces different forms of overhead, depending on both the deployment environment and the performance of the different middleware facilities, from profile parsing to semantic-based query resolution.

We have extensively evaluated both the quality of our matching algorithm and the overhead introduced by the adoption of semantic metadata and techniques. We have considered a variable test-bed search space of 30–100 profiled services with requirement/capability ontologies following a hierarchical classification tree. In particular, the test-bed ontology nodes are in subclass relations; the ontology tree depth (maximum degree of requirement/capability specialization) is 4 and its breadth (multiplicity of requirement/capability related concepts) is 3. Each service has either one or two capabilities modeled in our ontology. Each user’s request has a variable number of preferences, from 1 to 4, on a single capability. To evaluate the quality of our matching algorithm, we have measured recall, i.e., the extent to which all relevant registered services are retrieved (by avoiding false negatives), and precision, i.e., the extent to which only relevant services are retrieved (by avoiding false positives) [26]. Being our matching algorithm complete with respect to our service ontology, its recall is optimal. This means that AIDAS is able to find all services whose capabilities have a semantic relation (as defined in Section 4.2) with requested capabilities, according to the service ontology we have designed. Future work will be devoted to the exploration of heuristic-based pruning techniques to improve matching response times by possibly sacrificing completeness.

About precision, we have considered the case of services with two capabilities and user requests with two preferences. We have then compared the number of AIDAS-retrieved services with the service set retrieved by Jini (by exploiting the Jini Technology Starter Kit, version 2.0, and representing service capabilities as Jini attributes). AIDAS has demonstrated to improve Jini precision of roughly 77% in the considered testbed. This outperform is due to the adoption of a service ontology that separately defines service capabilities and requirements. While Jini only allows to define and look for service attributes, AIDAS allows to define either capabilities or requirements with different meanings. Users requirements and preferences are only matched against service requirements, thus reducing false positives.

Finally, we have evaluated AIDAS query response time (between an explicit user request and the determination of discovery results). This test was executed on an AMD Athlon XP 1600 processor, equipped with 256 MB of RAM, running Windows XP Home Edition. PME was implemented using Jena 2 and Pellet 1.3 (on JDK 1.4.2). In our test evaluation, services and middleware components reside on the same node. This test was carried out considering a variable number of preferences on the same capability. With a test-bed search space of 100 services, the response time for a service query varies from approximately 9 ms, for a query defining only one preference, to roughly 12 ms, for a more complex query defining four preferences. The query answering process basically involves four stages: parsing ontologies into a reasoner-compliant format, querying parsed ontologies based on the request, performing reasoning over query results and ordering results depending on the degree of compatibility. We have evaluated the single contributions of each phase to the total response time. Our tests show that the most time-consuming activities are ontology parsing and querying, which are responsible for roughly 55% and 40%, respectively, of total query response time. Reasoning takes a very limited percentage of the total time (about 5%), while ordering time is negligible. It is worth noting that a significant variation on query response time might stem from variable network conditions. However, since such a variation might be difficult to evaluate and control because it depends on external conditions, we do not to consider it in our evaluations.

7. Related work

Service discovery has always represented a crucial activity in the evolution and deployment of distributed systems. Discovery solutions for traditional distributed environments, such as Jini, CORBA or DCOM, typically rely on the assumption of a shared agreement among interacting entities about how to describe and to invoke a service. However, as this assumption cannot be made in pervasive environments, traditional discovery solutions seem inadequate and novel solutions for pervasive scenarios are emerging [10–14]. On the other side, the Web service community has started to relax these assumptions, by describing services using XML formats and retrieving them based on keywords and fixed taxonomies, such as in the case of UDDI and its white-pages and yellow-pages mechanisms. However, little support has been provided to perform service discovery based on service capabilities or user-defined data. In addition, Web Services protocols lack of semantics causes service discovery to be imprecise if the user is not provided in advance with a syntactically defined description of service features.

Authors of [25] developed an augmented version of an UDDI registry that permits to attach user metadata expressed by RDF triples to common service descriptions. This allows to overcome the expressive limitations of the UDDI registry by adding different kinds of metadata, e.g., service ratings, functionality profiles attached to services and semantic types attached to operation arguments. However, since inferencing is separated from discovery in order not to degrade discovery performance, this solution does not actually exploit the semantics of metadata as AIDAS does. In addition, unlike AIDAS, it is bound to a specific discovery protocol.

In recent years, several research efforts have emerged to enhance service discovery and matchmaking, particularly
by means of semantic-based technologies [29,32,34]. In [24], authors suggest that these efforts can be divided into two categories, i.e., special request language and query by example instance. To the former belong discovery solutions that adopt a special purpose query language, such as a SQL-like language for UDDI repositories. These solutions have the advantage of providing the user with several options for preference specification. AIDAS allows preference specification without requiring any additional language but simply exploiting its metadata model to describe both request and offer.

On the other side, “query by example” approaches adopt the same language for both service request and offer, describing the request as an instance of the ideal service. This requires to perform a matchmaking process to evaluate the similarity between the request and the offer instance. A well-known example of this approach is the Semantic Matchmaker developed at Carnegie Mellon University [23]. From CMU Matchmaker we adopt the established semantic relations of exact, subsumes, plug-in. Other relevant pieces of work have extended these basic categories with similarity-based relations between services [31], additional logic-based relations [34] or potential/partial match relations [30], and implemented appropriate algorithms to compute and give ranking scores to compatibility and/or similarity between services. AIDAS is an integration framework that exploits semantic-based matching to provide the user with a personalized view on services. Its key feature is the adoption of semantic metadata to customize the discovery experience based on user’s characteristics and preferences. Therefore, it does not focus on the issue of implementing powerful matching mechanisms, but on the integration of multiple features, such as semantic support configuration and semantic-based discovery, as well as on the central role of the user for the application. Similar considerations apply to our approach to preference handling, which is a potentially complex issue as shown by relevant existing work [38]. Our approach was intentionally kept simple in order to make preference specification a manageable task for the mobile user.

AIDAS relies on its own service ontology, expressed in OWL-DL. We are aware that several languages for service description have been proposed, such as OWL-S [23], WSMO [39] and Meteor-S [37], to model both service interface (input/output) and service process workflow. These languages and the matchmaking algorithms exploiting them are generally more focused on input and output description rather than on service capabilities. Since AIDAS is built to support mobile users in service discovery, our ontology describes service characteristics instead of service inputs and outputs as we believe this is a more intuitive description for a human user. However, we are considering to provide support in AIDAS for other languages.

Authors of [26] propose a classification of various discovery tools based on their precision, i.e., the extent to which the tool retrieves only the items the services is interested in, and on their recall, i.e., the extent to which the tool retrieves all the items the services is interested in. To achieve a good level of both precision and recall, they propose a process-based service model, which aims at capturing service behavior as a collection of sub-activities. Therefore, they define a specific query language and describe process models in terms of entity–relationship diagrams. AIDAS approach differs from the process-based one in that it does not model the service internal activity, i.e., its process model, but only its external interface, according to a declarative, object-oriented model of service. Another relevant difference in the logical approach to service matching is that AIDAS relies on subsumption reasoning and instance classification provided by description logic (using OWL-DL), while this solution executes variable bindings on clauses.

The discovery middleware proposed in [20] is the most similar solution to AIDAS. Both solutions address discovery issues for pervasive environments and both allow for the specification of user’s preferences. In particular, the solution in [20] implements an ontology browser for mobile devices and allows users to express their preference directly over ontology diagrams. Even if the direct manipulation of ontologies might increase efficiency in preference specification, as it requires a lighter computation on the middleware side, we believe it has the disadvantage of leaving to non-experienced users the burden of dealing with ontologies that may not always be clear to understand. In addition, unlike AIDAS, [20] does not provide a complete user profile model and cannot therefore exploit the corresponding information to determine the user’s context.

8. Conclusions and future work

Service discovery is a crucial activity in pervasive environments where mobile users need to be provided with context-aware and location-dependent services. Semantic languages seem to represent a suitable means to realize advanced discovery solutions that overcome the intrinsic limitations of traditional models. We propose a middleware that exploits semantic techniques to perform context-aware discovery of services based on the requirements and preferences expressed by mobile users. The design of the AIDAS middleware tackles the challenge of making semantic-based discovery viable even to resource-constrained devices. To this purpose, AIDAS integrates several middleware facilities capable of adapting semantic support to the different characteristics of mobile devices and of providing mobile devices with visibility on semantic functionalities hosted by nearby devices.

We are actively working on the AIDAS framework along various directions. We are currently implementing an AIDAS version that relies on a relational database implementation of Jena using a common repository to store OWL user/device/service ontologies. We are also developing a visual tool for the specification of user and service profiles, which will be integrated within the MM.
We are also aware that in open and dynamic scenarios it is necessary to deal with security issues. Therefore, we are enhancing the AIDAS framework with security features, e.g., authentication and access control features.

As future work, we plan to implement a new version of DM based on a network service discovery protocol instead of a registry, such as the Bluetooth discovery protocols or a P2P specific discovery protocol.

Another interesting future issue we envision to deal with is the resolution of conflicts that may arise between value or priority preferences. We believe that a possible approach may be the definition of meta-preferences, in a similar fashion as meta-policies are defined for policy conflict resolution [16].

Acknowledgements

This work was supported by the MIUR FIRB TOCAI, CNR Strategic IS-MANET and MIUR PRIN MOMA Projects.

References


