Towards Autonomous Context-Aware Services for Smart Mobile Devices

Thomas Strang

German Aerospace Center (DLR)
Institute for Communications and Navigation
D-82230 Wessling/Oberpfaffenhofen, Germany
thomas.strang@dlr.de

Abstract. In this paper a framework is presented which allows the discovery and execution of services on connected and partially autonomous mobile devices. Discovery and execution procedures are sensitive to the user's context (current location, personal preferences, current network situation etc.). We present a description language for service offers which is used to provide the necessary information for a service registry running on the mobile device itself. Services are executed in an abstract manner (in the sense of a non-specific implementation) from the user's point of view, getting an optimal result with respect to the current context out of a set of parallel invoked service implementations.

1 Introduction

In the past few years mobile computing has become very popular, the penetration of mobile devices like mobile phones or PDAs is growing fast. By using technologies implemented today, mobile devices enable the user to access Web-based data from nearly any place in the world when online by utilizing a Web/WAP-browser in their device. In contrast, when offline (e.g. in a plane, where the usage of online connections is prohibited), the user is unable to find new interesting services and restricted to execute only a small set of offline-applications like a calculator. It is a challenge to design an architecture which provides the user of a mobile device with personalized, situation-related services for discovery and execution, online and offline in a best effort sense.

Upcoming mobile devices are capable of using multiple access networks, and are partially programmable for third parties.The architecture presented in this paper is designed with such new features in mind. Even if the architecture does not restrict the mobile device to be a client only, this paper does not further investigate scenarios, where the mobile device is the server for clients in the wired part of the network.

In section 2 of this paper we describe our definition of the service terminology subsequently used. We introduce our architecture used to manage mobile services. A new kind of service registry, residing on the mobile device itself, enables autonomous service discovery and service execution in times of bad or no network coverage. The concept of context sensors, presented in section 3, shows how to integrate context-awareness during discovery and execution. The service offer description language proposed in section 4 will be used to create context-dependent service offers, and also provide a way to exchange mobile services for partial autonomous execution on the mobile device. A new type of abstract service invocation is discussed next (section 5), which delivers an optimal result with respect to the current situation. Finally, we discuss some implementation aspects (section 6), and draw our conclusion (section 7).

2 Services in Distributed Systems

Within the context of computer networks, the terminology of a "service" is used do describe a wide variety of different things. Thus, to avoid misinterpretation of what a service is and how they are used, a detailed specification is required.

We define a service as a *namable* entity being *responsible* for providing information or performing actions with specific *characteristics*. We distinguish between *MicroServices* and *MacroServices*. MicroServices encapsulate "atomic" operations, meaning that no MicroService engages any other MicroService. In contrast, MacroServices engage at least one other MicroService to perform its operation.

The entry point of a service, be it a MicroService or MacroService, is represented by an addressable instance in a sense of WSDL's "service ports" [4]. The entity representing a service needs to be namable to identify its operation in relation to a given namespace to distinguish services across different namespaces. Each service is engaged through its addressable instance, which is instantiated and controlled by a service provider. The service provider is required to manage the lifecycle (start/stop, pause/resume, etc.) of a service instance and to coordinate service interaction (input/output, invocation/result delivery, etc.) by adopting to a common protocol like JINI, CORBA, DCOM or SOAP for information exchange in distributed environments.

For the time of the duration of a service's operation, the service providing entity bears the responsibility to act in a sense the calling entity envisioned. This is modeled by a caretaker entity, which resides at the service provider and acts e.g. as a life-cycle-manager as well as a representative of the calling entity when the execution of the service is asynchronous. Thus, a service provider in our definition has similarities to a platform for software agents [8], and the caretaker entity can be seen as an extended life-cycle-manager like a *dock* [5].

2.1 Service Discovery and Interaction on Mobile Devices

In a distributed environment, a service has to be found by a client entity (human and/or program) before it can be used. Therefore, a potential service client contacts any form of service registry (e.g. UDDI registry, Bluetooth SDP Database, JINI Lookup-Service or Corba Trader), and requests the availability of providers which have been registered with the service registry to offer services with specific characteristics. These characteristics are usually classifications of a common ontology well known to both parties. An overview of typical protocols in use for service discovery can be found in [2] or [3].

Most architectures for distributed systems silently imply the existence of a reliable, wired network which interconnects all hosts involved in distributed operations. In the wireless world, important additional aspects like limited bandwidth, cost of bandwidth, sometimes unavailable network connections etc. have to be taken into account. Whereas the network layers are usually designed to handle these differences to the wired world very well, the consideration of them in the service layers is often totally insufficient.

In fast-changing contexts [6, 17] such approaches are not sufficient. Thus, to handle at least some of these dynamic context issues and to enable the most possible seamless operation of services on devices with occasionally unavailable network connection of any type and intermittent failures, we suppose an architecture element shown in fig. 1, where an instance of a service registry is running on a mobile device. In [11, 19] the authors showed the several advantages of having a service registry instance on a mobile device, whether in short range or wide range networks. This makes our approach different from other mobile service architectures like the *Capeus* [16], the *Centaurus* [10] or the *Ninja* [9] system, where the service registry (sometimes also named *Trader* or *Broker*) respectively the offer-demand-matcher resides in the wired part of the network.

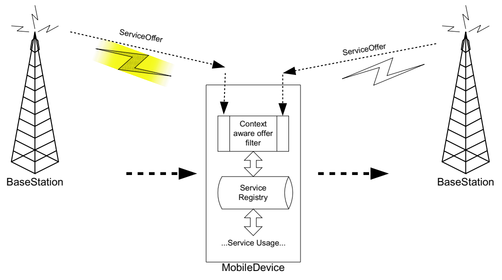

Fig. 1. Service Registry on the Mobile Device

Mobile devices like mobile phones or personal digital assistants (PDAs) are predominantly limited devices with respect to computational power and both volatile and persistent storage (memory). These limitations make it impossible to follow "heavyweight" approaches like CORBA or JINI for distributed service discovery and distributed service usage, even if they have attractive solutions in some areas. One architecture [21] tries to handle dynamic environments by *virtual services* which are composed using some flow language and bound to some physical services at runtime to reflect reconfigurations of the current situation. The scenario behind the architecture of [21] is not mobile device centralized, wherefore the proposed protocols and other architecture elements are not designed to be used on usually very resource-limited mobile devices. The MOCA architecture [1], whose design has the most similarities to ours, claims to be a service framework for small mobile computing devices. But having the service registry on the device is not sufficient. So is MOCA's invocation model for remote services based on dynamically downloaded bytecode, RMI or IIOP and custom Java classloaders, all together not available on typical wireless information devices. As the authors state in [15], MOCA addresses mainly back-end application logic, and a typical MOCA application is either implementing "traditional Java technologies such as the AWT" or "on the use of servlets that are hosted under MOCA", which are both too heavyweight for small mobile devices. The aspect of context-awareness (see section 3) is not handled within MOCA.

Instead of this, we propose an architecture in the following sections based on *Web Services*, a technique incorporating HTTP as the default transport protocol and XML as the representation format for data of any type. XML offers a platform-neutral view of data and allows hierarchical relationships to be described in a natural way. Compared to other architecture models, HTTP and XML are less resource consuming and, therefore, supported by upcoming smart mobile devices (see section 6).

The discovery of web services is usually performed by contacting a UDDI registry located somewhere in the network, but typically not on the same host as the client performing the search query. As explained, to be able to answer search queries even in times where no network connection is available, it is necessary to have a service registry running on the mobile device itself. Just running an UDDI registry on a mobile device is insufficient for several reasons, mainly because of its missing volume control features and its context-unawareness. Instead, we propose an *Advanced Mobile Service Registry* (AMSR) according to fig. 2 running on the mobile device.

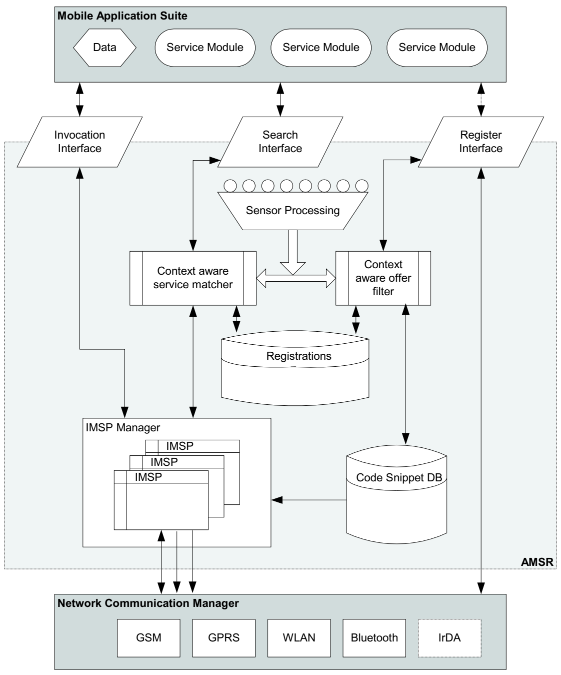

Fig. 2. Advanced Mobile Service Registry (AMSR)

The design of this architecture leans towards a CORBA object trader, e.g. allowing one to apply constraints, as well as a JINI lookup server by enabling code deposition at the registry, but also enables service discovery and execution on mobile devices even in cases of poor or no network connection. The adaptation to the various different types of (usually wireless) network interfaces is done by the *Network Communication Manager*. This module handles network specific issues like adressing in ad-hoc networks or interfacing to SDPoverIP when peering to Bluetooth devices.

Service offers (see section 4) are announced to the AMSR from the network (or the mobile application suite itself, if it also implements corresponding services) through a *Register Interface*, which routes such offers to a context aware offer filter (see section 3). Any search for a service with specific characteristics is performed using the *Search Interface*, and the demand is tried to match against any entry in the registration, where all the offers reside which passed the offer filter constraints. If the search is successful, an instance of a generic *Intermediate Service Provider* (IMSP) is created, which can be accessed using the *Invocation Interface* to perform an *Abstract Service Invocation* (see section 5).

3 Context-Awareness through Context Sensors

The effectiveness of discovery is heavily affected by the quality of information mutually exchanged to describe the requirements on and the capabilities of a service. In a human conversation, the listener uses context information as an implicitly given additional information channel to increase the conversational bandwidth. By enabling access to context information during but not restricted to service discovery, we improve the amount of transinformation from service provider to service user (human or application) and thus the quality of offer-demand-matching. Changes in context during service execution may directly affect a service's output.

We rely on the term context as defined in [6], which is "any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant [..]". Context information is obtained by *observing* relevant entities with *context sensors* (CS). Each CS is responsible for acquiring a certain type of context information and encapsulates how the information is actually sensed. It represents the state of a specific context type against user- and system-services.

Even if most publications like [14] rely on the current geographical position of a mobile device as the most important context information, there is more to context than location [17]. A substantial amount of entities may be observed by CS which base on hardware or pure software sensors. Most of the characteristics of a CS (high degree of individuality, adaptation-ability and autonomy) are also known as the main characteristics of software agents [8]. Thus part of our work is to analyze the requirements of an agent platform on mobile devices [20], and how a CS may be implemented following the agent paradigm. We assume agent-based CS to be a much more flexible concept to perform context sensing and processing than for instance the concept of *Cues* [18].

The AMSR architecture introduced in the last section enables access to the context during service discovery and service execution by using a bank of context sensors, thus our architecture is *context-aware* regarding the definition of context-awareness given in [6].

4 Service Offer Descriptions

A lot of research has been done on service discovery and associated protocols. They all can be used to determine interoperability like supported interfaces or valid value ranges. Only some of these protocols try to consider the context of the requesting entity (application or user). This is of key interest in mobile usage scenarios, where the current context has a strong impact to the needs of a mobile user. Different aspects of context like the position, time, personal preferences or costs should be considered in order to discover a service which matches the user's needs best.

As an instrument for describing a service demand containing context dependent constraints one can use *demand descriptions* like Content Aware Packets (CAPs) [16]. CAPs may be "used as communication units between service requester and service provider", and are predominantly designed to "describe a demand from the consumer perspective" [13].

Even less research has been done in context-dependent service *offer descriptions*. Service availability may be restricted to certain places or times. The result quality of a service may depend on network conditions, type of contract with the provider, availability of sub-services and so on. A concrete example is a printer which announces by an offer description to be available for print jobs in principle, but if a context sensor indicates a "toner low" state, this offer is a "non-preferable" one.

Any demand description must be analyzed and matched against the service offers known to the AMSR. To become a service candidate, the service provider or a representative entity called *registrar* is required to announce the service's availability, it's requirements and capabilities to the AMSR. For this purpose we propose the usage of offer descriptions called "Rich Service Offer" (RSO) according to fig. 3.

Offer Description
Software Interface Signature
List of Addressable Instances
Property Value Ranges (optional)
Dependency Declarations (optional)
Cost Indications (optional)
Code Snippets (optional)

Fig. 3. Rich Service Offer

It is important to notice that each MicroService/MacroService must be announced by at least one RSO to ensure its "visibility" to the AMSR.

4.1 Commit on Software Interfaces and Addressable Instance

During service discovery the first thing all participating entities have to commit on are the signatures of the software interfaces in use (*signature interoperability*). Thus a primary element of any RSO is a software interface signature, expressed as a WSDL *port type* identifier [4]. WSDL port types together with a namespace identifier are perfectly suitable to express in an open and extendable way, how to invoke a service, which input parameters are required and which output parameters can be expected.

If the service instance applying to be a candidate for being discovered is running somewhere in the network, but not on the mobile device (standard case for web services), this is the location identifier of the running instance, e.g. the content of the <soap:address ...> tag of the port definition of a WSDL document [4]. If the issuer of the RSO knows about the existence of an AMSR-local service implementation (e.g. by sending a code snippet along with the RSO), this may also be a location identifier pointing to a local implementation.

4.2 Classification Ontology

Beneath information about implemented interfaces and addressable interfaces, it is essential to characterize the offered service with properties any interested entity is able to evaluate (*semantic interoperability*), meaning those properties share a code space known to both parties (often called *Ontology*). Each property is expressed as a valid code out of a common taxonomy or identifier system. A taxonomy is defined as a set of categories, whereas a identifier system is defined as a set of identifiers, according to

$$taxonomy_A = \{cat_1, cat_2, cat_3, ..cat_n\} \tag{1}$$
$$identifiersystem_B = \{id_1, id_2, id_3, ..id_m\} \tag{2}$$

Samples for (1) are industry codes like NAICS, product and service classifications like UNSPSC or geographic location codes like ISO 3166. Samples for (2) are unique identifiers like Dun & Bradstreet D-U-N-S or phone numbers.

The most simple implementation of properties are key-value pairs, attached to an offer. A typical extension is to group a set of key-value pairs together, building a *template* of parameters (see UDDI `tModels` or JINI LUS `entry` attributes). Another extension are *dynamic* properties, where the value of the property is not held within the service registry itself, but is obtained on-demand from an entity nominated by the registrar (see CORBA trading object service or Bluetooth SDP). Known variations of key-value properties are models which take into account the hierarchical relationship of typical classifications. A sample for this is the CORBA service object `ServiceType` model or the Bluetooth SDP `ServiceClassIDList`.

If two or more service offers are evaluated by their properties to be a service candidate, this means those services fulfill the minimum requirements regarding technical (e.g. implemented software interfaces) and quality level (e.g. minute-precision departure times) constraints from the client's perspective.

The minimum requirements itself may be expressed as threshold values for each evaluated property. Book [12] gives samples how to classify different service characteristics, how to project general deterministic, probabilistic and statistical properties to a common metrical base, and how to calculate the distance between any combination of $\{request(property_i), offer_j(property_i)\}$ pairs, which allows one to specify an overall graduation of service offers with respect to a specific service request. This book gives also a good overview of different metric models with the interesting observation that a complex metric model is not necessarily a guarantee for the best result.

4.3 Costs of Usage

An important part of any service offer is the indication of the costs of usage. [7] postulates any service to have a nominal price, which is defined as a charge for the service being provided. We would refine this requirement by a distinction between the price indication for service usage (*cost of usage*) and a price indication for the content (*cost of content*). A sample for the first one is a reservation-processing fee, and the latter one is a ticket price for the same train ticket reservation service.

For any non free-of-charge service, the service offer must contain information about all applicable payment systems (e.g. eCash, credit card), payment channels (e.g. network, phone) and beneficiary, which is the entity to which the payment is addressed. It should be noted that we concentrate here on the cost of usage. The cost of content may base on the same debit system, but is negotiated on the service level.

Most payment systems require online validation of any account data (e.g. remaining credit units on a prepaid account, or credit-standing of a visa account). Service implementations described by offers containing cost indications for these payment systems thus require an established online connection at the time of validation, which is usually at the time of execution of the service, and hence these services cannot be executed offline.

For MacroServices cost indications are accumulative, meaning that cost indications of the composed MicroServices are merged. The service user typically requires a valid account at least at one payment system announced in the offer of any engaged MicroService.

4.4 Dependency Declarations

It is important to notice that in cases where services are offered which internally make use of other services - “visible” to the AMSR or not - this dependency has to be indicated to the AMSR. For instance if a service announced by the current RSO is able to perform its task without any network connection, but makes use of other services via the AMSR, those other services may require a network connection.

If those sub-services indicate to be non-free in the cost-sense, any budget which has been granted during service discovery for the service represented by the current RSO must be splitted and shared with all non-free sub-services.

4.5 Code Snippet Deposition

In cases where a service implementation does not require an existing online connection to perform its operation (e.g. a credit card number validation service), a code fragment implementing the operation can be deposited at the AMSR for usage. These snippets must follow some common code exchange format, like interpretable code (e.g. Java Bytecode, JavaScript) or abstract rendering directives (e.g. XML/XSLT files defining a GUI as a service front-end [15]) Note that it must be ensured that a proper execution handler for the code snippet exists on the mobile device (e.g. Java VM, Scripting handler or Renderer). One may apply an additional protocol to negotiate a proper format. In our case this information can be obtained from the `User-Agent` HTTP header field.

In principle and in comparison to JINI full service proxies, code snippets may, but need not be, a marshaled form of an already instantiated object. The AMSR is required to instantiate the local implementation if a service demand matches with this offer and the offer is not marked as a static service. Some devices may have a Virtual Machine which does for security reasons not allow the execution of bytecode which has not been present at the time of installation of the calling application (*closed late binding*) [20]. In this case, the usage of code snippets is limited to non-Java scripting.

4.6 XML encoding of RSOs

RSOs make use of XML respectively XML schemas as its canonical base to describe service offers with the characteristics shown in the previous sections because of XML's platform and application independent, structure preserving and extensible capabilities. It is reputable to use XML as a base for service or content descriptions, as one can compare with e.g. CAPs [16] or Centaurus' CCML [10]. The example in fig. 4 may illustrate the content of a RSO.

```
<?xml version="1.0" encoding="UTF-8"?>
<rso xmlns="urn:schema-RSO.xsl">
  <interface name="urn:position-service#getPosition" portType="PositionPortType"
    wsdl="http://foo.com/wsdl/position.wsdl"/>
  <instance port="http://foo.com/soap/servlet/rpcrouter"/>
    <property codesys="DUNS" code="12-345-6789" title="WGS84">
    <restriction class="region" type="country">DE</restriction>
    <precision min="1m" max="30km" shape="sector"/>
  </property>
  <dependency interface="urn:math#gaussKruegerToWGS84"/>
  <cost type="perRequest">
    <option system="LHMilesAndMore" beneficiary="foo.com" price="3" unit="miles"/>
    <option system="Visa" beneficiary="foo.com" price="5" unit="cents"/>
  </cost>
</rso>
```

Fig. 4. RSO sample

5 Abstract Service Invocation

Traditional mediators like the CORBA trading service mediates services in a distributed environment either according to the operation *select* or the operation *search*. The first one selects one service out of the list of offers which meets the requirements best, the latter one list all service offers meeting the minimum requirements to the requesting entity for further decisions (which is in fact a delegation of a selection decision). The mediator itself is not involved in the service invocation procedure after either operation has been performed.

5.1 A new operation: Spread & Join

We would like to introduce a third mediator operation which we called *spread&join*. This operation engages a group up to all services (best n) meeting the minimum requirements of a given request. The services are executed in parallel (*spreaded*, see fig. 5), and after a set of conditions (e.g. quality threshold passed) are reached, all results are collected and reworked (*joined*, see fig. 5) in a result object, to which the service requestor is holding a reference.

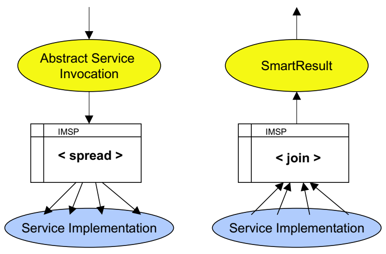

Fig. 5. Spread and Join operations

In this constellation the mediator becomes an active part of the service invocation procedure as well: It acts as a single service provider against the service requesting entity, and as a service requestor against any addressable instance. Thus, we call this entity a generic *intermediate service provider (IMSP)*. Inspired by the concept of virtual services in [21], each IMSP instance can be seen as an *abstract service* invoked by the client. At least one implementation of the abstract service is contributed by the service implementation registered for the corresponding matching offer.

Spreading a service operation over a group of n service providers according to their service offers means invocation of the same operation on n addressable instances in parallel, which leads to $m \leq n$ results reported to the IMSP. Spreading a service operation requires a budget splitting according to a distribution reasonable to the IMSP, if the operation has been restricted by a budget granted to the IMSP.

One problem is the increased the amount of exchanged messages caused by this kind of operation, especially when using expensive wireless network links.

Thus an interaction between the IMSP and the Network Communication Manager shown in the AMSR (see fig. 2) is required to handle the trade-off between minimizing the costs of network connections and maximizing the gain of joining multiple results. Additionally it makes sense to hierarchically cascade the IMSP facilities over the time-variant network infrastructure, which enables the spreading of an operation partly in the wired part of the network, which is much less expensive. Cascading AMSRs and its integrated IMSP facilities guarantees a store-and-forward functionality optimal for a variable network situation. Although any results are optimal in the local context of the reporting node, they are re-evaluated after reported to the next level of the hierarchy. Any AMSR/IMSP in the hierarchy tree is responsible to act in a sense as the calling entity envisioned, e.g. not exceeding any budget granted for a specific operation [5]: If necessary, budgets have to be splitted between service providers (cost sharing).

For any service invocation request received from a client the IMSP Manager manages an instance of a control structure named *SmartResult object* containing the elements *RequestProperties*, *ListOfAddressableInstances*, *Budgets*, *CurrentBestResult* and *AllResults*. The control structure is updated when any new information is available with relevance to the particular operation. This may be the receipt of a new result, or a new context situation (e.g. transition from connected to not connected), or something else. So the spread&join operation relates very much to context-aware service discovery and execution.

An interesting option is enabled by the AMSR/IMSP's autonomy in combination with object references. If the situation, which has been the base for distance vector calculations, changes (e.g. the user moves into a new spatial context or a network link is broken), the metric for any related SmartResult object has to be re-calculated. And even if a result has already been reported to the calling entity, a "better" result stated by the re-calculated metric can be signaled to the calling entity if this entity is still interested. This affects result updates as well as result revocations.

5.2 Selecting or Merging

If more than one result from a spreaded service operation is available, it is the IMSP's task to join them into a single result for the client. Figure 6 gives an overview of the available options.

If the return value expected by the client is a single primitive data type, the IMSP must extract one result out of the received results. If the return value expected by the client is a single object(-reference), the task of the IMSP is the same, extended by the ability to adopt values from a modified result object upon following sub-results with a postponed distributed object access. If the return value expected by the client is an array of primitive data types, it may be desirable to merge the sub-results, or to select a sub-result along certain criteria (e.g. longest array). If the return value expected by the client is a collection object like the Java Vector, the facilities of merging or selecting sub-results and distributed object access are combined.

client expects	**join options**
single primitive data type (e.g. `integer`)	select one sub-result
single object data type (e.g. `java.lang.String`)	select one sub-result (may be updated when reference given)
array of primitive data type (e.g. `integer[]`) or array of objects (e.g. `java.lang.Boolean[]`)	merge sub-arrays *or* select one sub-array
collection object of primitive data type (e.g. `java.lang.Vector` containing `integer` elements) or collection object of object data type (e.g. `java.lang.Vector` containing `java.lang.String` elements)	merge sub-collections *or* select one sub-collection (may be updated when reference given)

Fig. 6. IMSP join options

Different strategies may be followed to perform this join, depending on the termination condition specified by the client as well as the type of return value and the desired quality. If the client specified to be interested in any result in a given period of time (e.g. the first result the IMSP receives within the next 2 seconds), the join strategy is pretty simple. But much more often a client claims results according to certain quality parameters. This requires the IMSP to re-order all sub-results according to a given metric.

Any parameter which is required during the calculation of distance vectors (which is the base operation for any metric) must be given by the client together with the call itself, either implicit (e.g. input parameter of a function) or explicit (e.g. vector distance procedure identifier or timeout value). Because the quality of the operation which a service performs may differ from the quality level promised during registration, additional taxonomy or identifier system codes attached to the sub-results may help the IMSP to adapt better to the calling entity's desired result.

6 Implementation prototype

A prototype of the AMSR is currently under development, based on Java2 Micro Edition (J2ME), whose Virtual Machine (CVM/KVM) is designed to run on resource constraint devices from SetTopBoxes down to mobile phones or PDA's. Devices with the most restrictive limitations are covered by the *Mobile Information Device Profile* (MIDP) on top of the *Connected Limited Device Configuration* (CLDC), which define the subset of Java classlibraries a CLDC/MIDP compliant device must support.

To not overstrain the computational power, network and memory resources of small mobile devices running CLDC/MIDP, a lot of functionality known from full Java (Standard or even Enterprise Editions) have been omitted. For instance the late binding concept of Java has been restricted to be able only to load classes at runtime, which have been downloaded and installed at the same time as the application itself (*closed late binding*). Thus it is impossible to use a service on a mobile device, whose implementation is loaded from the network at runtime, which is the standard procedure e.g. in a JINI network. But even if full late binding would be possible on CLDC/MIDP compliant devices, JINI like approaches would not work because of the lack of standard network access via sockets: The only available protocol is HTTP in client mode. Additional profiles may contribute additional functionality, but devices targeted by J2ME will always be more restricted than devices in the fixed wired world.

Hence, our AMSR implementation considers this situation and is built using an XML parser on top of HTTP as transfer protocol. An XML parser is a valuable application type for Java equipped mobile devices, and there exist several different implementations (kXML, NanoXML etc.). As described in section 1, our approach employs the web service technology of SOAP and WSDL, both based on XML and HTTP, and the footprint of the current prototype (still missing the Network Communication Manager block) is about 75 KB.

7 Summary, Conclusions and Further Work

We have presented a new concept for service discovery and execution on resource constraint programmable mobile devices. Discovery and execution of services is provided if connected, but also if not connected to a backbone system. Discovery and execution procedures are sensitive to context changes. This is achieved by running a service registry called AMSR on the mobile device itself, which is updated with service announcements called RSO. Multiple service providers are, if available, employed to fulfill a service request concurrently. This parallel execution of services is hidden to the user by the abstract service interface, which is invoked by the client, and delivers the optimal result from the AMSR after execution.

First results with the prototype implementation showed that the concept is valuable and outputs the desired results.

Further work has to be done in the area of RSO schema definition. Several strategies to prioritize service offerings have to be worked out and compared to decide which fits best to the limits in storage space and processing power of the target devices. Scalability and inter-system tests have to be performed, and a wider range of sample services must show if all aspects are covered and reasonable. Further investigation has to be done if results from agent systems can be applied to our system (e.g. when comparing spread&join with an agent federation architecture), and if approaches for automated service chaining can be utilized in our approach.

References

[1] Beck, J., Gefflaut, A., and Islam, N. MOCA: A service framework for mobile computing devices. In *Proceedings of the ACM International Workshop on Data Engineering for Wireless and Mobile Access, August 20, 1999, Seattle, WA, USA* (1999), ACM, pp. 62–68.

[2] Bettstetter, C., and Renner, C. A comparison of service discovery protocols and implementation of the service location protocol. In *In Proceedings of Sixth*

EUNICE Open European Summer School - EUNICE 2000 (Twente, Netherlands, September 2000).

[3] Chakraborty, D., and Chen, H. Service discovery in the future for mobile commerce. http://www.cs.umbc.edu/∼dchakr1/papers/mcommerce.html, 2000.

[4] Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl, 2001.

[5] Dalmeijer, M., Hammer, D., and Aerts, A. Mobile software agents. http://wwwis.win.tue.nl/∼wsinatma/Agents/MSA.ps, 1997.

[6] Dey, A. K. Understanding and using context. *Personal and Ubiquitous Computing, Special issue on Situated Interaction and Ubiquitous Computing 5*, 1 (2001).

[7] Dumas, M., O'Sullivan, J., Heravizadeh, M., Edmond, D., and ter Hofstede, A. Towards a semantic framework for service description. http://sky.fit.qut.edu.au/∼dumas/publications.html, April 2001.

[8] Franklin, S., and Graesser, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Proceedings of the Third International Workshop on Agent Theories, Architectures, and Languages* (1996), Springer-Verlag.

[9] Gribble, S. D., Welsh, M., von Behren, R., Brewer, E. A., Culler, D. E., Borisov, N., Czerwinski, S. E., Gummadi, R., Hill, J. R., Joseph, A. D., Katz, R. H., Mao, Z. M., Ross, S., and Zhao, B. Y. The ninja architecture for robust internet-scale systems and services. *Computer Networks, Special Issue on Pervasive Computing 35*, 4 (March 2001), 473–497.

[10] Kagal, L., Korolev, V., Chen, H., Joshi, A., and Finin, T. Project centaurus: A framework for indoor mobile services. http://www.cs.umbc.edu/∼finin/papers/centaurus/.

[11] Kammann, J., Strang, T., and Wendlandt, K. Mobile services over short range communication. In *Workshop Commercial Radio Sensors and Communication Techniques - CRSCT 2001* (Linz/Austria, August 2001).

[12] Linnhoff-Popien, C. *CORBA - Communications and Management.* Springer, September 1998.

[13] Michahelles, F. Designing an architecture for context-aware service selection and execution. Master's thesis, University of Munich, 2001.

[14] Nord, J., Synnes, K., and Parnes, P. An architecture for location aware applications. In *Proceedings of the Hawai'i International Conference on System Sciences, Big Island, Hawaii* (January 2002), IEEE.

[15] Romn, M., Beck, J., and Gefflaut, A. A device-independent representation for services.

[16] Samulowitz, M., Michahelles, F., and Linnhoff-Popien, C. Capeus: An architecture for context-aware selection and execution of services. In *New developments in distributed applications and interoperable systems* (Krakow, Poland, September 17-19 2001), Kluwer Academic Publishers, pp. 23–39.

[17] Schmidt, A., Beigl, M., and Gellersen, H.-W. There is more to context than location. *Computers and Graphics 23*, 6 (1999), 893–901.

[18] Schmidt, A., and Laerhoven, K. V. How to build smart appliances. *IEEE Personal Communications* (August 2001).

[19] Steingass, A., Angermann, M., and Robertson, P. Integration of navigation and communication services for personal travel assistance using a jini and java based architecture. In *Proc. GNSS '99* (Genova, Italy, October 1999).

[20] Strang, T., and Meyer, M. Agent-environment for small mobile devices. In *Proceedings of the 9th HP OpenView University Workshop (HPOVUA)* (June 2002), HP.

[21] Wang, Z., and Garlan, D. Task-driven computing, May 2000.