Human Interactions in Dynamic Environments through Mobile Web Services

Daniel Schall, Robert Gombotz, Christoph Dorn, Schahram Dustdar
Distributed Systems Group, Institute of Information Systems,
Vienna University of Technology, Argentinierstrasse 8, 1040 Wien, Austria
{schall, gombotz, dorn, dustdar}@infosys.tuwien.ac.at

Abstract

In this paper we present the concept of Activity-Centric Collaboration Using Service-Oriented Architectures (ACCUSO), which addresses the requirements arising from ad-hoc collaboration in mobile teams. In ACCUSO, activities are used to map human actions to Web Services exploiting the potential benefits of SOA, such as service discovery and binding at run-time. The possibility to compose activities hierarchically from sub-activities and to redesign running activities provides the process-flexibility required in ad-hoc collaboration. We expand the notion of service orientation by introducing Human-provided Services (HpS) which provide functionality not realizable through software services. HpS are "implemented" by human actors (possibly being mobile), which remains transparent to the system, thereby allowing for the provisioning of HpS based on conventional WS-infrastructure. The feasibility and applicability of ACCUSO is demonstrated through a proof-of-concept implementation.

1. Introduction

Our research group currently focuses on developing novel technologies addressing the requirements arising in modern collaborative working environments (CWE). Our work is driven by the following assumptions on collaboration and team forms:

In the past collaboration took place in *stable teams,* which operated within static organizational structures and which would typically execute rather stable workflows, or processes. An example for a stable team is a department of a company.

Emerging team forms. In modern CWE, we have identified three emerging team forms, each highlighting specific developments witnessed in human collaboration. First, *Nimble Teams (N-teams)*, or short-lived teams, are typically assembled in order to tackle uncommon or unexpected issues. Usually, limited or no a-priori knowledge is available about the processes to be executed to solve such problems or to reach the intended goals. Hence, the collaboration is said to be ad-hoc. Such teams are best exemplified through expert groups or task forces, e.g., during political crisis or in an emergency scenario. Second, *Virtual Teams (V-teams)*, or project teams, are formed across organizational boundaries to reach a clearly defined goal. They usually operate following a fairly detailed project plan which defines responsibilities and time frames for sub-goals (milestones) and the overall project. Finally, *Mobile Teams (M-teams)* highlight the increasing mobility of participants which often has implications in terms of limited device capabilities or interruptions in the availability and reachability of team members. In the following, we use the term "N/V/M" to denote these emerging team forms.

1.1. A service-oriented architecture for ad-hoc collaboration

Addressing the requirements of N/V/M teams we propose to organize work and collaboration based on activities implemented through Web Services (WS). We term the approach Activity-Centric Collaboration Using Service-Oriented Architecture (ACCUSO). In a nutshell, a set of basic services is used to implement the features required from collaborative software. These features are presented to the user as activities which he can execute. The use of activities is beneficial to both users and developers of collaboration systems. For the user, activities provide an intuitive method for organizing work and for team coordination. For system developers, activities are a flexible and reusable construct to implement sophisticated functionality through composition of existing components, i.e., services.

2. Activity-centric collaboration using SOA

In this section we discuss the cornerstones of ACCUSO and present the key components of the system.

2.1. Services for collaboration

The foundation of our system is a fully implemented service-oriented architecture, including service registry, service lookup, and a number of Web Services [2]. These WS implement core features required in collaboration, regardless of team form, task, or domain. These are

2007 IEEE International Conference on Web Services (ICWS 2007)
0-7695-2924-0/07 $25.00 © 2007 IEEE

instant messaging, document management, and a calendar service, to name but a few. Multiple implementations of each service may exist, potentially provided and maintained by different service vendors. All services must be published in a service registry in order to be available to the team at runtime. To reduce the complexity of service discovery, we assume to have knowledge of services' categorizations. We use the following terminology. A *service family* groups services of a given functionality, e.g., "Instant Messaging" or "Document Management". *Service classes* are specific implementations within a service family, e.g., "MyMessenger" by company A or "Doc-Man" by company B. Within a service class, multiple *service instances* may exist, each uniquely identified by a service endpoint. By requiring information regarding service categories in registries we abstract from the problem of determining the semantics of available services which is not the focus of our research.

2.2. Human-provided services

A defining characteristic of ACCUSO is the concept of *Human*-provided *Services* (HpS). HpS allow human actors to provide their abilities and work force to the team through standard interfaces. For example, a user might provide his expertise in a given field to the team through a "Consulting Service", a "Document Review Service", or a "Translation Service". We believe that the use of HpS may be beneficial in two ways: Firstly, they dramatically increase the number of services potentially available in service-oriented infrastructures. While we strongly advocate the service-oriented paradigm, we acknowledge its current limitations when services are needed which cannot be implemented through software (alone). HpS can help overcome these limitations. Secondly, they allow for the abstraction from the human individual (when suitable) and for the application of automatic and standardized techniques for publishing (service registry) and discovering (service lookup) certain services. We argue that such abstraction from the human actor may help to optimize collaborative processes by decreasing the amount of time spent on team coordination and communication.

2.3. Activities

Activities are the key element of ACCUSO. We define an activity as "a set of one or more *related* action items *performed* by one or more human actors and *as perceived by* humans". This definition is best exemplified by an entry in a person's to-do list or calendar e.g., "Reply to Amanda's email" or "Identify suitable partners for joint-venture proposed in Doc. 67-3". In daily (human) practice, the term often serves as a "container" for an arbitrary number or type of action items, or sub-activities.

ACCUSO is based on the proposition that "everything people do" (in CWE) can be viewed as an activity that is performed using resources and tools. Analytically, such activities can be decomposed recursively into sub-activities until an "atomic" level is reached. These atoms of activities are *action items* [4] which are performed using tools. We apply this view as follows:

The features provided by collaborative software should reflect the activities users need to perform. These activities can be modeled as compositions of sub-activities. The atomic building blocks of *complex activities* are *basic activities* which are performed using (Web) services (see Figure 1, 1).

2.3.1. Activity modeling. Activities are declared in XML-based activity descriptors stored in an Activity Management Service (AM service). Activity descriptors may contain references to other activities, i.e., sub-activities, and to services (family, class, or instance) needed to perform the activity. As an example, the (basic) activity "Send Instant Message" can be mapped directly to a service family, or to be more precise, to an operation within that service family. Listing 1 illustrates the corresponding activity declaration.

Listing 1. Declaration of basic activity

```
(1) <activity>
(2)   <ID>Send Instant Message</ID>
(3)   <service>
(4)     <family>IMServices</family>
(5)     <operation>send</operation>
(6)   </service>
(7) </activity>
```

Now, consider a second activity named "Submit for Review", which, in contrast to the example in Listing 1, maps to an operation in an HpS service family (rather than a conventional software Web Service), e.g., the operation `reviewContent(document)` in the family "Document Review Service". In fact, underlying Web Services used in complex activities may be a combination of software- and human services. Through composition of such basic activities a complex activity can be declared as shown in Listing 2.

Listing 2. Declaration of complex activity

```
(1) <activity>
(2)   <ID>Finalize Document</ID>
(3)   <sub-activities>
(4)     <ID ref="Submit for Review" />
(5)     <ID ref="Send Instant Message" />
(6)   </sub-activities>
(7) </activity>
```

These examples show only fragments of actual activity declarations but they should suffice to convey the approach taken in activity modeling. We point out that the definition of new activities does not have to be reserved to system developers. In fact, ACCUSO specifically emphasizes the possibility to design and redesign activities during ongoing collaboration.

2.3.2. Activity execution. All declared activities are available to actors for instantiation and execution. During instantiation, an ID is assigned to the new instance, e.g., "Send Message to John Doe", and the activity declaration is copied (by value). Also, numerous parameters can be specified at instance level, e.g., owner, due-date, dependencies to other activities, etc.

Moreover, sub-activities may be assigned to different team members, thereby providing an intuitive mean of team coordination.

Activity instances may be remodeled or redesigned at every stage of execution. Sub-activities may be added or removed whenever necessary. This flexible model addresses the requirements of ad-hoc collaboration.

3. Use cases

This section describes two methods of integrating a mobile worker into ongoing collaborations in an efficient manner. Both cases assume an existing team which organizes work according to activities, kept persistent in the Activity Management Service. At some point a mobile worker enters the space of the operating team, thereby becoming a potential collaborator.

Case 1: Autonomic service discovery. Suppose that a team instantiates an activity named "Evaluate Open-Source Solution". This activity requires a "Consulting Service" and a final presentation to be held. We specifically mention the need for a final presentation to highlight required spatial proximity. At the time of instantiation, such a service is not available in the Service Registry used by the team. Consequently, a flag "Discover and Notify" is set for the service in this activity. In the current prototype, this is realized through periodic polling by the service consumer. In the next implementation step this functionality will be provided through a publish/subscribe mechanism implemented in the Activity Management and the Service Registry.

Upon entering the team's space, the mobile worker's device publishes the services deployed on it to the local Service Registry. Assuming the mobile worker provides a "Consulting Service", this service is discovered by the potential consumer and the execution of the activity can be continued. The fact that the service is an HpS is transparent to the system, as provisioning and consumption adhere to the same standards as conventional Web Services, e.g., WSDL.

Case 2: User-driven integration. A mobile worker may be integrated into collaborations by declaring activity-to-service mappings he considers to be useful. Issues like trust or rights management within teams are not addressed at this stage; however existing solutions could be integrated. Declared activities may be instantiated or used in modeling of other complex activities. The ability to declare activities allows a team member to easily publish and provide capabilities and opportunities to the team that may not have been considered before.

4. HpS architecture overview

In this section we present a service-oriented architecture that allows human-provided services to be incorporated into ACCUSO. The concept of HpS is applicable to a wide range of collaborative business scenarios in N/V/M teams. Architecture and implementation presented in this paper specifically address the requirements in ad-hoc and mobile teams. These requirements include service discovery in ad-hoc mode and the ability to deploy Web Services on mobile devices such as Smartphones and PDAs. Mobile Web Services pose additional challenges to a service-oriented economy as mobile devices have limited resources in terms of CPU/battery power and memory constraints. Additionally, mobile service may become unavailable due to interruptions in wireless network connectivity and users' movements. Algorithms for disconnection prediction and pro-active network handover strategies have been proposed by the wireless communications community [13] to alleviate service availability problems; however this is out of scope of this work.

The main components of our architecture are depicted in Figure 1. The Activity Management Service is the main entry point for every interaction. From the Service Consumer's point of view, the AM service exposes a WS-interface for activity search and retrieval, and publishing of new activities (Figure 1, 5). The consumer may specify an LDAP filter in form of a RFC 1960 compliant string. A simple example query could be "get all activities that belong to a certain category and are available at a certain location area".

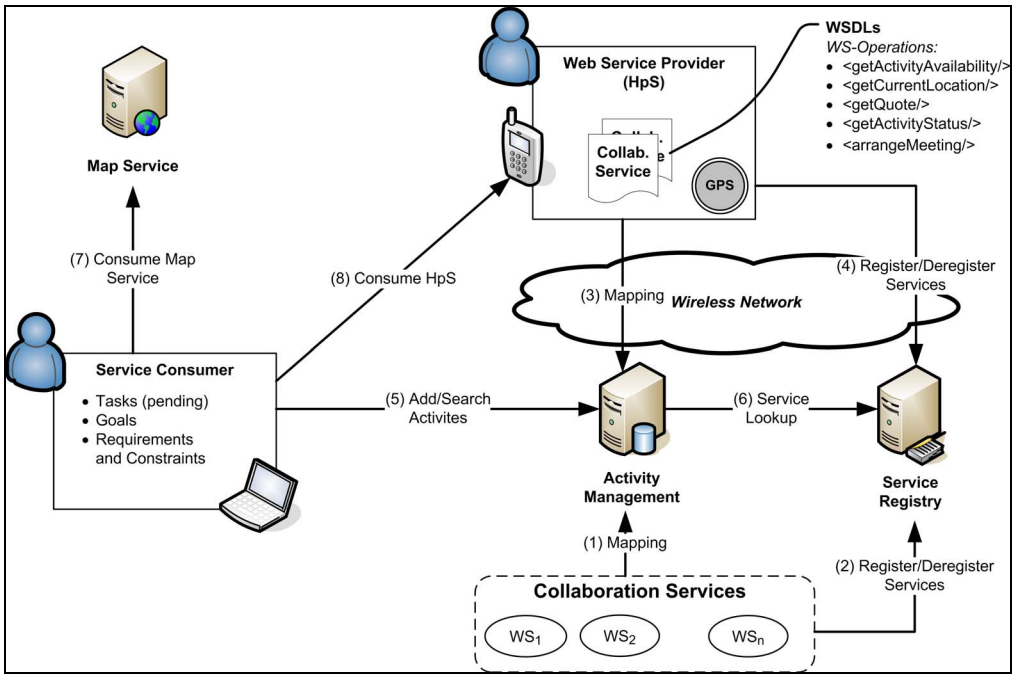

Figure 1. Mobile services in human collaboration

The corresponding filter string is `(&(category=document.reviews)(|(postalCode=1040)(postalCode=1050)))`. Note, that the semantic richness of LDAP filters is rather low. We therefore plan to use richer filters (e.g., allowing for specification of ontological terms) in the future. The Service Registry contains service-related information such as service endpoints (i.e., service locations) and service-descriptions (WSDLs) for published services.

Our system is designed to cope with mobility and networks that may operate in infrastructure mode as well as ad-hoc mode. In particular, we use the service location protocol (SLP) to locate services dynamically. SLP specifies a multicast protocol to enable support for ad-hoc mode. Services may smoothly migrate between managed and ad-hoc networks. If a directory agent (DA) is present on the network, multicast suppression is applied and the central DA manages service registration and requests. Both entities Activity Management and Service Registry have been designed to work on regular computers and embedded systems.

4.1. Activity management service

The class-diagram in Figure 2 shows the layout of the entities stored in the Activity Management Service. The *UserProfile* holds basic user information (i.e., service provider) such as name, email, etc. Each user may specify multiple addresses including home address and business address. If a user provides a business activity, the user may not want to reveal the home address. The *Location* object is more generic and can be used to express the current location. For example, a user is currently located at a particular address or a number of users are located at a particular address. Location information is obtained from a GPS sensor connected to the mobile device (see Figure 1, Web Service Provider).

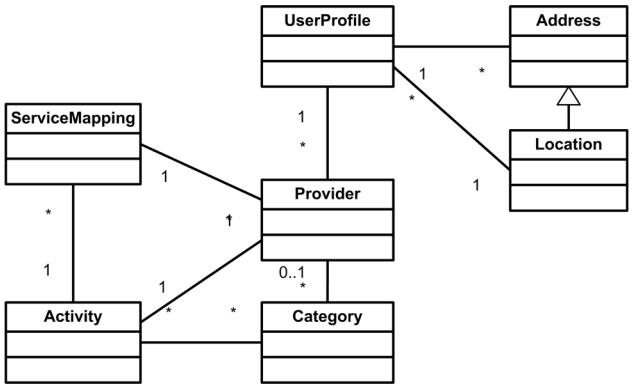

Figure 2. Activity-to-service mapping model

The entity *Provider* allows the user to publish a service for a particular activity. As a result, this entity holds information such as userId, activityId, and a reference to the *ServiceMapping*, which essentially contains the service endpoint and the methods used to realize a particular activity. Note, a service (i.e., an HpS) may be used in a number of activities by fine grained provisioning or association of WS-operations to activities. A *Provider* is commonly associated to a *Category*, which

can be considered as a certain abstract class of services provided in the context of an activity. Finally, an *Activity* may be provided by a number of users and may also belong to a number of categories (depending on the preferences of the author of the activity).

4.1.1. Activity scenarios. Activities that are mapped to Web Services can be used in a number of ways. The basic example is that a user creates an activity declaration along with a service mapping (e.g., Consulting Service). The service consumer locates activities (and corresponding services) by searching the Activity Management Service and may obtain information regarding providers, such as coarse grained location information and their published service endpoints.

The consumer may also want to execute an activity where, however, no service provider exists at this stage (i.e., *pending tasks*). To illustrate this example in depth, suppose the service consumer publishes an activity in form of a job description. Interested human service providers may publish a service mapping in the AM service, hence becoming a potential service provider for an activity or task. The consumer will be notified that a matching service is now available, given his task at hand. This activity or job description may contain a number of contextual constraints and requirements. An example filter: `(&(&(availabilityStart<=08:00AM) (availabilityEnd>=05:00PM))(&(postalCode=1040)(physicalMeeting=yes)))`. The string encodes the required availability time frame of the service provider, a constraint on the location area (neighborhood as a district), and the ability to arrange physical meetings.

In addition to these two interactions, it is likely that services are being developed and made available in form of archives (e.g., jar archive in Java), possibly including activity descriptions. These services can then be downloaded and installed on the service provider's client device and be published along with other services. This idea is well aligned with the vision of OSGi, where Java containers may run on embedded devices and modular services (OSGi bundles) that interact with each other through defined interfaces.

5. Design and implementation

Our current framework is designed to run on mobile computers (Windows XP laptops), mobile devices (PocketPC and Symbian platform), and embedded Linux platforms. The prototype is mainly implemented using Java (e.g., the Activity Management) and .NET for a simple location enhanced activity explorer application. OSGi was our choice for a pervasive service architecture.

Mobile devices are becoming increasingly powerful, not only in terms of hardware capabilities, but also supporting rich Java APIs. Devices such as the Symbian 9.1-based SE m600i [12] support the CDC environment, which brings major advantages for the application developer. For example, CDC supports user defined class loaders, the reflection API and thereby meeting requirements for OSGi container deployment, which was not possible on CLDC-based devices, and also the ability to interact with native methods via the Java Native Interface (JNI). CDC on Smartphones enables us to progress towards richer Java applications on mobiles and also managed services that can be deployed in OSGi containers in form of bundles.

5.1. OSGi container technology and services

Two OSGi containers can readily be used for mobile or embedded devices, the popular open-source implementation from Knopflerfish (short KF) with the overall size in the minimal configuration of 288kb [4]. The other option is Concierge [5], an OSGi R3 implementation with a footprint of 79kb. In our test environment KF was used on the PocketPC (service provider implementation) and Concierge OSGi to run the Activity Management Service. IBM's Service Management Framework (OSGi container suitable for PDAs) has not been tested in our framework.

5.1.1. OSGi-based services. To get an idea of OSGi-based services & dependencies, and expected size of the framework, we provide a list of all required bundles in Table 1.

We have developed an Activity Management Service and an Activity Store (i.e., the database) both packaged in one bundle, listed in Table 1 (1). Nevertheless, AM service and Activity Store (AS) can be installed on separate devices. For example, the AS can be installed on a super-peer node in a network equipped with large storage and multiple AM services that interact with a dedicated AS. We have implemented two versions of the store, a non-persistent store and persistent store. The non-persistent store keeps activity-entries, profiles, etc. in the memory, whereas the persistent store utilizes an embedded SQL database based on HSQLDB, listed in Table 1 (2), to save all activity related information in the file system. All required dependencies are listed at the bottom of Table 1 (6-9), which are needed for Apache Axis as well as kSOAP (details are given in Table 2).

5.1.2. WS-toolkits. In our framework we have used toolkits already available in KF's bundle repository – Apache Axis and kSOAP. Both are packaged as OSGi bundles and can be deployed in KF or Concierge. However, kSOAP has limited features in terms of Web Services support, for example WSDL support, and performance issues when processing SOAP messages [1].

Table 1. Basic OSGi bundles and services

Bundle name	*Size*
(1) activitymgmt.jar	29kb
(2) hsqldb.jar (persistent store)	629kb
(3) axis-osgi_all-0.1.0.jar	1,328kb
(4) ksoap-osgi_all-2.0.0.jar	139kb
(5) jslp-osgi-0.7.0.jar	62kb
Dependencies	
(6) commons-logging_all-2.0.0.jar (35kb), (7) cm_all-2.0.0.jar (62kb), (8) jsdk-2.2.jar (34kb), (9) http_all-2.0.0.jar (90kb)	

Particularly on PocketPCs, we have only a subset of the J2SE APIs available as Java on mobile devices is supported through the J2ME platform (Java 2 Platform Micro Edition). The J2ME specifications define the Connected Device Configuration (CDC), which is a subset of J2SE, and the Connected Limited Device Configuration (CLDC). In contrast to CDC, CLDC provides libraries such as the Connection Framework suitable for devices with a small memory footprint but not part of J2SE. IBM's J9 is a widely used JVM that supports both specifications, CLDC and CDC. We use J9 for our PocketPC based WS-provider implementation. To our knowledge Apache Axis, listed in Table 1 (3), cannot be run on J9 due to a number of missing dependencies that are not supported by CDC. For that reason we use kSOAP, Table 1 (4) to provide Web Services on PocketPC-like devices.

Aside from the dependencies, the Apache Axis OSGi bundle has a rather large footprint in its current configuration, thus not being the best choice for mobile devices. Note that Apache Axis includes WS-client and WSDL generation features, which can be removed to reduce the bundle footprint.

Moreover, we are experimenting with embedded Linux platforms such as Gumstix [8] to host OSGi services (Figure 3). It is well possible to install a JVM (e.g., JamVM – an open-source Java virtual machine for Linux) on such boards. The board shown in Figure 3 features a 400 MHz (Intel XScale PXA255) CPU, 16 MB flash memory, onboard Bluetooth wireless connectivity, and can be extended by connecting a Wifi expansion board. Boards can be flashed with a full Linux (being shipped with pre-flashed Linux v.2.6). We deployed the Activity Management Service along with the Activity Store on this platform using the Concierge OSGi container including the Apache Axis and kSOAP bundle, listed in Table 1, (3) and (4) respectively, as means to provide embedded Web Services. We envision a case where Activity Management Services and Service Registries are deployed on a number of such embedded platforms to provide a local point of attachment (i.e., network neighborhood) for mobile devices equipped with Bluetooth and WLAN interfaces to register their services.

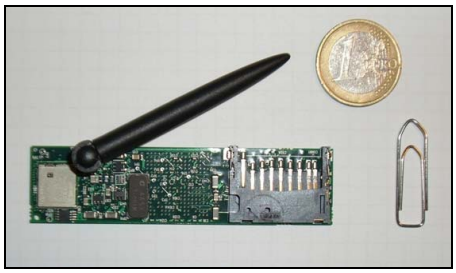

Figure 3. Gumstix embedded Linux platform

Note, Gumstix boards can be configured to act as WLAN access points or Bluetooth PAN (Personal Area Network) servers to enable IP connectivity for situated Laptops, PocketPCs, or SmartPhones and could thereby function as gateways to the Internet. A summary of the (tested) configuration of our testbed is given in Table 2 (K is abbreviated for the KF container and C for Concierge):

Table 2. Configuration of testbed

Role	*Platform*	*K/C*	*Dep.*
Service Provider (*basic bundles*)	Laptop (WinXP)	•/-	(3,5,6,7,8,9)
	PocketPC (Windows Mobile)	•/-	(4,5,8,9)
	Smartphone (Symbian v.9.1)	-/•	(4,5,8,9)
Activity Mgmt./ Registry	Laptop (WinXP)	-/•	(1,2,3,5,6,7,8,9)
	Gumstix (Linux v.2.6)	-/•	(1,3,5,6,7,8,9)

Note that, for the mobile service provider only the basic OSGi bundles including dependencies, as defined in Table 1, are listed. The actual collaboration services, GPS position provider, etc. are not listed as they depend on the specific collaboration use case.

5.1.3. Service discovery. The service location protocol (SLP) can operate in ad-hoc as well as infrastructure mode. If a large number of nodes reside in a particular network segment or neighborhood, a directory agent (DA) can be deployed which results in multicast suppression behavior. When a directory agent detects a probe or resolution request sent by multicast, the DA sends an announcement (DA advertisement) for itself. By listening for these announcements, clients detect directory agents and switch to the DA-specific protocol. If a DA is unresponsive, clients revert back to use the multicast protocol.

For our purposes we can utilize SLP in different modes to discover services. First of all, a Service Lookup component could act as an SLP UA (User Agent) that

scans the network for matching services by means of multicast requests. Secondly, the Activity Management acts as a SLP SA (Service Agent) announcing the AM service that can be discovered by mobile devices (i.e., mobile service provider). Thirdly, the Service Registry implements a DA registering all via SLP announced services.

In our prototype implementation we focused on the second and the third option. An SLP service URL basically consists of three elements, the URL schema "service:", the service type, and the service address specification [11]. The syntax: `service: URL = "service:" service-type ":" site url-path.`

We defined the AM service with following service-url: `service: activityMgmt: http://dan.tuwien.ac.at/activitymgmt.` The part "activityMgmt" denotes an abstract service type (class of service types). In addition, services may have an unlimited number of name/value pairs, called attributes (`attr-id "=" attr-value`). An example for the Activity Management's attributes:

1. persistent = true (i.e., collocated with AS)
2. addressLine = Argentinierstrasse 8
3. city = Vienna
4. postalCode = 1040

The first line denotes whether registrations are saved persistently or not. Lines 2-4 denote the store's address that can then be retrieved from devices to determine their approximate semantic location (neighborhood). "Semantic location" refers to the location information which can be interpreted by humans (e.g., street, city) as opposed to low level sensor information (e.g., GPS coordinates).

Listing 3. Activity management interface

```
(1) public Address getStoreAddress()
(2) public boolean
    registerProfile(UserProfile profile)
(3) public String
    putActivityEntry(ActivityEntry entry)
(4) public ActivityEntry[]
    getActivityEntries(BasicFilter[]
    basicFilters)
(5) public ActivityEntry[]
    getActivityEntries(String rfcFilter)
(6) public boolean
    removeActivityEntry(String authKey)
```

The actual registration of services (e.g., activity-service mappings) is then made by invoking WS-operations on the service. The simple interface is shown in Listing 3. Method 1 in Listing 3 allows consumers and service providers to retrieve the store's address, for example for visualization on a map. Method 2 provides the ability to register publicly available user-profiles including addresses. An activity entry can be added by invoking method 3, which returns a globally unique identifier. Methods 4 and 5 can be used to retrieve activity entries by specifying a search filter, either as RFC1960 compliant string (method 5) or by passing a BasicFilter as argument to method 4 (the BasicFilter is a complex XML data type as opposed to the string representation). Finally, method 6 allows the removal of an activity entry.

Alternatively, as mentioned before, the Service Registry can be operated in SLP DA mode. It would then respond to multicast probes or requests by responding with DA advertisements. For that purpose we have used and extended an open-source Java-based SLP API (jSLP, [6]) that currently supports SLP UA and SA mode. Service provider could for example register a "Document Review" activity by specifying `"documentReview"` as the abstract SLP service-type, the url-path defining the service endpoint, and additional properties such as:

1. activityName = WS-articles-review
2. activityDescription = Expertise in gSOAP TK
3. userId = john.doe@domain.com

The Service Registry updates the Activity Store upon receiving service registrations (i.e., activity mappings).

5.2. Service provider client

We have extended Knopflerfish's AWT Desktop bundle (small footprint of 75kb suitable for mobile devices) allowing the (mobile) user to manage his personal services. A "GPSproducer" bundle is used to obtain GPS traces from an external Bluetooth enabled GPS receiver via the Bluetooth Serial Port Profile. GPS position information can then be consumed by other local bundles. If a service provider reveals his physical location by allowing GPS coordinates to be requested from the provider's Web Service, the map (MapPoint service) can be refreshed periodically to visualize the location information in real-time (Figure 1, 7). Alternatively, if the service provider does not disclose detailed location information, still a location estimate can be given at a higher level of granularity by returning the address of the Activity Management Service/Service Registry where the provider is registered.

6. Related work

The background of our activity-centered perspective on human collaboration is situated in the CWE domain, motivated by aspects of Activity Theory. Dustdar presents an activity-based management system for globally operating teams for process-aware collaboration and coordination [3]. Christensen and Bardram hypothesize that computer systems must handle human work activities as first class objects [9]. They introduce an Activity Management Subsystem capable of supporting mobile

users in ad-hoc collaboration, specifically targeting the healthcare domain. In contrast to our work, they do not employ a service-oriented architecture. Bardram discusses an Activity-based Service Discovery approach based on Jini and UPnP [10], however, not considering mobility aspects. An important aspect of CWEs are workflow management systems (WfMS) and the WS-BPEL specification. In the context of mobile computing, Hackmann et al. introduce a BPEL-workflow process execution engine for mobile devices [7] and similar to our work, utilize the kSOAP package to implement a SOAP server on embedded devices. In our framework we aim at supporting ad-hoc collaboration, as opposed to rather static business processes like BPEL workflows. Note, however, that mining of such ad-hoc interactions could reveal patterns which may serve as an input for workflow modeling. Rellermeyer and Alonso present R-OSGi [6], a project with the goal of providing a seamless and noninvasive middleware for accessing remote services in OSGi frameworks, in particular using Concierge OSGi R3. In contrast, in our research we are interested in embedded Web Services, deployed in OSGi containers that can be used to foster human collaboration.

7. Conclusion

We have presented the concept of Activity-Centric Collaboration Using Service-Oriented Architectures (ACCUSO). In ACCUSO, services are composed in a flexible and reusable manner, namely through activity declarations which can be instantiated and executed by users. The possibility to redesign activity instances during run-time provides the flexibility required in ad-hoc collaboration. Moreover, the hierarchical composition of activities from sub-activities in combination with the possibility of assigning sub-activities to different team members provides an intuitive yet efficient method for team coordination.

Activities are executed using Web Services, contributing to our approach the potential benefits of SOA, such as dynamic service discovery and binding at run-time. Furthermore, we expand the notion of SOA by providing the opportunity to publish Human-provided Services (HpS) providing functionality not realizable through software services. The fact that HpS are "implemented" by a human actor is transparent to the system, thereby allowing provisioning of and interactions with an HpS based on the same infrastructure as used for software (Web) services. We plan to explore provisioning of HpS in the context of N/V/M teams further, for instance, given the goals of a team, who is the expert (i.e., *HpS*) who could contribute to the team's tasks in an efficient manner?

Since two or more services of the same type may be provided by different HpS, potentially having different device capabilities, we are investigating algorithms such as Logic Scoring of Preference (LSP) to rank services and to select the most suitable service.

The issue of mobile WS-providers implemented for Java ME, however, remains challenging. WS APIs such as the JSR-172 only support the WS-consumer model, while kSOAP lacks the support of RPC proxies/stubs (serialization/deserialization of SOAP objects). The EU-IST project Amigo (Ambient intelligence for the networked home environment) addresses this problem by extending kSOAP's features (e.g., stub binding).

8. Acknowledgements

Part of this work was supported by the EU STREP Project inContext (FP6-034718).

9. References

[1] D. Schall, M. Aiello, and S. Dustdar, "Web Services on Embedded Devices", *International Journal of Web Information Systems (IJWIS)*, Troubador Publisher, vol. 2(1), Feb. 2006.

[2] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architecture and Applications*, Springer, 2004.

[3] S. Dustdar, "Caramba - A Process-Aware Collaboration System Supporting Ad Hoc and Collaborative Processes in Virtual Teams", *Distributed and Parallel Databases*, Kluwer Academic Publishers, vol. 15(1), Jan. 2004, pp. 45-66.

[4] Knopflerfish OSGi R4 Framework, online at: http://www.knopflerfish.org/.

[5] Concierge OSGi R3 Framework, online at: http://concierge.sourceforge.net/.

[6] J.S. Rellermeyer, and G. Alonso, "Services Everywhere: OSGi in Distributed Environments", *EclipseCon 2007*, Santa Clara, CA, USA, Mar. 5-8, 2007.

[7] G. Hackmann, M. Haitjema, C.D. Gill, and G.C. Roman, "Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices", *International Conference on Service Oriented Computing (ICSOC)*, Springer, LNCS vol. 4294, Dec. 2006, pp. 503-508.

[8] Gumstix Way Small Computing, online at: http://gumstix.com/.

[9] H.B. Christensen, and J. Bardram, "Supporting Human Activities - Exploring Activity-Centered Computing", *4th International Conference on Ubiquitous Computing*, LNCS, vol. 2498. Springer, Sept. 2002, pp. 107-116.

[10] J.E. Bardram, "Activity-based service discovery – an approach for service composition, orchestration and context-aware service discovery", TR 2004-PB-67, Centre for Pervasive Computing, Aarhus, Denmark, 2004.

[11] RFC 2608 - Service Location Protocol, Version 2, June 1999 (http://www.faqs.org/rfcs/rfc2608.html).

[12] Developers' Guidelines - Java ME CDC, Feb. 14, 2006 (http://developer.sonyericsson.com).

[13] A. Goldsmith, *Wireless Communications*, Cambridge University Press, 2005.