A Privacy-Maintaining Framework for Context-Sensitive Service Discovery Services

Colin Atkinson, Philipp Bostan, Thomas Butter, Wolfgang Effelsberg
Institute of Computer Science
University of Mannheim
Mannheim, Germany
{atkinson, bostan, butter}@uni-mannheim.de, effelsberg@pi4.informatik.uni-mannheim.de

Abstract—**Despite the rapid growth in the number of mobile devices connected to the internet via UMTS or wireless 802.11 hotspots the market for location-based services has yet to take off as expected. Moreover, other kinds of context information are still not routinely supported by mobile services and even when they are, users are not aware of the services that are available to them at a particular time and place. We believe that the adoption of mobile services will be significantly increased by context-sensitive service discovery services that use context information to deliver precise, personalized search results in a changing environment and reduce human-device interaction. However, developing such applications is still a major challenge for software developers. In this paper we therefore present a framework for building context-sensitive service discovery services for mobile clients that ensures the privacy of the users' context while offering valuable search results.**

Keywords: context-aware systems, service discovery, mobile applications.

I. Introduction

Although an increasing number of mobile devices are equipped with the ability to automatically sense and identify their location, the use of location based services has so far failed to reach the levels expected. To date, only a few mobile services are receiving widespread use, such as push-E-Mailing services and navigation services, and a so called "killer application" has yet to emerge. Moreover, other sensor technologies have rapidly evolved over the last few years and now support the automatic sensing of other kinds of context information on mobile devices. Together these provide the basis for the next generation of services for mobile devices and users – context-sensitive services that deliver value tailored to a user's context. However, supporting commercial context-sensitive services that are usable by a broad range of mobile users as well as services that offer high revenue for service and telecommunication providers is still a major challenge for developers.

In contrast to desktop applications, mobile applications must cope with additional problems arising from the influence of their environment. Key challenges include mobility, resource limitations, heterogeneity, personalization and stricter requirements on usability. In view of these constraints, it is even more important to provide mobile users with enhanced support to find suitable services effectively. Humans should not have to engage in long-winded interaction patterns in order to find services since mobile devices provide only limited input capabilities. Users are also usually unwilling to enter large search requests typical of browser-oriented search engines on desktop computers. Context-sensitive service discovery has the potential to deliver significant added value since it considerably reduces the level of interaction required from the user. Furthermore it delivers personalized, precise search results that are tailored to the user's current situation.

To provide better support for context-sensitive service discovery, the SALSA (**S**oftware **A**rchitectures for **l**ocation-**s**pecific Tr**a**nsactions in Mobile Commerce) project of the Mobile Business Research Group at the University of Mannheim has developed a generic service discovery platform for service brokers and service providers. This platform also offers a client framework that supports the development of generic mobile client applications for use with context-sensitive service discovery as well as the dynamic integration and execution of discovered services. An important strength of the SALSA approach is that it considers the user's privacy in the process of context-sensitive service discovery. Especially when sensitive context information like a device's current location is involved, many mobile users fear that context information about them will be misused. As will be presented in this paper, this issue is handled in the overall SALSA architecture by a mechanism that ensures privacy during service discovery.

The main goal of the SALSA project was to develop a generic framework and a reference architecture to support service discovery based on context information. The implementation of our prototype was therefore inspired by the following scenario. A mobile user is moving around in a possibly unknown area (e.g. a foreign city), and would like to use some of the real-world services that are available in that vicinity. The user might also be interested in various electronic services (e.g. a tourist guide) that are specialized for a certain domain and fit his current situation. This is mainly described by his context attributes that may include such things as his current location, the current time, the current weather, and his profile.

Since the number of potentially available services could be very large in this scenario, the system should use this context information to drive the discovery of suitable services. Key-word based searches of the kind applied in desktop browser-oriented search applications are not useful in such a scenario because browsing a large result set is not appropriate for users of small, tiny mobile devices with limited input capabilities. Thus, the approach to context-sensitive service discovery that we present in this paper enhances a mobile user's search request with a set of context attributes that drive the service discovery process. The search results that are returned upon a search request are tailored to the user's current situation, containing a choice of the most relevant services in a personalized, ranked order. This tremendously reduces the human-device interaction involved in searching for suitable services and thus the user's overall level of effort.

The remainder of this paper is structured as follows. Section II starts with a discussion of previous definitions of the term of "*context*". We analyze these definitions in detail and afterwards introduce our view of context. Then, we introduce a simple context model and the approach used to represent context in our framework. This is followed by a classification of services and our schema for service descriptions required for the matching of context and services. In Section III, the overall SALSA architecture is discussed in more detail. We first introduce the basic principles of a Service Discovery Service (SDS) that uses context for the process of service discovery. Following that, the server and the client framework are presented which both offer systematic support for the development of applications and services that support context-sensitive service discovery. In Section IV, we describe the underlying context framework that supports the handling of context on different layers. These layers include context sensing, context resolution and context aggregation and inference. A detailed consideration of our approach for context-sensitive service discovery, based on the matching of context and service descriptions, is presented in Section V. We also introduce our approach for transforming service descriptions as well as our approach for context matching. In Section VI, we present past research work that is related to context-sensitive service discovery and frameworks. Finally, Section VII closes the paper with a summary of our presented approach and technology.

II. Context and Service Representation

In this section we introduce and discuss previous definitions of context and present the definition that we use in our approach. We then present a simple context model and a representation format suitable for context processing and matching within our framework. This is followed by the presentation of various definitions related to services and a schema that supports the matching of services against context for the purpose of service discovery.

A. *The Definition of Context*

In past research, many definitions of the term "*context*" for context-aware computing have been introduced. In [2], Baldauf et al. present a survey on context-aware systems that contains various definitions of the term "context". Many of the definitions published in early research work have been redefined and extended over time. As the authors in [3] point out, most of the past definitions are based on concrete examples and categories, while other definitions use paraphrases or synonyms. This also indicates the problems involved in coming up with a clear definition of the notion of context.

From a natural langrage point of view, the term is defined in the Merriam Webster dictionary [4] as follows: *"the parts of a discourse that surround a word or passage and can throw light on its meaning"* and *"the interrelated conditions in which something exists or occurs: the Environment, the Setting"*. The first part of this definition can be interpreted to mean that context is something implicit that can be used as additional information to give something an enhanced meaning. The second part of the definition has a more general nature and defines synonyms for context. In the free online dictionary of computing [5] context is defined as: *"That which surrounds, and gives meaning to, something else"*. Both of these definitions leave a great deal of room for interpretation and cannot be transferred directly into a form that allows context-aware systems to determine whether or not different kinds of information can be regarded as context.

One of the first definitions was given by Schilit and Theimer (initiators of context-aware computing with the PARCTab project) in 1994 in [6]. They define software as context-aware if it uses *"location information to adapt according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time"*. In later work Schilit et al. define the three important aspects of context as: *"Where you are, who you are with and what resources are nearby"*. They also enumerate a few examples, like lightning, noise level and others.

In 1996, Brown defines context within the stick-e-document project as: *"Context can be a combination of elements of the environment that the user's computer knows about"* in [7] and also enumerates a few examples. This definition leads especially to the question – how does the computer know about the user's environment? This is also raised by Brown in [8] where he discusses the question of whether context is automatically detected or user-delivered information (implicit vs. explicit). In 1997, Ryan, Pascoe and Morse introduced their definition of context in a mix of paraphrases and examples as: *"Context-awareness describes the ability of the computer to sense and act upon information about its environment, such as location, time, temperature and user identity"* [9]. This definition has been refined by Pascoe into: *"Context-awareness is the ability of a program or device to sense various states of its environment and itself"*. It highlights the implicitness of context. In a later definition he adds that context is: "*the subset of physical and conceptual states of interest to a particular entity*".

This concept of defining context in relation to some entity was later elaborated by Dey et al in [10] leading to one of the most widely used and often cited definitions: *"Context 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 to the interaction between a user and an application, including the user and applications* *themselves"*. Many recently introduced definitions are based on the definition of Dey or extended versions of this definition. Other recently introduced definitions try to offer a more sophisticated view of context [3]. However, not only do they introduce additional complexity, most of these definitions fail to clarify exactly what kinds of information are to be regarded as context.

In summary, we can observe that defining context by enumerating examples is static and places limitations on what can be regarded as context. This can be improved by defining a taxonomy of context categories. However, none of the enumeration-based approaches provide a deterministic way of establishing whether or not a particular piece of information is context. All definitions based on synonyms are also vague since they leave a great deal of room for interpretation. Furthermore, they shift the problem to the detailed definition of the chosen synonym, most of which are too generic. Too much freedom for interpreting the notion of context essentially removes any semantics from the term since everything can then be regarded as context. Using more complex and formal definitions of context that subdivide context into different categories do not solve the basic problem either.

The basic problem with most of the definitions is that it is very difficult, and sometimes impossible, to determine whether or not a particular piece of information can be regarded as context. In order to avoid this, in our project we have develop a refined definition of context that conveys useful semantics (i.e. that distinguishes between information that is context and information that is not context). We think that the most important distinction between context and "pure" information is related to the question of whether context is implicit or explicit information. In [11] Pascoe, gives a definition of context that includes the notion that context is "sensed" information which does not have to be explicitly provided by the human. Lieberman et al. also identify the distinction between implicit and explicit context in [12] and Dey states that context may either be determined implicitly (e.g. from a personalized device) or explicitly by user-delivered data in a login dialogue. It is assumed that in both cases the identity of the user is the same and thus both should be regarded as context.

Considering the example of a weather service, a user could explicitly enter the name of the city for which he wants weather information or the city name could be implicitly determined by sensors that detect the mobile device's current location. In the second case, it is clear that the implicitly determined information (i.e. the city name) should be viewed as context, but what about the first case where the name of a city is explicitly input by the user? What if the city name entered by the user does not correspond to his current location? How can the weather service determine if the entered city really belongs to the current context of the user or not? In general, how can a system determine whether it should treat a particular piece of information as context or not? It is clear that the boundary between implicit and explicit information is somehow fuzzy, but in fact implicitly delivered information, and information derived from it, is less prone to errors and thus more reliable and valuable for applications and services. Furthermore, the decision about whether information should be treated as context within the boundaries of a certain system or application must be made at design-time, since context is relative to the system. For the SALSA project and the work presented in this paper we use the following definition: *"Context is any relevant information that can be gathered implicitly or derived based on implicitly gathered information within the boundaries of a system or application and is used to determine the behavior of a system or service"*.

B. Context Representation in SALSA

The overall SALSA framework consists of a client and a server framework, which both provide general support for the development of context-sensitive mobile applications. Both parts of the framework need to collect and process context, but most of the context is collected on the mobile client and enhanced through additional context processing in the server framework.

For these purposes a common representation of context is needed that is shared between the involved participants. While many complex models for representing context have been proposed [13,14], we have chosen a simple, flexible representation that is easy for service providers to apply and manage. In SALSA, a context set is represented by an XML document that contains multiple context attributes. Each of the contained context attributes is in principle an instance of a certain context data type. To support run-time validation of the context set and to provide a template for the definition of context attributes, we have defined a few standard context data types as an XML schema. Context data types are similar to data types of typical programming languages (e.g. *String*, *Integer*, etc.), but they are extended by a few specific data types (e.g. *PositionCircle* represents geographical areas). An example context set is shown below.

```
<Context>

  <PositionCircle name="SALSA.Position.GeoPosition">
    <Center>
      <Latitude>49.48739</Latitude>
      <Longitude>8.4705</Longitude>
      <Altitude>88</Altitude>
    <Center>
  </PositionCircle>

  <Time name="SALSA.Time.CurrentTime”>
    <Hours>12</Hours>
    <Minutes>32</Minutes>
    <Seconds>21</Seconds>
  </Time>

  <String name="SALSA.Time.Weekday">
    <Value>Saturday</Value>
  </String>

  <Integer name="SALSA.Time.FreeTime">
    <Value>30</Value>
  </Integer>

</Context>
```

This example represents a context set that contains four different context attributes using different context data types using our XML representation schema. It contains the geographical position of a mobile user in GPS coordinates, the time and weekday on which the search request was issued, and the user's available free time inferred from personal data in the mobile client's calendar.

A context attribute is characterized by a specification of its data type, its name and its respective values. The name is constructed using namespaces to provide a globally unique identifier which is important for the context matching process as will be explained in Section V. If an application or service needs data types that are not supported in the standard XML schema, the SALSA framework allows arbitrary new context data types to be added in a simple way. This can be achieved by the extension of the XML schema to include new context data types as needed.

C. Service Types and Descriptions

The approach for context-sensitive service discovery presented in this paper allows various different kinds of services to be discovered. In general, we define a *service* as anything that delivers some kind of value to someone or something. The main distinction we make is between *electronic services* and *non-electronic services*. We define *electronic services* as services that deliver value to the mobile user by electronic means in the form of information. In the scenario introduced in Section I, examples of electronic services are an electronic gastronomy guide, an event-guide, a bargain hunter or a tourist guide. Electronic services can be offered in various different forms, for example as traditional web sites or as Web services using SOAP and WSDL.

A *non-electronic real-world (business) service* is defined as a service that does something to improve the state of a user in a certain way. Examples related to our scenario are a restaurant, a bar, a tourist site or a shop. The Venn diagram in Figure 1 presents the relationship between these service categories. It basically shows that Web services are electronic services and electronic services are services in general.

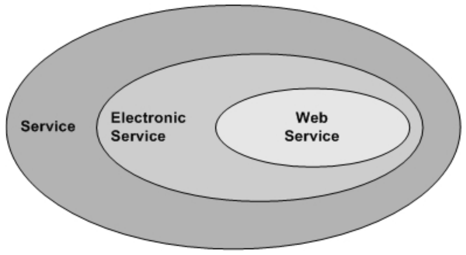

Figure 1. Service Taxonomy.

The following four types of services can be described and discovered using our approach for context-sensitive service discovery:

- real-world (business) services (non-electronic),
- Web services (electronic),
- web sites (electronic),
- SALSA services (electronic).

Real-world (business) services are represented as services in our Venn diagram. The fourth kind of service – the SALSA service – is specific to SALSA and consists of a Web service (representing the service interface) and optionally downloadable software components for dynamic installation and execution in the SALSA client framework. These may be graphical user interface components for interaction with the service, business logic components or security components for example.

To describe these different service types we have developed a generic XML schema that consists of two separate parts: a core part and a domain-specific part. Like OWL-S [15], the core description part, as depicted in Figure 2, is divided into three categories - *ServiceProfile*, *ServiceProperties* and the *ServiceGrounding*. We have extended and tailored the profiles of OWL-S for the purpose of context-sensitive service discovery. The domain-specific part is used to extend service descriptions in cases where additional attributes are needed to describe services of a domain whose properties are not covered by the generic schema. The description of gastronomy places, for example, requires additional attributes to describe their special characteristics. The separation of the core description of the common properties from the domain-specific extensions adds more flexibility and allows service providers to tailor service descriptions to the service's domain.

The **ServiceProfile** mainly contains general information about the service and its provider. First there is a textual description of the service along with its classification according to one of the following schemas: UNSPSC (United Nations Standard Products and Service Codes) [16], the NAICS (North American Industry Classification System) [17] or an arbitrary, self-defined categorization schema. Furthermore, contact information about various key individuals that have some responsibility related to the service is included. The domain-specific part of service descriptions can optionally be added to the *ServiceProfile*. For example, a context-sensitive event guide service that needs to add specific properties to event description can include its own schema in the *ServiceProfile*.

The **ServiceProperties** section of our service description schema introduces the properties of a service, including the spatial and temporal availability of the service as well as aspects like payment and security restrictions. While a real-world (business) service clearly has spatial and temporal restrictions, at first sight it is not so obvious that electronic services may have such properties as well. However consider an electronic context-sensitive shopping guide for the city of Berlin. By providing temporal availability information about the general opening times of stores and malls, the system is able to avoid returning services that are only available during the day in response to service requests issued at night when stores are closed. Also spatially restricting this specialized shopping guide to the Berlin area allows the system to avoid returning this service in response to search requests issued in a different city. While the value of temporal availability attributes for non-electronic (business) services is self-explanatory, spatial restriction information can be used for example to define an area of service delivery. For example, a pizza service might only deliver its service in a circular area with a radius of 10 miles. Finally payment and security restrictions are described in the *ServiceProperties* category. These are mainly used to filter out services during the service discovery process which do not fulfill requirements

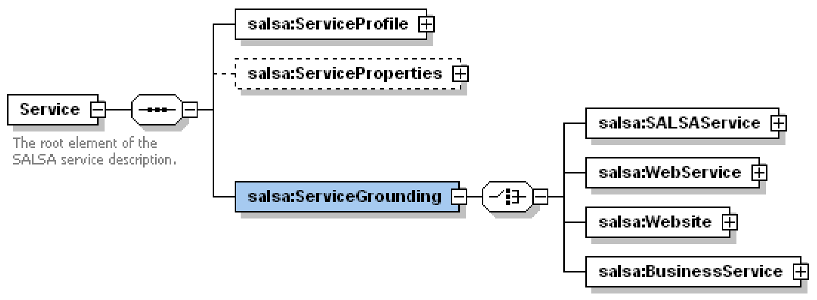

Figure 2. Service Description Core.

concerned with payment methods and security restrictions defined by the user or the service itself.

Finally, the **ServiceGrounding**, the last category of our service description approach, delivers access information for the different services depending on their type. A non-electronic service, for example requires the description of its physical location while an electronic service requires access information like web or server addresses. The difference between the four different types of services – Web Services, Business Services, Websites and SALSA services – is mainly reflected in their different groundings as indicated in Figure 2. Each type of service requires its own description schema for the service grounding. A Web service defines its WSDL grounding, a web site defines its internet address and a real-world (business) service defines its physical and geographical location. While the first three types obviously provide the necessary access information, the latter one, SALSA Service, needs to be considered in more detail. This type defines Web service port types for its service interface and optionally multiple internet addresses for downloadable software components to support dynamic reconfiguration of the mobile client.

III. The SALSA Architecture

In this section, we introduce the architecture that was developed within the SALSA project to support context-sensitive service discovery. We first introduce the basic ideas behind our *Service Discovery Service (SDS)* based on the principles of service-oriented architectures. We then present the architecture of the prototype we implemented to verify the scenario introduced in Section I. This is followed by a description of the SALSA server framework that supports the implementation of context-sensitive SDSs. Finally, we consider the SALSA client framework that offers a simplified way to implement arbitrary context-sensitive mobile client applications as well as client applications that interact with *Service Discovery Services*.

A. *Service Discovery Services*

At the heart of our overall context-sensitive service discovery approach is the concept of a *Service Discovery Service* (SDS) that acts as a service broker in the SALSA architecture. Following the principles of the famous SOA triangle shown in Figure 3 an SDS has the role of the service broker (registry). The SDS allows service providers to register their services using a service description that follows the schema introduced in Section II. A mobile client corresponds to the service requestor and uses the SDS as a service broker to find suitable services. The mobile user finally uses or consumes a service, which is either an electronic service or a real-world (business) service.

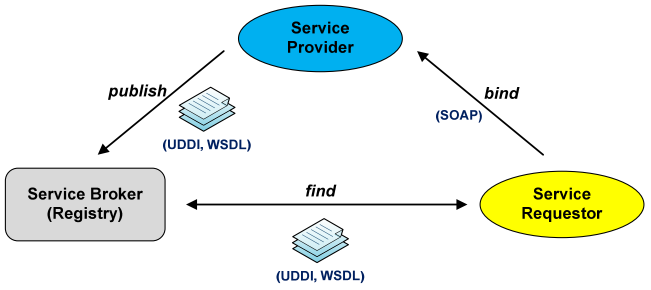

Figure 3. The SOA triangle.

In another paper [18], we have identified and analyzed several configurations that could theoretically be applied for the scenario that we introduced in Section I. For various reasons (evaluated also in [18]) we adopted the *User-Managed Linear Configuration* depicted in Figure 4 for our prototype.

This configuration is characterized as “user-managed” and “linear” because the user is involved in the selection of lower-level service brokers that are also SDSs. The main advantages of this configuration are enhanced privacy, transparency in pricing of SDSs and much lower bandwidth requirements since communication between the mobile client and the SDSs is minimized. Also, the client application consists of less complex software. The user-managed, linear configuration presented in Figure 4 involves the *User*, a *Mobile Client*, the *Universal SDS (USDS)*, multiple lower-levels *SDSs* and multiple *real-world (business) service providers*.

According to the scenario introduced in Section I, the mobile user is interested in immediately getting value from real-world (business) services that are relevant to his current context. Thus, the mobile client first sends an initial search request to the Universal SDS (USDS) that acts as a first-level service broker. This initial search request consists of implicit context collected on the mobile client. The USDS uses our context-sensitive service discovery technology to return a list of service descriptions of lower-level SDSs. These are specialized service brokers that have registered themselves as service providers at the USDS. Examples of lower-level SDSs are tourist guides, event guides and gastronomy guides. The mobile user selects one of these specialized service brokers from the returned list of suitable services (e.g. based on costs or user rating, etc.) and the mobile client then connects to the service broker. A new search request, enhanced by new explicit parameters (e.g. domain-specific parameters input via the service's graphical user interface) and the determined context set is then sent to the chosen lower-level service broker (SDS) which returns a list of suitable service providers. These service providers (e.g. a restaurant, a bar, a café, etc.) in turn have registered their services with the specialized second-level or lower-level SDSs.

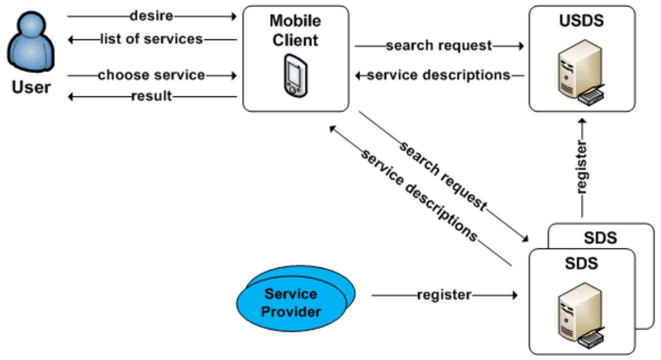

Figure 4. User-Managed Linear Configuration.

In the case where the lower-level broker is a gastronomy guide, for example, the returned list consists of an ordered set of service descriptions representing gastronomy places. On the other hand, in the case of a bargain hunter service, a list of descriptions of shops with special offers (e.g. coupons, etc.) is returned. In all cases, the returned services are ordered by the degree to which their properties matches the context of the request, as will be explained in Section V. Finally, the mobile user can choose a service provider from the list and consume the service (e.g. by going to eat in a restaurant, visiting a tourist site, using coupons in a shop, etc.). In the case where the lower-level broker has registered electronic services that are even lower-level brokers, this interaction can be performed over multiple levels.

In Figure 5, we present an alternative configuration which is called the *Server-Managed Hierarchical Configuration*. Using this configuration, each SDS aggregates search results from specialized lower-level SDSs in a hierarchical way. This minimizes communication between the mobile clients and SDSs, but it reduces the flexibility in user-managed service selection and especially leads to problems when payment is involved. In this configuration the user has no control over which SDS is contacted. Furthermore, as we have already discussed in [18], multiple alternative configurations are possible. For example, SDSs can be federated by location or category. Moreover, mobile clients can aggregate search results from lower-level specialized SDSs returned by the USDS, similar to the presented configuration in Figure 5.

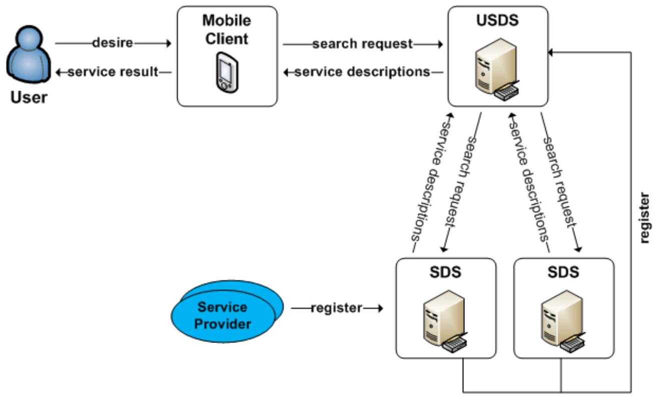

Figure 5. Server-Managed Hierarchical Configuration.

B. Server Framework

Our SDS technology takes the form of a generic server framework that implements all the functionality needed to realize services offering context-sensitive service discovery. In this subsection we introduce the basic components of the SALSA server framework. Since our architecture follows a component-based design, all of the system components can be exchanged with other supporting technologies as long as the interfaces remain the same. For example, our service registry realized using an XML database could be replaced by a relational database. Figure 6, presents an overview of the server framework and its various components.

In principle, each SDS consists of a *Service Registry*, a *Service Search Engine* and multiple components that are responsible for context processing. Service registration can be supported in various ways. The service broker may provide web-based forms on a web site portal to support domain-specific service registration and the management of the registered service description. Another possibility is the automated transformation of existing service descriptions or database schemas to the XML document representation and automated registration via the API of the *Service Registry*. In our prototype the registry is realized using the Natix XML database [19] developed at the University of Mannheim. Natix stores the XML-based service descriptions directly and uses an optimized query mechanism based on the XPath query language [20] to retrieve XML documents. Every incoming search request, enhanced by a context set, is handled first by the *Service Search Engine* component which coordinates the following processing steps:

- pre-processing of the received context set,
- querying the registry for service descriptions with explicitly defined query parameters,
- transformation of the service descriptions into a contextual representation for matching,
- matching the context set against the transformed service descriptions.

The pre-processing of the received context set is handled by the *Context Manager*. It invokes the *Context Resolution Engine (CRE)* to enhance context using external *Context Provisioning Services (CPS)* and the *Context Aggregation* component to infer new context attributes. The *Service*

Search Engine is responsible for querying the registry for service descriptions using XPath queries based on the explicitly defined search parameters. The explicitly defined search parameters in a search request mostly relate to the domain-specific extensions of service descriptions (e.g. a user might specify "*Italian*" kitchen within a gastronomy guide service).

The *Service Transformation* component is used to transform pre-filtered service descriptions into a contextual representation for context matching. Finally the *Context Matcher* is used to match the pre-processed context set against the transformed list of service descriptions so that the *Service Search Engine* can return a list of suitable services to the mobile client. In Section V we will consider this process in more detail when we refer to context-sensitive service discovery.

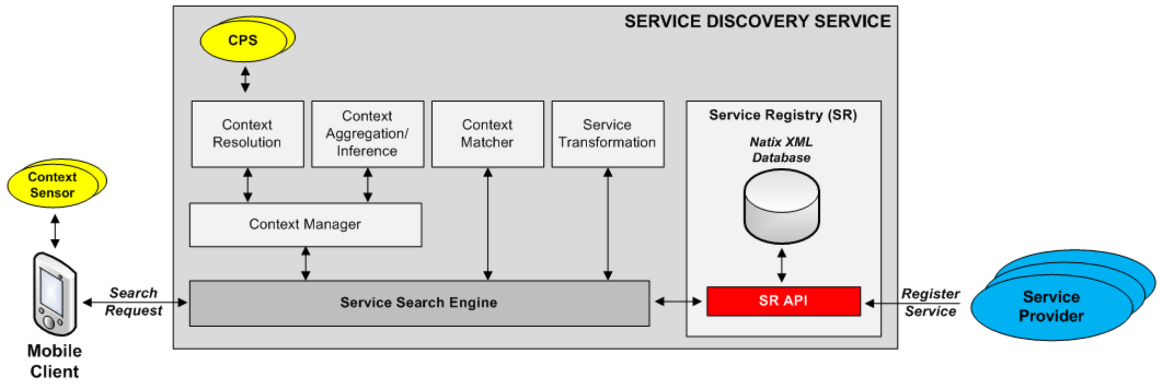

Figure 6. The SALSA Server-Framework.

C. Client Framework

This subsection introduces the SALSA client framework that provides the basic support for the development of mobile client applications that are able to interact with SDSs for context-sensitive service discovery. The client framework has been developed in a generic and component-oriented way in order to support the implementation of independent, self-contained context-sensitive applications for other purposes as well. An overview of the client framework and its components (described in another paper in more detail [21]) is presented in Figure 7. In the following paragraphs we will briefly explain each of the framework's components.

The *Context Manager* component is responsible for the management of context sources and the updating and delivery of context attributes. The *Context Manager* can be subscribed to Context Sources that use either a "*pull*" or "*push*" mechanism to obtain current context information. A further responsibility of the *Context Manager* is the administration of context attributes that can be declared by the user as "*public*", "*private*" or "*blurred*" as will be explained in more detail in Section V. If search requests are initiated by the mobile user, the *Context Manager* is responsible for creating a context set that contains all available context attributes and their respective values. This context set is serialized in an XML representation and embedded in every search request that is sent to a service provider as an implicit parameter. Finally, the *Context Manager* is used for local context matching when search results are received by the client and need to be matched with "*private*" context attributes as will be elaborated in Section V.

The *Component Manager* of the client framework is responsible for the life-cycle management of components used to reconfigure the mobile client application. An important capability of the SALSA framework is that service providers can provide downloadable components to enhance the client-side graphical user interfaces, business logic or security services. Furthermore context sensors and sources can also be added to a client application by the *Component Manager*.

The generic *Communication Framework* implemented in SALSA offers support for well-established Enterprise Computing communication protocols such as SOAP or IIOP from the Common Object Request Broker Architecture (CORBA) [22]. It therefore offers an API on top of the communication layer so that different communication protocols can be supported. The current version implements the SOAP protocol [23].

The *Security Manager* is responsible for ensuring secure communication. It is therefore connected to the previously introduced *Communication Framework*. Furthermore it optionally offers anonymous communication using a TOR (The Onion Router) anonymity network approach.

The *GUI framework* [24] implemented in the SALSA framework is based on the XML User Interface Language (XUL) and offers support for the implementation of adaptable user interfaces to cope with several issues. The XUL approach separates the presentation and application logic whilst offering portability for different Java ME platform configurations. This furthermore eases the porting of graphical user interfaces to different client devices with different configurations. The reconfiguration and adaptation of user interfaces is especially useful if the current context changes (e.g. the mobile user is driving at high speed in a car and the content presentation is adapted accordingly). The implemented GUI framework avoids extensive programming effort for developers of mobile applications.

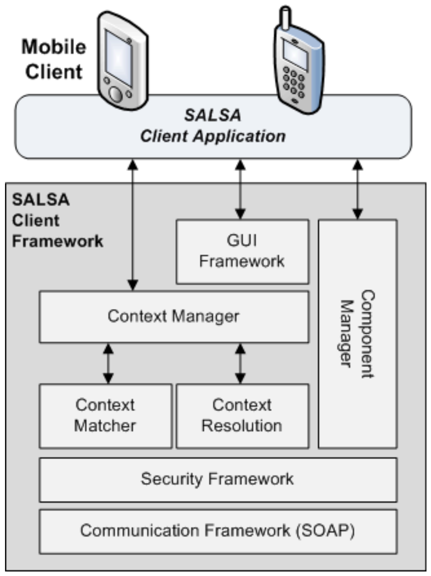

Figure 7. The SALSA Client-Framework.

IV. CONTEXT HANDLING IN SALSA

Before we introduce our approach for context-sensitive service discovery in Section V, in this section we describe how context is handled in the SALSA framework. This essentially involves different context and service description post-processing steps which are needed to match the context to transformed service descriptions. To this end, we first introduce a layered model of context processing followed by a detailed consideration of each layer.

A. Context Processing Layers

The basic goal of context processing is to enhance the client-delivered context information to the highest possible level to support higher precision service retrieval in the process of context-sensitive service discovery. Figure 8 introduces the different context processing layers which are in general supported by the framework. Each of these layers is realized through different components and mechanisms. We will first discuss the layers in an abstract way, and then introduce details of the mechanisms and examples for each layer in the following subsections.

In the first layer, which is called *context sensing*, raw sensor data received from context sensors is delivered to the context sources. These convert raw data into SALSA context attributes using the specified context data types. Context sources are the primary providers of context attributes for the upper context layers. In the second layer, named *context resolution*, a mechanism is applied which enhances the low-level context into new, higher-level context attributes. These new context attributes either represent a low-level context attribute with another meaning or a new, independent context attribute which has been derived from low-level context attributes. The delivery of new context attributes in this layer is realized by so called *Context Provisioning Services* (CPS) which will be explained in part C of this section in more detail. Finally, the third layer, called the *context aggregation and inference* layer, introduces another kind of high-level context that is generated from multiple context attributes based on pre-defined rules.

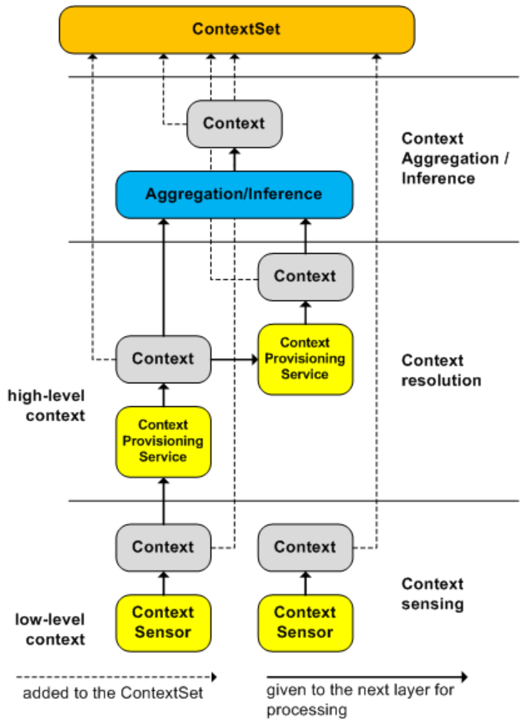

Figure 8. Context Processing Layers

All context attributes together build the context set that is used in the process of context-sensitive service discovery. As indicated in Figure 8, context attributes need not necessarily be processed to higher-level context and can be placed directly from the first or second layer into the final context set used for subsequent context matching. In SALSA, both the client and the server framework support context processing. While the server framework supports all layers presented in Figure 8, only the first two layers are supported in the client framework due to the restricted resources available. The second layer, context resolution, is only supported in a limited way in the client framework for the same reason.

B. Context Sensing

The essential ingredients for high precision service retrieval, based on context-sensitive service discovery, are context sensors and context sources which deliver the initial context attributes. Context processing in general starts on the mobile client with the delivery of data detected by context sensors in the form of raw sensor data (e.g. the current location in GPS coordinates) or components on the mobile client that act as context sources and deliver implicit context attributes like the current time or the user's free time. Theoretically, different context sensors and sources can be integrated into the *Context Manager* in the client as well as in the server framework, but in practice they mostly reside on the mobile client. Context sources and sensors work together either in a push or pull model. Context sources are registered at the local *Context Manager* which automatically pulls context attributes from registered context sources when a search request is issued by the mobile user. Alternatively, the context sources push the context attributes based on a local event or at regular time intervals.

To verify our approach and build our first prototype, we implemented a few context sensors and context sources which deliver different kinds of context attributes collected on mobile clients. The main focus was on the development of high-precision positioning technologies. This was realized using a sensor fusion approach that supports algorithms for indoor and outdoor positioning based on GPS, wireless network and Bluetooth technologies. As well as the positioning of mobile users, we have developed algorithms that use a digital compass to determine a mobile device's alignment [25]. Using the context sensors for positioning we have implemented several context sources that offer context attributes like the geographical position, alignment or speed, which is estimated using collected positioning data over time. Other context sources implemented for our prototype deliver context attributes like the mobile devices screen resolution, color depth, network speed, connection type, current time, personal free time and other information derived from user profiles.

C. *Context Resolution*

Moving to the next layer in our context processing architecture, the low-level context delivered by primary context sources is resolved to create additional context attributes for the final context set. As explained previously and shown in Figure 9, Context Provisioning Services (CPS) are the services that deliver this additional capability. In principle, there are two different kinds of CPSs available. The simple form of CPS acts only locally and is mostly based on simple resolution algorithms that, for example, resolve the current date into a week day or the current time to the time of day. The other kind is more complex and uses external services to resolve context attributes - for example, turning the current location from GPS coordinates into a city name (e.g. using an external GeoService) or resolving a city name into the weather information for that city. Both of these examples use external services to obtain the required context attributes. Due to their complexity and the fact that they use external services that require network and power resources which are expensive in mobile networks, complex CPSs are generally only used within the SALSA server framework. In contrast, simple CPSs may be used in mobile clients as well.

Once the context manager of a mobile client has pulled all context attributes from the registered context sources when a search request is initiated, the resulting context set is processed in the *context resolution* layer by the *Context Resolution Engine*. This processing step is executed on the client framework with a simplified engine and on the server framework with full functionality using the implicit context set in an incoming search request. The architecture of the *Context Resolution Engine* is based on SOA principles as presented in Figure 9.

CPSs are registered at the CPS registry that stores all descriptions. The *Context Resolution Engine (CRE)* receives a context set and uses the CPS registry to search for suitable CPSs. It then invokes these directly to resolve the context attributes that are contained in the context set. The description of a CPS contains a unique service ID, a name and a link to a WSDL description. All CPSs on the server framework are implemented as Web services while those on the client framework are implemented as simple classes. The most important part of the description provides information about the context data type and attributes that the service is able to resolve and the data type and attributes it finally delivers as a result.

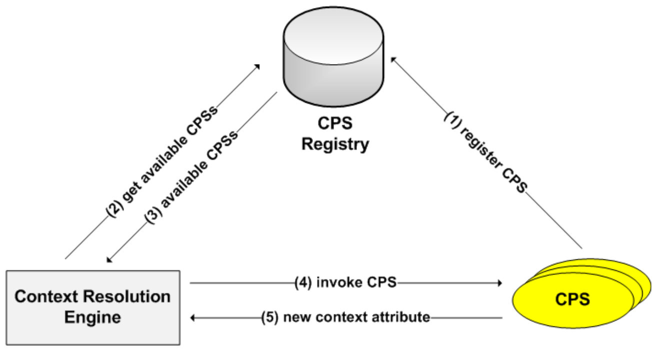

Figure 9. Context Resolution

The API of the CRE in general offers three possible usage modes. In the first mode, a request for context resolution specifies the context attribute of the context set that should be resolved and the CRE tries to find a suitable CPS in the registry that is able to do this resolution. In the second mode, a request specifies the context attribute that should be delivered to the CRE. The CRE then looks in the CPS registry for CPSs that can resolve to this context attribute. With the required input specified in the CPS description, the CRE checks if the provided context set contains the required context input. In the third mode, which is the most prevalent mode, the resolution request specifies the context set as a parameter. The CRE then takes the context set and goes through each context attribute, checking whether there is a CPS in the registry that can resolve a context attribute to new context. If this is the case the CRE sends a call to the CPS, receives the resolved context and adds it to the beginning of the context set description. When all context attributes that are contained in the context set have been processed, the described procedure starts again going through all context attributes of the list since it could be possible to further resolve one of the newly added context attributes. This recursive process continues until the CRE finds no more context attributes to add. Finally all possible resolutions of the client-delivered context set have been applied and the CRE finally returns the enhanced context set to the *Context Manager* for further processing within the SALSA framework.

D. *Context Aggregation and Inference*

The last step in context processing takes place in the third layer named *context aggregation and inference*. This is appropriate for applications that need enhanced context attributes. These mechanisms mostly map the meaning of multiple context attributes describing a situation into a new context attribute based on certain assumptions and conditions. We use a simple but practical approach for formalizing these in our framework based on a rule engine. The advantages of a rule engine are efficient evaluation of rules and support for the dynamic definition, refinement and extension of rules without the need to re-compile or re-install the service implementation. Since the implementation is still in a rudimentary form we will give only a simple example in this paper.

```
if (temperature > 25.9 AND skyconditions == sunny){
  weatherConditions=goodWeather
} else {
  weatherConditions=badWeather
}
```

The above rule is represented in a common programming language style and needs to be adapted to the rule engine's language. It takes two weather attributes and aggregates them to infer a new context attribute representing the weather condition at a higher-level.

V. Context Matching

The final step in our approach for context-sensitive service discovery is the matching of the pre-processed context set against services represented by their service descriptions. In this section we first present the basic ideas behind our context matching approach, and then we provide a description of the service transformation mechanism and the intrinsic step of context matching.

A. *Service Discovery*

After the mobile client has issued a search request and the pre-processing of the initial context set has been finished, the next step in our context-sensitive service discovery process is the matching of context information with potential service descriptions. This is handled by a two-step matching process. As previously mentioned in this paper, a search request in SALSA always consists of explicit and implicit parameters. In the first step of the matching process, the explicit parameters are used in an XPath query to pre-filter service descriptions that contain the specified properties. The second step uses the implicitly delivered context set and the pre-filtered service descriptions in a process that we refer to as context matching in the SALSA framework.

Our analysis of user requirements indicated that one of the main concerns of users related to mobile applications is privacy [26]. Therefore, we developed a context matching approach that provides an option to maintain the privacy of the user's context. As introduced in Section III, in the SALSA framework the mobile user is able to label context attributes as "*private*", so that they will not be sent to the server in search requests. The user can also set certain context attributes to be "*blurred*". For example, the geographical position may be set to be artificially blurred to make it less accurate. Not all context attributes can be blurred, such as the languages spoken by a mobile user as inferred from the user profile. The standard option for context attributes is "*public*" where the context attribute is submitted in a search request with exactly the value that has been determined.

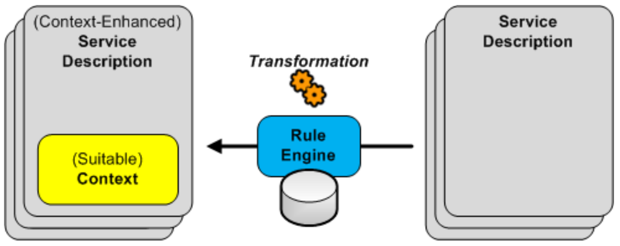

Figure 10. Service Transformation

Our approach to context matching is based on a transformation mechanism as illustrated in Figure 10. It uses rules to transform each service description that matched in the first step into a contextual representation. This context set represents the optimal context that is most suitable for the service under consideration. After the transformation, the context set is embedded within the service description.

Given the requirements for privacy, the main advantage of this approach is that context attributes that have been configured as private can still be used locally with the same context matching approach on the mobile client. This filtering and personalization is performed using the list of service descriptions returned by the SDS. Since each description contains its own transformed context set, the *Context Manager* in the client framework is able to apply context matching locally. Another major advantage of this approach is its flexibility with respect to rule definition, where rules for transformation can be added, changed and deleted dynamically at anytime and in an easy manner. Using this mechanism, the privacy of context attributes can be preserved since the service provider who receives the search request is not aware of the exact context of the mobile user, but is still able to deliver valuable search results [26].

B. *Transformation of Service Descriptios*

In this subsection we present our approach to service transformation. Before going into detail, we start by illustrating our approach using the previously introduced context-sensitive gastronomy guide as an example service. The gastronomy guide is a specialized service that allows gastronomy places to register their real-world (business) service with information that is mapped to a description defined and supplied by the gastronomy guide service provider. This description schema is based on the core service schema presented previously in Section II and a domain-specific extension to describe the special properties of gastronomy places.

For example, in a service description, the domain specific fact "*outdoorSeating*" indicates that a gastronomy place offers seating outdoors. This fact may be captured by the following rule (presented in a simplified notation).

```
  if (outdoorSeating) {
     weatherconditions=goodWeather
  }
```

The above rule is applied for each service description that is returned after matching the explicitly defined properties of the mobile user's search request with the service descriptions in the gastronomy guide SDS registry. If the fact "*outdoorSeating*" is contained in the service description, the rule is applied and a context attribute "*weatherConditions*" is set to the defined value.

As a second example, we may define a rule based on the type of gastronomy place.

```
if (FastFoodRestaurant) {
      freeTimeMin=20
  }
else if (Restaurant) {
      freeTimeMin=60
}
else if (Café) {
      freeTimeMin=30 {
}
```

Depending on the type, we set the context attribute "*freetime*" to a certain value depending on the previous rule specification (e.g. a fast food restaurant or a café requires less free time than a regular restaurant). Applying all specified rules to the service descriptions, the transformed facts which are represented as context attributes are added to the context set that is embedded into the service description and used for later context matching.

In the server framework we realized this approach using Drools [27] which is a Java-based rule engine that supports the description of rules using XML. Following the principles of Drools, or of rule engines in general, each rule contains a condition (left hand side) and a conclusion (right hand side). When a certain condition is true (e.g. a certain fact has been discovered in the service description), the rule is triggered and the associated action is applied. Drools allows the conditions and conclusions within the left and right hand parts of a rule to be defined in two ways, either using Java statements or XSL style sheets.

We have identified two different kinds of rules, static rules and dynamic rules, which can be applied in our transformation approach. The static rules are triggered by facts that are elements of the core service description. We have therefore defined a set of standard rules and routines for the transformation process using the Drools Java style, which could also be replaced at any time by the XSLT templates style. Dynamic rules, on the other hand, are triggered by facts contained in the domain-specific extensions of service descriptions as introduced in the examples of this section. For this kind of rule we have defined standard transformations to pre-defined context data types. Thus, service providers who want to implement context-sensitive services using the SALSA framework only needs to map facts to a standard transformation with a condition and a conclusion. These rules are called dynamic rules, since they can be changed, refined and deleted anytime.

The presented transformation approach applies the same context representation used in context processing using the same data types and namespaces. This is a prerequisite for the context matching approach that will be presented in the next subsection.

C. *Context Matching*

The final task in our context-sensitive service discovery approach is the matching of the client-delivered and pre-processed context set to the server-side transformed service descriptions, each containing its optimal context set. The process of context matching iterates over all pre-filtered service descriptions, extracts their optimal context set and iterates over each contained context attribute. For each context attribute, the data type and the namespace are extracted and the context matcher iterates over the client-delivered context set and searches for context attributes from both descriptions that have the same data type and namespace. For each equal context attribute the context matcher applies a predefined matching routine that is defined for each context data type.

The result of this context matching process is a list of services ordered and ranked based on the degree to which context attributes match. To transform service descriptions into the contextual representation, the service provider may choose which context attributes should be mandatory and which should be optional for matching. If a context attribute is marked as optional, then the matching degree is calculated based on the matching of optional attributes. If a context attribute is marked as mandatory, then one non-matching mandatory context attribute may lead to a matching degree of zero. Note that context attributes are only matched if they appear in both kinds of descriptions. If a mandatory context attribute does not appear in the client-delivered context set it is not evaluated in the matching process, since it might appear locally as a context attribute to be matched locally on the client. Further mechanisms can be applied at the client side using personalization based on preferences and previously analyzed user behavior to further refine the choice of services [26].

In Section II, we mentioned that our context model and schema may be extended with new context data types if context attributes are required that cannot be represented by an already available context data type. If a new context data type is added to the schema, a new context matching routine needs to be implemented to support context matching for the respective type.

VI. RELATED WORK

Since Schilit et al. initiated research on context-aware computing in 1993 starting with their PARCTab project [28] various other researchers have also focused on the subject of context-sensitive service discovery. However, in the early days, many research projects focused exclusively on context-sensitive service discovery related to hardware, like near-by printers or other low-level services. In the following we give a general overview of work related to service discovery using context and to work that relates to context frameworks.

The CB-Sec project [29] introduces an architecture that focuses on the discovery of Web services using context. To this end, a service description schema was developed that includes constraints, requirements and context functions that are used by a brokering agent to evaluate, filter and rank services that best fit the conditions represented by a specific context. Context is collected by the context gatherer that receives contextual information from software and hardware sensors and is stored over time in the context data base that is available to the whole system. In the CB-Sec project, the context matching process evaluates for each specified context functions if a service is suitable for the requestor at the time of the request. Our approach allows a similar specification of the service's optimal context, but with the advantage of additional context matching on the mobile client under consideration of context privacy.

In [30], Kuck and Reichartz present an approach for the context-sensitive discovery of Web services based on the matching of the user's context and enhanced service descriptions that are stored in a UDDI repository with additional information. Their service descriptions contain information inferred from the syntactical and textual contents of WSDL descriptions as well as feedback information, e.g. the context at the time of service recommendation. Unlike our work, however, this work is restricted to the context-sensitive service discovery of Web services.

In the COSS approach [31], ontologies are used for the description of context attributes and services. Service advertisements and requests are represented as documents, and service requests include attributes defined by the user. An attribute like "nearby" is enhanced by rules that are evaluated during the matching process, for example the user's location is within a certain distance to the service's location. In other work of this research group, the WASP project, a service platform for mobile context-aware applications [32] was developed. In both approaches the rules are defined as actions that are executed if the criterion for a certain context attribute becomes true. The framework is intended more to support context-sensitive applications in general, while our approach directly targets context-sensitive service discovery.

In [33], Korpipää presents the Context Management Framework (CMF) that was created especially for context-sensitive mobile applications. The context manager is the main component of the CMF's. Applications can use the context manager to register for context sources to be able to receive and update their values. The context recognition services can infer new context values from low-level context-sources, similar to our approach of context resolution. The context model applied in CMF is based on RDF. Unlike our approach, however, the CMF offers no mechanisms to ensure the privacy of context information.

Other work, like the NEXUS project [34], the SOCAM architecture [35] or the DAIDALOS project [36] also present and implement architectures for a context framework. Each of these introduces a different model and representation format for context as well as different components and processing. Compared to the architecture in this paper, they offer a more generic approach for the development of context-sensitive applications, while our approach focuses on the context-sensitive discovery of services. Apart from the DAIDALOS project, none of these consider the privacy of context information as we do.

VII. Conclusions

In this paper we have introduced a generic, component-based framework that enables the development of (a) services that support context-sensitive service discovery, and (b) context-aware mobile applications that make use of these services. To build the SALSA framework, we introduced a simple model and representation format for context as well as an extensible and flexible description schema for services that have various advantages. Through the clear separation of context processing layers, we have defined a context processing architecture that can be embedded within mobile clients and within services as needed. The client framework supports the handling of context sources and the management of context attributes using a user-friendly mechanism to configure context attributes with different permissions as introduced in Section III. The client framework also introduces a flexible architecture that offers the dynamic reconfiguration and integration of the provider's service components for execution at run-time.

Within the server framework we have introduced several mechanisms for context handling. The step of context resolution allows service providers to obtain as much context information as possible without the need for complex implementation work. Our novel approach for context matching, based on the transformation of service descriptions into a contextual representation, is a key advantage of our approach. It (a) offers the possibility of dynamic rule declaration for service providers, and (b) allows context matching on the mobile client using "*private*" context attributes and the embedded context representation within service descriptions. Using this approach the privacy of the user's context can be retained.

By implementing the presented prototype scenario with example services (context-sensitive gastronomy guide, event guide, tourist guide and a bargain hunter), we have shown that the approach can be applied and extended in a simple way. Arbitrary context data types can be defined as needed, the service description schema can be extended within the domain-specific part and different kinds of applications that apply context-sensitive service discovery can be created. Furthermore, our prototype implementation for mobile commerce applications shows that services can easily be made context-sensitive to provide high precision and personalized service retrieval. This also minimizes the mobile user's effort in service discovery and opens new revenue chains for service as well as for the context providers.

In future work, we are planning to extend the SALSA framework to support a more complex model of context that better supports the inference and aggregation layer in context handling. We also plan to implement example applications for mobile business to show that our approach can be applied in other application fields in a similar way.

ACKNOWLEDGMENT

This work was supported by a grant from the Landesstiftung Baden-Wuerttemberg.

REFERENCES

[1] C. Atkinson, P. Bostan, and T. Butter, "Context-Sensitive Service Discovery for Mobile Commerce Applications," in Proc. of the 4th International Conference on Wireless and Mobile Communications (ICWMC 08), IEEE Computer Society, Washington, DC, 2008, pp. 352-357, doi= 10.1109/ICWMC.2008.33.

[2] M. Baldauf, S. Dustdar and F. Rosenberg, "A survey on context-aware systems," Int. Journal of Ad Hoc and Ubiquitous Computing, vol. 2 (4), 2007, pp.263–277, doi=10.1504/IJAHUC.2007.014070.

[3] A. Zimmermann, A. Lorenz, and R. Oppermann, "An Operational Definition of Context," in Proc. of the conference CONTEXT 2007, LNAI4635, Springer-Verlag Berlin Heidelberg, August 2007, pp. 558-571, doi=10.1007/978-3-540-74255-5_42.

[4] The Merriam-Webster Free Online Dictionary, http://www.merriam-webster.com/dictionary/context

[5] The Free On-line Dictionary of Computing, http://foldoc.org/context

[6] B. Schilit, N. Adams, and R. Want, "Context-aware computing applications," in Proc. of the Workshop on Mobile Computing Systems and Applications, 1994, pp. 85-90, doi=10.1.1.37.9380.

[7] P. J. Brown, "The Stick-e Document: a Framework for Creating Context-aware Applications," in Special Issue: Proc. of the Sixth International Conference on Electronic Publishing, Document Manipulation and Typography, Palo Alto, A. Brown, A. Brüggemenn-Klein, and A. Feng, Eds., vol. 8, no. 2&3, John Wiley and Sons, June 1996, pp. 259-272.

[8] P. J. Brown, J. D. Bovey, and X. Chen, "Context-aware Applications: from the Laboratory to the Marketplace," Personal Communications, IEEE [see also IEEE Wireless Communications], vol. 4, no. 5, 1997, pp. 58-64.

[9] N. Ryan, J. Pascoe, and D. Morse, "Enhanced Reality Fieldwork: The Context-aware Archaeological Assistant," Computer Applications in Archaeology. Oxford, 1997.

[10] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, "Towards a Better Understanding of Context and Context-awareness," in Proc. of the 1st International symposium on Handheld and Ubiquitous Computing (HUC 99), London, UK: Springer-Verlag, 1999, pp. 304-307.

[11] J. Pascoe, "Adding Generic Contextual Capabilities to Wearable Computers," in Proc. of 2nd International Symposium on Wearable Computers, October 1998, pp. 92-99.

[12] H. Lieberman and T. Selker, "Out of context: Computer systems that adapt to, and learn from, context," IBM Systems Journal, vol. 39, 2000, pp. 617-632.

[13] T. Strang and C. L. Popien, "A Context Modeling Survey," in Workshop on Advanced Context Modelling, Reasoning and Management, UbiComp 2004 - The Sixth International Conference on Ubiquitous Computing, September 2004, doi=10.1.1.2.2060.

[14] P. Dockhorn Costa, G. Guizzardi, J.P.A. Almeida, L. Ferreira Pires, M. van Sinderen, "Situations in Conceptual Modeling of Context," in Proc. of the 2nd International Workshop on Vocabularies, Ontologies and Rules for The Enterprise (VORTE'06), Hong Kong, 2006, doi=10.1109/EDOCW.2006.62.

[15] The OWL Service Coalition, "OWL-S: Semantic Markup for Web Services," http://www.daml.org/services/owl-s/1.1/overview, 2004.

[16] UNSPSC – United Nations Standard Products and Service Codes, http://www.unspsc.org/download.aspx.

[17] NAICS – North American Industry Classification Standard, http://www.census.gov/epcd/www/naics.html.

[18] M. Aleksy, C. Atkinson, P. Bostan, T. Butter and M. Schader, "Interaction Styles for Service Discovery in Mobile Business Applications," in Proc. of the 17th International Workshop on Database and Expert Systems Applications (DEXA), IEEE Computer Society, 2006, pp. 60–65. doi= 10.1109/DEXA.2006.75.

[19] Natix – A native database system for XML, http://pi3.informatik.uni-mannheim.de/~moer/natix.html.

[20] M. Brantner, S. Helmer, C. Kanne, G. Moerkotte, "Full-Fledged Algebraic XPath Processing in Natix," in Proc. of the 21st International Conference on Data Engineering (ICDE), IEEE Computer Society, Washington, DC, 2005, pp. 705-716, doi=10.1109/ICDE.2005.69.

[21] M. Aleksy, T. Butter, and M. Schader, "Architecture for the development of context-sensitive mobile applications" in Int. Journal of Mobile Information Systems, vol. 4,no.2 , April 2008, pp. 105-117.

[22] Object Management Group, "The Common Object Request Broker: Architecture and Specification. Version 3.0.3," OMG Technical Document Number formal/04-03-01, 2004, ftp://ftp.omg.org/pub/docs/formal/04-03-01.pdf.

[23] M. Gudgin, M. Hadley, N. Mendelsohn, J.J. Moreau, H.F. Nielsen, A. Karmarkar and Y. Lafon, "W3C SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)," W3C Recommendation, April 2007, http://www.w3.org/TR/soap12-part1/.

[24] T. Butter, M. Aleksy, P. Bostan, M. Schader, "Context-aware User Interface Framework for Mobile Applications," in Proc. of the 27th international Conference on Distributed Computing Systems Workshops (ICDCSW), IEEE Computer Society, Washington, DC, p. 39, 2007, doi=10.1109/ICDCSW.2007.31.

[25] T. King, S. Kopf, T. Haenselmann, C. Lubberger, and W. Effelsberg, "Compass: A Probabilistic Indoor Positioning System Based on 802.11 and Digital Compasses," in Proceedings of the First ACM International Workshop on Wireless Network Testbeds, Experimental evaluation and Characterization (WiNTECH), Los Angeles, CA, USA, September 2006.

[26] T. Butter, S. Deibert, and F. Rothlauf, "Using Private and Public Context - An Approach for Mobile Discovery and Search Services," In: T. Kirste, B. König-Ries, K. Pousttchi, and K. Turowski (eds): Mobile Informationssysteme - Potentiale, Hindernisse, Einsatz, pp. 144-155, Bonner Köllen Verlag, Bonn, 2006.

[27] Drools – Rete OO, Java Rule Engine, http://jboss.org/drools/documentation.html.

[28] B.N. Schilit, M.M. Theimer, and B.B. Welch, "Customizing Mobile Applications," in Proceedings of USENIX Symposium on Mobile & Location-Independent Computing, USENIX Association, August 1993, pp 129–138.

[29] S. K. Mostefaoui and B. Hirsbrunner, "Context Aware Service Provisioning," in Proc. of the IEEE/ACS International Conference on Pervasive Services (ICPS), IEEE Computer Society, Washington, DC, 2004, pp. 71-80, doi=10.1109/ICPS.2004.13.

[30] J. Kuck and F. Reichartz, "A collaborative and feature-based approach to Context-Sensitive Service Discovery," in Proc. of 5th WWW Workshop on Emerging Applications for Wireless and Mobile Access (MobEA), 2007.

[31] T.H.F Broens, S.V Pokraev, S.V. M. van Sinderen, J. Koolwaaij, and P. Dockhorn Costa, "Context-aware, Ontology-based, Service Discovery," in: European Symposium on Ambient Intelligence (EUSAI), Lecture Notes in Computer Science 3295, Springer, 2004, pp. 72-83.

[32] S. Pokraev et al., "Service Platform for Rapid Development and Deployment of Context-Aware, Mobile Applications," in Proc. of the IEEE International Conference on Web Services (ICWS), IEEE Computer Society, Washington, DC, 2005, pp. 639-646, doi=10.1109/ICWS.2005.106.

[33] P. Korpipaa, J. Mantyjarvi, J. Kela, H. Keranen, and E. Malm, "Managing Context Information in Mobile Devices," in IEEE Pervasive Computing, vol. 2, no. 3, 2003, pp. 42-51.

[34] F. Dürr, N. Hönle, D. Nicklas, C. Becker, K. Rothermel, "Nexus - A Platform for Context-aware Applications," in Proc of the GI-Fachgespräch "Ortsbezogene Dienste", 2004.

[35] T. Gu, H.Wang, H. K. Pung, and D. Q. Zhang, "An Ontology-based Context Model in Intelligent Environments," in Proc. of Communication Networks and Distributed Systems Modeling and Simulation Conference, 2004, pp. 270-275.

[36] M. Strimpakou, I. Roussaki, and M. E. Anagnostou, "A Context Ontology for Pervasive Service Provision," in Proc. of International Conference on Advanced Information Networking and Applications (AINA), 2006, pp. 775–779, doi=10.1109/AINA.2006.15.