Towards Context-Sensitive Service Aspects

Tobias Rho
rho@cs.uni-bonn.de

Mark Schmatz
schmatz@cs.uni-bonn.de

Armin B. Cremers
abc@cs.uni-bonn.de

Dept. of Computer Science III, University of Bonn
April, 15th 2006

Abstract

The advantage of services in terms of a service-oriented architecture is at the same time its disadvantage. Services must be general enough to fit the needs of as many clients as possible. On the other hand, such generality cannot address each single requirement. Although services can be parametrized in order to customize behavior this can only happen in a reasonably outlined domain and must have been anticipated.

We propose *context-sensitive service aspects* to solve this dilemma. Services can be kept compact and unaware of their context and the aspects adapt them to the current environment.

Additionally we introduce an abstraction layer, the context management, which decouples context retrieval and utilization.

1 Introduction

State-of-the-art mobile applications have a strong need for context-sensitive behavior. Location-based services have become widely accepted and are used e.g. for special mobile phone rates or navigation.

But, context-aware behavior is usually hard-coded into the application itself using the deployed libraries on the device. Since these have to be known at development time context-processing is limited to the known libraries on the corresponding target device. Besides that, not all context-sensitivity can be anticipated. Being bound to one device neither composition nor sharing of context information is possible. To build flexible applications that adapt themselves to the current situation, the underlying architecture must provide the means to *dynamically reconfigure* the application based on *context information*.

Service-oriented architectures (SOA) help to support the dynamic reconfiguration of applications. They modularize applications by decomposing them into distributed components, so called *services*. Applications are build by composing these services and configuring them at runtime. Therefore we base our architecture on service-orientation.

Several service composition techniques have been proposed [1, 2][1], but they all miss the possibility to incorporate context information in the composition. We consider this an essential property for a dynamic environment where services, like body function sensors, are only optionally available. Therefore we propose a new composition technique based on aspect-oriented programming which takes context information into account.

The concept of aspect-oriented programming (AOP) [3] provides the means to inject code into existing programs without anticipating the changes in the base program. The code is *oblivious* to the concern implemented by the aspect. Extracting context-dependent behavior into aspects and weaving them based on the current situation can solve the depicted dilemma. This is supported by Martin [4]. He claims that dependencies within software systems should only exist from the stable to the unstable parts. Nordberg [5] mapped this idea to the concepts of AOP and components composition. Aspects help to improve the software design by encapsulating the unstable composition concern. AOP frameworks like [6] have been proposed which realize this concept.

Ambient intelligence introduces an even more unstable element - the varying context, which influences the runtime adaptation of the service composition. Using AOP to encapsulate the service composition the services stay compact and stable because they are independent of adaptation strategies and context information.

However, common aspect languages only consider the event flow of programs. The AOP terminology for program flow events is *join point*. Typical join points are method calls, field accesses or thrown exceptions. In an ambient intelligence setting these join points are not sufficient. The properties of the envi-

[1] In the context of web services a huge number of composition approaches have been developed. We name BPEL4WS only as an placeholder for BPEL, WSFL or XLANG.

ronment must also be taken into account. To combine contexts and join points a powerful pointcut language is needed. Since the concept of context always relies on a certain subject we must also take into account that various subjects have different ways to perceive context. The combination of different context views is necessary to reason about more complex relations between subjects, like auxiliary dates for a group of businessmen[2]. Therefore, the pointcut language must be capable of including information outside the context view of the local machine.

There are also some restrictions compared to common AOSD approaches. We cannot apply AOP techniques in full extend on the SOA level, because the concrete implementation of services is, at most times, not accessible to our local system. And even if the service implementation is available there may still be different views onto the same service from the local or a remote system which are in a different contexts. That is why we restrict the join point model to calls on the service level. This also conforms with the *information hiding principle* [7] which postulates that an evolvable design should provide functionality through a controlled interface and hide the concrete implementation details.

This paper introduces the *Ditrios* architecture and the service aspect language *CSLogicAJ*, which provide *context-sensitive services aspects* for the service-oriented architecture *OSGi* [8].

In the following section we give a brief description of the technologies we base on and point out their shortcomings. Then, in section 3 we address the mentioned shortcomings by introducing the Ditrios architecture. Section 4 describes how our approach represents and manages context information. Section 5 explains how we integrated the processing of context information within Ditrios in terms of adapting services context-sensitively with service aspects.

In Section 6 we describe the realization of service aspects, the language CSLogicAJ, a generic context-aware aspect language for Ditrios. Section 7 gives a real world scenario solved with the new technologies, Section 8 presents an overview of related work and Section 9 concludes this paper.

2 Base Technologies

2.1 OSGi

The *OSGi Service Platform* is a specification introduced by the *OSGi Alliance* in March 1999 [9]. Its aim is to specify a SOA-based framework which allows network delivery of managed services to local networks and devices. Frameworks designed according to this specification enable comfortable registering resp. unregistering and use of modular software components - in terms of OSGi: *bundles* and *services*.

2.2 Shortcomings

OSGi is a framework which has to fit many different requirements. It has to stay technically low-level while providing all further needed support as an API to accomplish the requisites. The following sub sections enumerate the limitations of OSGi concerning an ambient intelligence setting and its particular requirements.

2.2.1 Stale References

Concerning, for example, abrupt service departure often implies the appearance of so-called *stale references* ([9], page 112) which is a very common problem to handle. On the one hand OSGi provides tools and techniques like listeners and trackers ([10], page 333ff) which help dealing with this but on the other hand listeners considered harmful as mentioned and explained in [8].

As a solution [8] introduces the *Whiteboard pattern* which relieves many but not all responsibilities of the developer's burden. However, both approaches imply dealing with a typical middleware concern which should not carried out by the client side.

2.2.2 No direct networking support

OSGi has no direct applicable support for inter-framework communication and hence no support for handling remote services. Though providing the *io* package known from *J2ME* no high-level API exists simplifying remote communication.

2.2.3 Manual service retrieval / management

Concerning SOA, clients have to care about service retrieval themselves. Furthermore, they have to manage the requested services after the retrieval succeeded. Latter implies dealing with the departure and the re-arrival of services and, of course, stale references. This is tedious and actually a typical middleware concern. Considering OSGi in terms of a platform for highly dynamic services this problem even increases.

2.2.4 Manual service composition

OSGi does not provide composition support for services. The standard procedure is to find services over a the service registry, track and use them.

[2]This example is thoroughly discussed in Section 7

This approach does not consider relationships between services and therefore there is no way to express compositions of services. In particular dynamic (re)composition of service according to external circumstances is not supported.

3 Ditrios

Ditrios[3] is a complete SOA framework based on OSGi allowing context-sensitive weaving. Its aim is to combine the technologies of aspect-oriented software development and service-oriented architectures based on OSGi. This integration results in a middleware framework encapsulating the complete service management ranging from searching and tracking to providing of services while remaining fully transparent to the clients at the same time. On top of that so-called *service aspects* give flexible control over services and enable generic adaptation triggered by either method-calls or context changes.

A special interface, the *ClientService*, establishes communication between Ditrios and client applications and thereby provides access to the Ditrios framework over a simple API.

A client inquires services via its ClientService by means of a special request object (which consists mainly of a LDAP search string). Found services are then bound to the corresponding request object so that the client can identify its acquired services over the associated request.

3.1 Service adaption

The power of Ditrios is the support for transparent and easy adaption, extension and substitution of services with the help of service aspects. A newly introduced join point model combined with an extended pointcut language enables this adaption to be generic and context-sensitive.

3.2 Proxy indirection

All services belonging to the same request are wrapped within a proxy. Due to the use of service aspects and their implied policies one adequate service can be chosen as the default service which is then transparently utilized by the client. In the case that all services of a proxy are gone, a "fallback" service can be interconnected which induced by a service aspect.

Because of the proxy indirection clients are not aware of the indirection and also do not have to be aware of stale references, service unavailability, service upgrades[4] or substitutions.

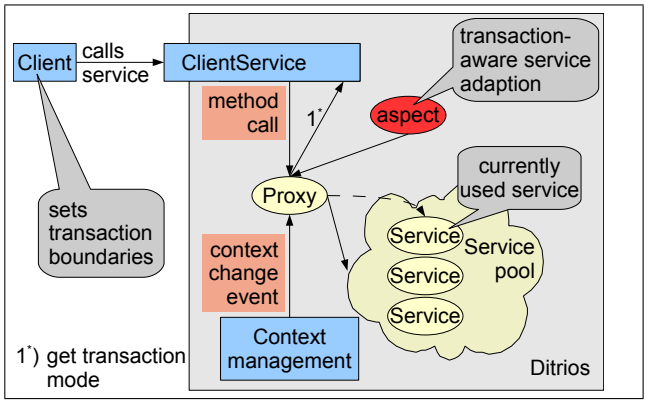

Figure 1: Services are transaction-aware

3.3 Transaction-awareness

Sometimes service adaptions like upgrades and substitutions must not occur while within certain processing state on the client side. Therefore, the client must be able to define transaction boundaries wherein no context change, service upgrade, etc. may affect the program flow.

In order to know about whether being in a transaction or not, the proxy can request the ClientService for the transaction mode and additionally service aspects have in turn access to this information.

Figure 1 depicts how transaction works.

However, if services are semantically equivalent, indicated through their annotated attributes, service substitution may be accomplished.

3.4 Remote capabilities

Ditrios is not limited to locally deployed services. Requests can also be delegated to remote Ditrios systems which in turn try to track the requested services. If appropriate services can be found on a remote Ditrios system references to them are injected to the inquiring ClientServices. For this to be done references to the ClientService instances are also passed to the remote systems. The underlying technique is based on *Java RMI* [11].

3.5 The Ditrios workflow

In order to be able to participate in the Ditrios framework a client has to acquire a ClientService instance which is thenceforward exclusively assigned to

[3] **DI**stributed **TR**acking and **I**nterception **O**f **S**ervices

[4] A service upgrade could arise if e.g. a client would be identified as a legitimate in using some premium service.

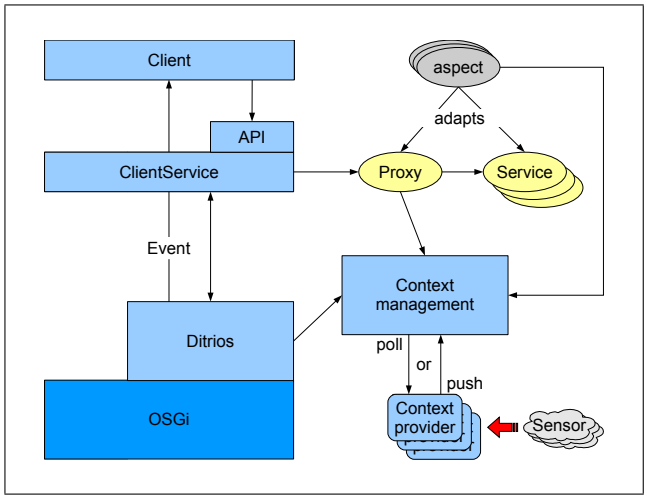

Figure 2: Ditrios workflow

it. The ClientService will immediately be tracked by the Ditrios framework after instantiation[5].

After that the client may request one or more services by defining appropriate search requests. All requests are delegated with the help of the ClientService to the Ditrios core system which in turn attempts to track the corresponding services. A proxy instance wrapping all matching services will be created for each request. References to the proxies are then returned ("injected") to the ClientService so that the owning client can utilize them over the API.

Every status change of the requested services is reflected by an event which informs corresponding clients provided that they implement the appropriate event listener.

Figure 2 depicts the workflow.

Service weaving Because services are wrapped within proxies, Ditrios is able to intercept method calls and weave service aspects in between. Besides this synchronous interception, Ditrios can also weave asynchronously triggered by context changes like e.g. updated sensor information or service arrival / departure.

In section 5 we will explain the weaving process in more detail.

[5]This is achieved due to the use of the Whiteboard pattern [8]. The ClientService itself is designed as a service and only needs to be registered with the framework in order to be used.

Key	Domain	Type
types	WSDL URL	url
accuracy	quality (0-100)	int
accuracy unit	meter | seconds | ...	string
qos[6]	0-100	int
cost	Amount	Currency
cost unit	Sec | KB | ...	string

Table 1: LDAP keys assigned to context services

4 Context Management

4.1 Definition and representation of context

Before we consider context management itself we have to give a short definition of what we call context information.

First of all, the concept of context is always relative. It relies on a certain subject which is the current center of reference. Considering two persons or devices at the same time their perceived context is mostly different. Their view onto the world is different, because they *are* in a different environment and have different capabilities to observe their environment.

We consider all information *perceivable* from the considered system as context. To be perceivable by a computer system we have to make it available in a structured form. At the same time contexts may be perceived in different ways. Therefore we must also consider and order context information by quality.

We structure the context information with the help of *WSDL* [12] types. This decision was influenced by several properties of WSDL. At first it provides global, language independent type descriptions with mappings to a substantial set of languages. Second, this approach naturally integrates with web services.

4.2 Tracking and analyzing context

The service-oriented architecture is facilitated to track and select context information. Context information, like hardware sensors, profile information and service tracking information are advertised as OSGi services implementing the Interface IContextProvider, see Figure3. They optionally specify the keys of Table 1 to enable a reasonable context selection.

The context management system uses a snapshot approach to gather new context data similar to in accordance with common context management systems [13]. On every snapshot the context provider services are asked for changes[7].

[7]We also provide a push mechanism for context information, but since context management is not the main focus we will not

```
interface IContextProvider {
   Delta getDelta(TimeStamp);
}
```

Figure 3: IContextProvider

```
( onchange | before | after | around )
  <name>(<parameters>) : <pointcut expression>
{
      <advice body>
}
```

Figure 4: Semi-formal syntax of CSLogicAJ's generalized aspect construct.

Two kinds of context information are supported: entities and relations - in accordance with the entitiy relationship model. Entities are described as WSDL types, relations as WSDL messages.

Every entity must be globally referable by an unique id. Typical examples are persons, devices, locations. Composition and selection of services is not implemented on this architecture level. The AOP techniques in Section 6.4 are facilitated for this purpose.

5 Service aspects

Service aspects are the heart of the Ditrios architecture. They operate on proxies and services as depicted in figure 2 and provide the means to control dynamic composition of services. Building up the glue between context information and services they enable the latter to be context-sensitive. Since service aspects themselves are enclosed in services holding state for the implemented concern weaving and unweaving can simply be realized through activating resp. deactivating them within the framework.

We distinguish two different kinds of weaving implying two different advice characteristics. Although they share the same syntax they have different semantics. The following will show the difference.

Synchronous advice is applicable on the service message level. Such advice intercept calls to service interfaces taking the program flow and all available context information into account. The commonly known *before*, *after* or *around* advice can be used to execute additional code. Furthermore, transparent (re)binding resp. (re)composing of services is possible due to the proxy architecture. Synchronous advice corresponds to the common dynamic weaving.

Asynchronous advice react exclusively on context changes indicated through *context providers*. Such advice is comparable to event-condition-action (ECA) rules known from relational databases. The *event* is the change of the context, the *condition* is represented by a pointcut designator and the *action* by the advice code. Asynchronous advice does not need program-execution pointcuts.

The terms *asynchronous* and *synchronous* refer to the execution time relative to the application work flow. Based on these two kinds of advice the architecture controls service composition and dynamic code weaving. All information gathered and provided by context providers is consolidated in the context management (see section 4). Due to its provided listener and polling mechanisms[8] service aspects gain access to the context information. With the help of service aspects existing services become context-aware and therefore have the possibility to behave accordingly.

The next section will describes a realization of service aspects - the CSLogicAJ language.

6 CSLogicAJ

CSLogicAJ (Context-aware Service-oriented LogicAJ) is a realization of service aspects. It is based on the generic aspect language LogicAJ. LogicAJ[14, 15] is an aspect-oriented extension of Java that supports meta-variables in pointcuts, introductions and advices. This section only sketches the main concepts of the language. A thorough introduction can be found in [14]. CSLogicAJ only inherits the advice concept, the call pointcut, and logic meta-variables from the LogicAJ language. Synchronous advice (see Section 5) is executed at runtime by intercepting methods calls on the service level of the Ditrios architecture and incorporate the context information from context provider services. Asynchronous advice is identified through the *onchange* advice[9].

The pointcut expression is reevaluated every time the context information the expression depends on changes. But, the onchange advice is only activated if at least one of the declared parameters has changed.

discuss it here in detail.

[8] Depending on the chosen mechanism context changes are received actively or passively.

[9] Which is an abbreviation for "on context change".

```
after() :
     call(?ret ?type.?m(..)) &&
     current_service(?service) &&
     service_attr(?service,
                  "service.id", ?id)
{
   log("called method: " +
     ?"m" + " on service "+ "?id");
}
```

Figure 5: Service level logging call pointcut.

```
import_context "http://../time.wsdl";
onevent() :
  time(?Time, ?_, ?_, ?_, ?_) &&
  element(?Time, "hour", 10)
{
  Time time = ?Time;  // automatically
                      // mapped & casted
  print("time for breakfast: " + time);
}
```

Figure 6: Import context information.

6.1 Logic Meta-ariables

The logic meta-variables share their semantics with logic variables in Prolog. Readers unfamilar with Prolog, or logic programming in general, see e.g. [16] for an introduction. Logic meta-variables are denoted syntactically by names starting with a question mark, e.g. "?entity". They can be used uniformly in pointcuts and advices.

In addition to meta-variables that have a one-to-one correspondence to individual Java language elements, LogicAJ introduces logic list meta-variables that can match an arbitrary number of elements, e.g. any number of call arguments or method parameters. These variables are indicated syntactically by two leading question marks, e.g. "??parameterList". Their introduction is motivated by the fact that in generic application scenarios it is often necessary to express statements like "match *every* constructor invocation". Unnamed logic meta-variables are indicated by an underscore (?_ and ??_). The pointcut may contain several unnamed meta-variables, they are all treated as distinct variables.

6.2 Pointcut Language

CSLogicAJ's pointcut language is a logic language with typed logic variables and all-solutions meta predicates[10]. It allows *remote evaluation* and provides means to select service level join points and context information and supports aggregating the latter. The synchronous advice must contain a call pointcut to select the methods to intercept. Figure 5 gives an example of simple logging aspect which tracks calls to services and writes the method name and service id to a log file. The predefined pointcut *current_service(Service)* binds the called service to its argument, *service(Service)* binds all available services. The pointcut *service_attr(key, value)* gives the developer access to the properties associated with the called service.

[10]We provide the *all-solution* predicates *findall*, *bagof* and *setof* known from Prolog [16].

6.2.1 Importing and Querying Context

To use context information in an aspect we must import the namespace of the context. Figure 6 shows the syntax for the import. Since context information is described by WSDL[11] we only have to reference the WSDL URI for the context description.

Now we can use the declared types in equivalent pointcuts. Figure 6 uses a time context which provides information about the current time. The complex type *time* contains the elements *hour*, *minute*, *second* and *milliseconds* and is mapped to the predicate *time(?TimeEntity, ?Hour,?Minute, ?Second,?MS)*. You can directly refer to one of the arguments via the *element* pointcut. The first argument is the entity, the second the name of the element, and the third its value. Figure 6 demonstrates the use with a restriction of the time value to the 10th hour[12].

Context information, with provided Java bindings, is automatically wrapped into the corresponding objects for every binding and can therefore be easily referred to in the advice body. For every list of objects we provide an array. The minimal set of bindable context values are basic types like int, float, string, boolean etc. The example makes use of the automatic mapping of context to Java objects by assigning ?time to time variable.

6.2.2 Remote Pointcuts

Pointcuts are by default executed on the local context management system. This is not sufficient when we need to query the context of another user, e.g her/his profile information.

In this case the remote execution of pointcuts is possible. All we need is the URI of the remote Ditrios system to execute the query. This information is e.g. available when the other user is already known to a context provider like a location provider. A concrete example is given in the example section in Figure 11.

[11]Other languages like e.g. OWL are also imaginable.

[12]The pointcut time(?Time, 10, ?_, ?_, ?_) is equivalent to the pointcut description in Figure 6. The figure just demonstrates the use of the element pointcut.

6.3 Service Composition

On the OSGi platform the client selects a service by querying the OSGi platform and keeps a reference to the queried service. Here, the clients should not know any details about context-aware service variants and therefore we must be able to replace bound services on the architecture level.

Therefore we provide service *composition* means[13] in the aspect language. All services can be queried by the pointcut *service(?Service)*, the pointcut *service_attr* described in Section 6.2 resolves the attributes of the service. Since the client, which uses a service, is not by itself identifiable, a rebinding of a service is made based on the request issued by the client, see Section 3 for details on service requests. Therefore we provide a *request(?Request, ?LDAP)* pointcut which binds all request and their LDAP queries[14]. The pointcut *requested_services(?Request, ??Services)* binds the services resolved by the platform to *??Services*. The pointcut *current_request(?Request, ?LDAP)* binds the current request for a synchronous advice. The pointcut *in_transaction(?Request)* checks the transaction status of the request's client, see Section 3.3 for transaction awareness.

To rebind a service the method *bind(Request, Service)* is available in the advice body of the aspect. The advice in Figure 7 exemplifies the use of the bind method. The before advice intercepts all calls to methods of the Time service and binds the corresponding Time service and request. Line 7 checks that the client is currently not in an transaction. Otherwise the client assumes unvarying behavior from the service and we would then not be allowed to change it. In line 8 the current time zone bound via the corresponding context provider. The lines 10 - 12 look up an service with the same *time_zone* for the current position. If the time zone has changed and a *Time* service for the current country exists the currently bound time service is replaced with the new time service in the advice body.

6.4 Context Provider Selection

The service composition mechanisms are also used to manage the available context provider services (Section 4.2). If different context provider services offer the same kind of information, e.g. the current GPS position, aspects can be used to choose between them based on their context specification given in Figure 1.

[13] In literature also the terms *service choreography* is used for the binding of services.

[14] Actually, we use a request object which encapsulates the LDAP string instead of applying the LDAP string directly. The object can be identified by its id.

```
 1 import_context
 2   "http://../time_zone.wsdl";
 3 before(?country) :
 4  call(?_ Time.?m(??args)) &&
 5  current_service(?old_time_zone) &&
 6  current_request(?request) &&
 7  not(in_transaction(?request)) &&
 8  time_zone(?time_zone) &&
 9  service(?curr_time_zone) &&
10  service_attr(?curr_time_zone,
11         "time_zone",
12             ?time_zone) &&
13  not(equals(?old_time_zone,
14             ?curr_time_zone)) {
15   bind(?request, ?curr_time_zone);
16 }
```

Figure 7: Runtime service rebinding.

The specification is available in the service attributes and can be queried by the *service_attr(?service, ?key, ?value)* pointcut.

Figure 8 illustrates how aspects are facilitated to implement a policy for the context provider selection. First, an aspect X requests a certain context information, here the location. The context management will query the service registry for a qualified context provider. At that point the synchronous aspect Y controls the selection among the available context providers based on their LDAP properties (see table 1). The aspect may at that point consider user preferences or consult the user for concrete selection. If new context providers become available in the future the aspect's policy will be evaluated again and a new context provider may be selected.

6.5 Exception handling

In dynamic environments like ambient intelligence settings services typically fluctuate. There may be several reasons, like an unstable network connection or deactivation of a service on another device. Therefore exception handling for remote calls is essential. Since this is not the focus of this paper we only sketch the Ditrios solution here. In our setting two typical failures may occur.

1. A used service is not available anymore while a method call is executed.
2. An remote pointcut evaluation is interrupted.

In the first case a *ServiceRemoteException*[15] will be thrown which is passed to the Ditrios architecture. Synchronous advice can intercept the exception with

[15] A runtime exception defined by Ditrios.

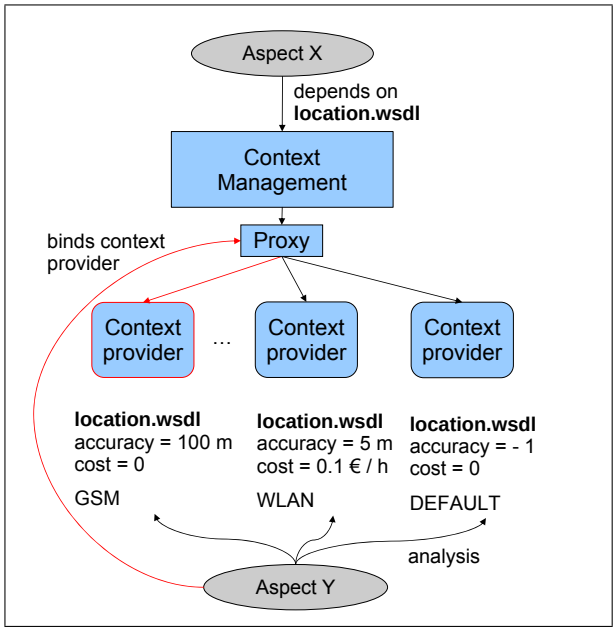

Figure 8: Selecting between different content providers.

the pointcut *serviceException* [16] and rebind the service request to another, e.g a fallback or default service. Different rebinding alternatives for different situations can be implemented with different advices. If no pointcut matches the exception the ServiceRemoteException is thrown to the calling client.

In the second case the evaluation of the enclosing advice's pointcut designator is stopped and the pointcut evaluation is repeated. This is necessary since bindings for the pointcut's arguments may already exist from previous backtracking steps.

7 Example

Consider the following scenario: two business people are at an exhibition. They know each other and have an appointment the next day. However, for both it would be a great help if they could meet already today easing their schedule tomorrow. This, of course, requires knowing that the other is nearby. Unfortunately they do not know about that. This example motivates that awareness of arbitrary context information, like the presence of other persons, can help managing daily routines. The prior example represents any situations where further knowledge would be important or at least advantageous.

A solution for the described scenario would be a *appointment optimizer* service aspect[17]. Assuming that both partners use mobile devices each running the Ditrios framework they could have deployed a very naive *date book* service and entered the appointment for the agreed time. However, for collaboration purposes the date book's data modell must be available externally.

The workflow would be as follows: in the moment of entering the exhibition hall the business men's mobile devices resp. the Ditrios systems on them participate in the exhibition hall's Ditrios system[18]. Seconds later they receive a notification about newly available services. This could be everything but in our case only the mentioned *appointment optimizer* service aspect and a *device resolver* service matter. Now, having access to the offered services the devices are henceforth able to optimize existing appointments.

In more detail, the *onchange* advice[19] defined in the optimizer aspect

1. gathers all contacts from the *date book* service where appointments exist within e.g. the next two days[20].

2. resolves all nearby contacts.
   This is achieved through a *device resolver* service.

3. queries the resolved candidate contacts for today's free time periods.
   For this to be done the contact's remote Ditrios system is queried through remote pointcut evaluation (see 6.2.2).

4. calculates auxiliary dates.

Concluding, in the advice body the user will be notified about possible optimizations for each candidate. An appropriate GUI displays all candidates with the calculated auxiliary dates and offer the notification of the respective counterparts in order to arrange bringing forward the appointments.

The following figure depicts the workflow.

7.1 The aspect implementation

Now we explain the date optimizer aspect in detail. Its purpose is to compose and analyze the necessary context information and find the auxiliary dates for present contacts. On every change of dependent context the condition is reevaluated.

Figure 10 show the complete aspect. At first we import the context specifications from the given WSDL

[16] The *serviceException* pointcut binds the service request, the properties of the service and the name and the arguments of the called method.

[17] The aspect is in turn wrapped within a service.

[18] This could be done over a *gateway* service.

[19] introduced in section 10

[20] The time period would be configurable, of course.

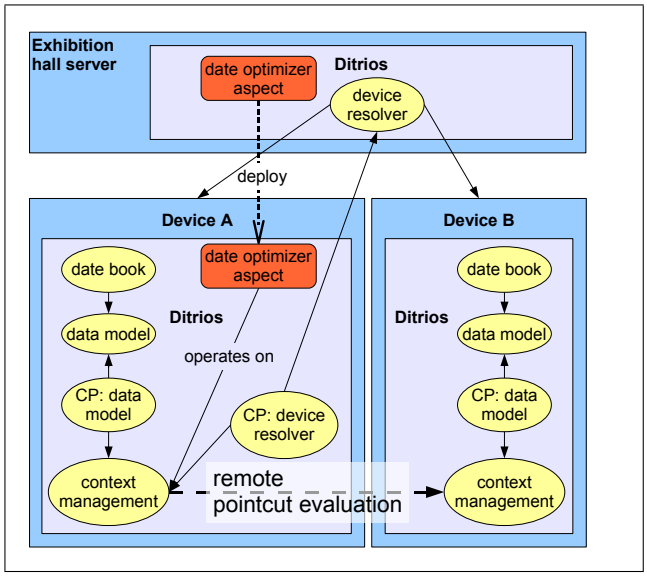

Figure 9: Deployment diagram for the example

```
import_context "http://../appointment.wsdl";
import_context "http://../location.wsdl";
import_context "http://../time.wsdl";
import sam.person.*;
import sam.location.*;

aspect AppointmentOptimizer {
  pointcut dates_next_days(?Appm) :
    today(?Today) &&
    element(?Appm, "date", AppmDate) &&
    date_diff(?Today, ?AppmDate, ?Diff) &&
    between(1, 3, ?Diff);
  pointcut nearby(?Person) : ...
  pointcut ditrios(?Attendee,
                   ?DitriosURI) : ...
  pointcut free_periods_today
             (??FreePeriods) : ...
  pointcut calc_aux_dates(
        ??FreePeriods, ?AuxDate) : ...
  onchange(??Attendees, ??FreePeriods) : ..
}
```

Figure 10: AppointmentOptimizer

descriptions. For the scenario we need access to our appointments, the current location of people at the exhibition and we need to know the current time. Then we define some auxiliary pointcuts to structure the pointcut description of the forthcoming onchange advice. The first advice *dates_next_days*, which was exemplarily implemented, looks up the appointments for the next three days. The *nearby* pointcut determines if an person is in an radius of 300 Meters[21]. The *ditrios* pointcut looks up the device used by the attendee. This relation is provided by the location context provider. The device exposes the unique ditrios URI which is needed to reference the ditrios system and query its context management system.

The *free_periods_today* pointcut binds the list meta variable ??FreePeriods to ranges of free time slots in the schedule of the other attendee. Finally, the *calc_aux_dates* pointcut compares these time slots with the local date book and binds ?AuxDates to all possible auxiliary dates. If no auxiliary date can be found the pointcut fails.

Figure 11 shows the asynchronous advice which composes the pointcuts in its pointcut designator. It uses the forall predicate to bind all attendees and their auxiliary dates to the ??Attendees, and ??AllAuxDates meta variables[22]. All defined pointcuts are composed to one logic expression whereby the free_periods_today pointcuts is evaluated remotely at the attendees devices. If the pointcut succeeds the attendees and auxiliary dates are bound and mapped to arrays of Java objects in the advice body. Now a GUI is opened and the user is told about the possible schedule changes.

8 Related Work

Most context-aware platforms concentrate on the context management providing a well-defined interface for the application. The application anticipates the interfaces to the context management and is therefore itself context-aware. One example is the SOCAM approach[17] by Gu et al., an approach built on top of the OSGi framework. Context reasoning is carried

[21] A more sophisticated version could make configurable in the user profile.

[22] The semantics of the findall predicate is ajar, but slightly different from the semantics known from Prolog.

It binds the bindings of the first argument variables to the list of variables of the last argument instead of binding one list of bindings for all variables.

```
onchange(??Attendees) :
    findall([?Attendee, ??AuxDates],(
     appointment(?Appm) &&
     dates_next_days(?Appm) &&
     element(?Appm, attends, ?Attendee) &&
     nearby(?Attendee) &&
     ditrios(?Attendee, ?DitriosURI) &&
     remote(?DitriosURI,
       free_periods_today(??FreePeriods)
     ) &&
    calc_aux_dates(
         ??FreePeriods, ??AuxDates),
     [??Attendees, ??AllAuxDates]),
    not(empty(??Attendees))
    {
        Person[] atts = ??Attendees;
        Appointment[] auxs = ??AllAuxDates;
      openGUI(atts,auxs);
    }
```

Figure 11: onchange advice

out with a rule system based on first-order logic. The context model is described with the help of the Web Ontology Language OWL[18]. Automatic adaptation of the program based on the current context state is not supported.

Dargie et al. [19] introduced an approach for the analysis and modification of event traces based on context information. This work assumes that all operations are carried out via a central event system. Jadabs[20] is an OSGi based AOP approach for distributed services. Services can be adapted at runtime with the dynamic aspect language Nanning. Jadabs does not propose an context model and only considers the local state in its join points.

Fuentes et al. [21] propose a ambient intelligence DSL for architecture level adaptations. Adaption strategies can be written which change the application at runtime. The analysis of the context is carried out with an extension of the OWL which provides first-order logic analysis. The main differences to our approach are, that the system only reacts on concrete events and executes all analysis locally on the current system.

[22] introduce a context-aware aspect language. Here it is possible to adapt the advice execution to the runtime state of the program. A Java-based context model is proposed and the pointcut language of AspectJ is extended to bind and evaluate the current context. Their concept is a pure language level approach.

The Ditrios architecture provides an exception handling mechanism by means of aspects. Architecture level exception handling for distributed services has also been proposed by [2]. The *DeEvolve* platform provides an exception handler concept with user interaction to look up service alternatives on failure. The handlers are defined in a XML description which is also used for the service composition. Although the means are different the possibilities are similar. The aspects can used for automatic reconfiguration or incorporate the user into the selection process.

9 Conclusions and Future Work

Working with service-oriented architectures in general and OSGi in particular implies dealing with middleware concerns, more than ever if the involved services are highly dynamic. However, the developer should focus on the business logic and not care about these middleware details. Changing the application depending on context changes usually requires a separate context management system which must be manually queried.

This paper presented the service-oriented architecture Ditrios and the generic aspect language CSLogicAJ for the Ditrios architecture which liberate the developer from considering middleware concerns and context-awareness in their services. They can use an aspect languages which uses a unified logic language to query services, contexts and the program execution. The language can consider different contexts, local and remote context, at the same time making complex consideration of the environment possible. And, the aspect language provides the means to reconfigure service composition, by controlling all services references.

Not all components of the presented approach are implemented, yet. The Ditrios platform, LogicAJ and local context management are implemented and will be integrated in the future. Several optimization problems are still to solve. The reevaluation of the pointcut expressions in asynchronous advice should be optimized in future, since the complete, global reevaluation is expensive. Techniques from deductive databases, like the *view update* [23], could be facilitated which only calculate the difference between former and current query evaluation. Since the generic parts in CSLogicAJ are expanded at runtime the recompilation of the code can lead to an enormous runtime overhead. We developed caching strategies to avoid this overhead, but we still have to evaluate the solutions.

References

[1] J. Klein F. Leymann D. Roller S. Thatte F. Curbera, Y. Goland and S. Weerawarana, "Business process execution language for web services, http://dev2dev.bea.com/techtrack/bpel4ws.jsp".

[2] Sascha Alda and Armin B. Cremers, "Towards composition management for component-based peer-to-peer architectures", in *Proceedings of the Workshop Software Composition (SC 2004)*, April 2004, pp. 42 – 58.

[3] "Aspect-oriented software development", http://www.aosd.net.

[4] Robert C. Martin, "Design Principles and Design Patterns", June 2004.

[5] Martin E. Nordberg III, "Aspect-oriented dependency management", pp. 557–584.

[6] Shigeru Chiba and Rei Ishikawa, "Aspect-oriented programming beyond dependency injection", in *ECOOP*, 2005, pp. 121–143.

[7] D.L. Parnas, "On the criteria to be used in decomposing systems into modules", *Communications of the ACM*, vol. 15, no. 12, pp. 1053 – 1058, December 1972.

[8] OSGi Alliance, *Listeners Considered Harmful: The "Whiteboard" Pattern - Revision 2*, August 2004.

[9] OSGi Alliance, *OSGi Service Platform Core Specification - Release 4*, August 2005.

[10] OSGi Alliance, *OSGi Service Platform Service Compendium - Release 4*, August 2005.

[11] Sun Microsystems, Inc., *Java Remote Method Invocation (Java RMI)*.

[12] W3C, *Web Services Description Language (WSDL)*, March 2001.

[13] Andreas Zimmermann, Andreas Lorenz, and Marcus Specht, "Applications of a context-management system", in *CONTEXT 2005*, 2005, pp. 556–569.

[14] Tobias Rho and Günter Kniesel, "Uniform genericity for aspect languages, technical report iai-tr-2004-4, computer science department iii, university of bonn", in *Uniform Genericity for Aspect Languages, Technical Report IAI-TR-2004-4, Computer Science Department III, University of Bonn*. Dec 2004.

[15] Günter Kniesel and Tobias Rho, "A definition, overview and taxonomy of generic aspect languages", *L'Objet*, vol. to appear, 2006.

[16] W. F. Clocksin and C. S. Mellish, *Programming in Prolog*, Springer-Verlag New York, Inc., New York, NY, USA, 1987.

[17] Tao Gu, Hung Keng Pung, and Da Qing Zhang, "Toward an osgi-based infrastructure for context-aware applications", in *IEEE Pervasive Computing, vol. 03, no. 4*, Oct-Dec 2004.

[18] C. Welty M.K. Smith and D.L. McGuinness, "Owl web ontology language reference", http://www.w3.org/TR/owl-ref, Feb. 2004, World Wide Web Consortium (W3C) recommendation.

[19] Waltenegus Dargie, Olaf Droegehorn, and Klaus David, "Sharing of context information in pervasive computing", in *In Proc. of the 13th IST Mobile and Wireless Communication Summit*, June 2004, pp. 839 – 843.

[20] Andreas Frei, *Jadabs - An Adaptive Pervasive Middlerware Architecture*, No. 16273, ETH, October 2005.

[21] Fuentes and Jimenez, "An ambient intelligent language for dynamic adaptation", ECOOP Workshop OT4AmI, 2005.

[22] Eric Tanter, Kris Gybels, Marcus Denker, and Alexandre Bergel, "Context-Aware Aspects", in *Proceedings of the 5th International Symposium on Software Composition (SC 2006) LNCS, Springer-Verlag.*, March 2006.

[23] Andreas Behrend and Rainer Manthey, "Update propagation in deductive databases using soft stratification", in *ADBIS 2004, Budapest, Hungary.*