The Journal of Systems and Software 82 (2009) 1285–1297

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Context-aware service engineering: A survey

Georgia M. Kapitsaki [a,*], George N. Prezerakos [b,1], Nikolaos D. Tselikas [c], Iakovos S. Venieris [a]

[a] *Intelligent Communications and Broadband Networks Laboratory, Dept. of Electrical and Computer Engineering, National Technical University of Athens, 9 Heroon Polytechneioy str, Zografou, 15780 Athens, Greece*
[b] *Software and Service Engineering Laboratory, Dept. of Electronic Computing Systems, Technological Education Institute of Piraeus, Petrou Ralli & Thivon 250, 122 44 Athens, Greece*
[c] *Dept. of Telecommunications Science and Technology, University of Peloponnese, Karaiskaki Str., 22100 Tripolis, Greece*

ARTICLE INFO

Article history:
Received 18 April 2008
Received in revised form 15 February 2009
Accepted 21 February 2009
Available online 6 March 2009

Keywords:
Context
Context-awareness
Service engineering

ABSTRACT

Context constitutes an essential part of service behaviour, especially when interaction with end-users is involved. As observed from the literature, context handling in service engineering has been during recent years a field of intense research, which has produced several interesting approaches. In this paper, we present research efforts that attempt mainly to decouple context handling from the service logic. We enumerate all context management categories, but focus on the most appropriate for service engineering, namely source code level, model-driven and message interception, taking also into account the fact that these have not been dealt with in detail in other surveys. A representative example is used to illustrate more precisely how these approaches can be used. Finally, all three categories are compared based on a number of criteria.

1. Introduction

Adaptive, personalized services that take into account location as well as other user-related data predicate the existence of well-formed information with respect to user's environment, referred to as contextual information or context. Many precise definitions of context can be found in the literature (Chen and Kotz, 2000; Schmidt et al., 1999). However, according to the classic definition by Dey and Abowd (2000), "*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*". The techniques that enable the exploitation of contextual information in services are generally known as "context handling" techniques, while the use of context to provide relevant information and/or services to the user, where relevancy depends on the user's task, is known as "context-awareness". Context handling is of vital importance for developers and service architects, since it provides dynamic service behaviour, content adaptation and simplicity in usage for the end-user.

For the above reasons, a strong trend in context-aware services research is clearly visible in the last few years, involving issues like raw context retrieval from the environment, service design and development, usability studies, etc. The paper approaches context-awareness in services from the viewpoint of service developers, i.e. it is assumed that context information has been retrieved from the environment via appropriate mechanisms (e.g. sensors) and that a basic processing has already been performed. Thus, contextual information becomes available by invoking dedicated context providing services, which can be combined with service engineering techniques, where service engineering can be defined as the specialization of software engineering that targets the development of applications for consumption by end-users. More specifically, we are interested in service development strategies that allow the decoupling of context handling from the core service logic, since this approach leads to simpler software process models especially in the stages of design and code writing for context-aware services. The survey serves as a comprehensive presentation towards the research community regarding the state of the art on interesting methodologies in context-aware service engineering. Moreover, it can be used as a reference for anyone interested in designing and creating such services with a focus on the end-user and can assist developers in choosing among the potential solutions the approach that best suits their development objectives.

The paper is structured as follows. Section 2 lists the context management categories and Section 3 introduces a reference scenario for the rest of the paper. Section 4 is dedicated to source code level approaches for context-aware service engineering, whereas Section 5 presents model-based approaches and corresponding context models. Message interception approaches are described in Section 6. Section 7 hosts a comparison of the presented approaches and finally Section 8 concludes the paper.

* Corresponding author. Tel.: +30 210 772 2551; fax: +30 210 772 1092.
E-mail addresses: gkapi@icbnet.ntua.gr, georgina.kapi@gmail.com (G.M. Kapitsaki), prezerak@teipir.gr (G.N. Prezerakos), ntsel@uop.gr (N.D. Tselikas), venieris@cs.ntua.gr (I.S. Venieris).
[1] Tel.: +30 210 538 1132; fax: +30 210 538 1260.

0164-1212/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2009.02.026

2. Categories of context solutions

During our survey we have considered a variety of solutions to context-aware service engineering proposed by different researchers and developers and tried to distinguish the main tendencies in regard with programming paradigms, development patterns and mechanisms proposed. Based on this, the approaches can be roughly divided in the following categories:

(1) Middleware solutions and dedicated service platforms.
(2) Use of ontologies.
(3) Rule-based reasoning.
(4) Source code level programming/language extensions.
(5) Model-driven approaches.
(6) Message interception.

There are, however, generic cases where more than one paradigms or patterns are used together in the same approach (e.g. policies and rule enforcement may be included in a service platform). Hence, the above approaches are not completely disjoint and a potential developer may opt for a combination of several techniques, in order to handle context. Moreover, such a categorization poses the additional issue of the system architecture that will be required, in order to support the operation of the service. We chose not to deal with this question and to ignore the distinct architectural layers involved in service provision by focusing on business logic adaptation itself and not on the supporting system. At the heart of every context-aware service, relevant business logic has to be executed in one or more nodes and this logic has to be adapted to context changes. Our interest lies in design and development approaches that facilitate business logic adaptation, especially by decoupling the service logic from the context handling layer.

According to this line of thought, in this survey we focus on the last three approaches since they provide design and development techniques that allow developers to separate the service logic from the context handling layer. Context-aware services that have been developed in this manner can be built on top of dedicated platform solutions or be combined with rule engines; therefore, the selected approaches do not preclude the use of additional approaches from the above list resulting in hybrid approaches. In some cases, the approaches in question follow the Web Service (WS) technologies; this is expected because of WS being a popular trend in the service domain. Furthermore, we concentrate mostly on server-side solutions; therefore, in the majority of the presented approaches herein context adaptation is performed server-side. This does not preclude the possibility to use the same or additional strategies, in order to handle context information on the client side, e.g. in the application segment executing on a mobile device. Taking into account the above the present survey focuses on approaches 4, 5 and 6.

With respect to the other approaches in the list, the first category, i.e. middleware solutions and dedicated service platforms, has been studied by Baldauf et al. (2007). In their survey they present various aspects of context-awareness, e.g. in which ways can data be collected and retrieved, what context models exist, how is context categorized, etc., and focus on the characteristics of the most representative context management architectures. Therefore, we have not included this category in our survey. For more details the reader can refer to the above survey and to the interesting works of Context-awareness sub-structure (CASS) presented by Fahy and Clarke (2004), Service-oriented Context-Aware Middleware (SOCAM) found in Gu et al. (2004) and WildCAT presented by David and Ledoux (2005).

The second category, i.e. ontologies' usage, exploits the principles of the semantic web (Berners-Lee et al., 2001) to produce ontologies that describe context information and its associations and provide means for reasoning and inference. They are relying usually on Resource Description Framework (RDF) (W3C, 2004a) and Web Ontology Language (OWL) (W3C, 2004b) semantic languages and are combined with middleware or frameworks to provide more complete context management. Such an ontology is accompanying the SOCAM architecture, where context is represented in OWL and supports context classification, semantic operations and reasoning on context information focusing on smart home environments (Ying and Fu-yuan, 2006). We have also omitted this category from our survey, because ontologies usually model context information specific to a chosen application domain (i.e. smart home environments in the paper referenced above). Although the study of ontologies is important, it would be more appropriate in a separate survey; its inclusion would extend considerably the length of the paper.

Regarding rule-based reasoning approaches, a rule-based system is a combination of a number of rules and a set of activation conditions for these rules. They are simple in the sense that they offer a similar functionality to if–then–else statements, but in a more elaborated way. They provide a knowledge basis that drives the activation of a behaviour based on the system knowledge that the rules and the activation patterns offer. A significant approach that integrates object-oriented programming and rule-based reasoning has been provided by D'Hondt and Jonckers (2004). Nevertheless, this category has not been analyzed further in the survey, since only few and quite early studies from mainly one research group present respective approaches specific to context management for services (Daniele et al., 2007 and Costa et al., 2008). It will be interesting to observe the solutions to be proposed in this direction in the near future.

With respect to approaches 4, 5 and 6, context can be handled directly at the code level by enriching the business logic of the service with code fragments responsible for performing context manipulation, thus providing the service code with the required adaptive behaviour. Source code level handling approaches usually involve the extension of existing programming languages either at the syntax level or by providing complementary external mechanisms. Additionally, in many occasions, the delivery of a service requires a more complicated software engineering process, which passes through the stages of analysis and design prior to the actual code development for the service. One such process, which relies heavily on modeling, is the Model-Driven Engineering (MDE) paradigm (Schmidt, 2006) aiming at the definition of domain-specific modeling languages, transformations between meta-models and even the semi-automatic or even fully automatic production of executable code. In the development process the focus is given on the platform-independent modeling of the application that drives the transformation to the application code. In this case the question comes to mind regarding how context can be described by appropriate models that can be combined with the respective service models towards the production of an operational service. Finally, context adaptation in services by means of message interception is performed by intercepting the incoming and outgoing messages of a service and modifying them accordingly without affecting either the core service and its functionality or the underlying middleware technology. However, message interception techniques can also be applied to programming languages by intercepting and modifying accordingly the application objects; again without performing changes to the main application.

Wherever possible, taking also into account space restrictions, we try to analyze and evaluate the use of approaches 4, 5 and 6 by means of a simple, but representative example, which is detailed in the following section.

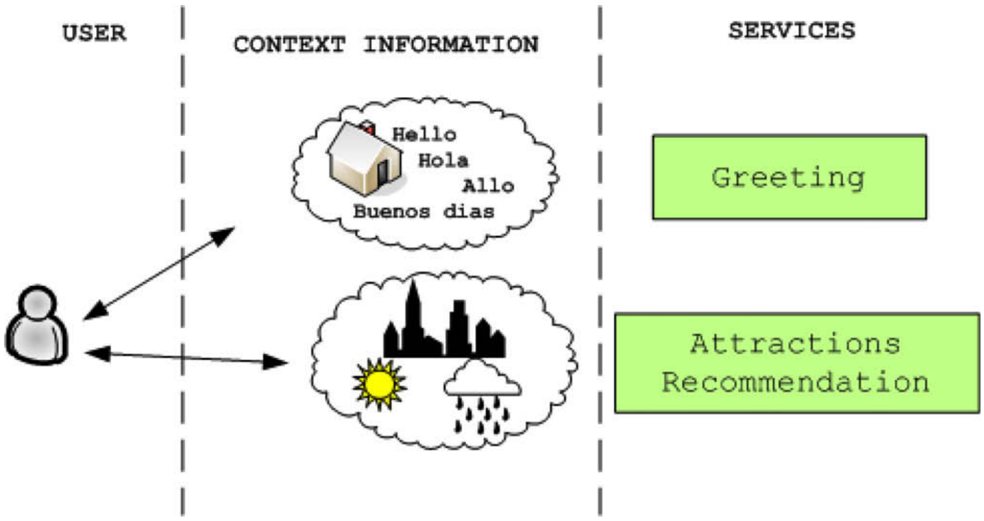


Fig. 1. Tourist service example.

3. Running example

The example used throughout the paper is that of a service mashup. A service mashup is the result of a service composition process that ties together several, possibly independent, services. The resulting mashup is a new service that offers previously non-existent functionality by combining features that were already offered by the component services. In the general case each component service as well as the service mashup may be supported by a different service provider.

The example in this paper is a tourist service that consists of two sub-services (Fig. 1). The first sub-service offers operations for presenting a greeting message to the user based on the following contextual information: his nationality, which determines the language used in the message, the time of day, which determines the greeting scope of the message (morning, evening greeting, etc.) and the current user location, which is used to add the name of the corresponding city in the message. An attractions and activities recommendation forms the second sub-service. This service is based on user preferences values that can be found in a stored user profile (e.g. managed by a database system) and the current weather information (outside temperature, season information, rain likelihood, etc.), which determines whether outdoor activities could be taken into account in the attractions list.

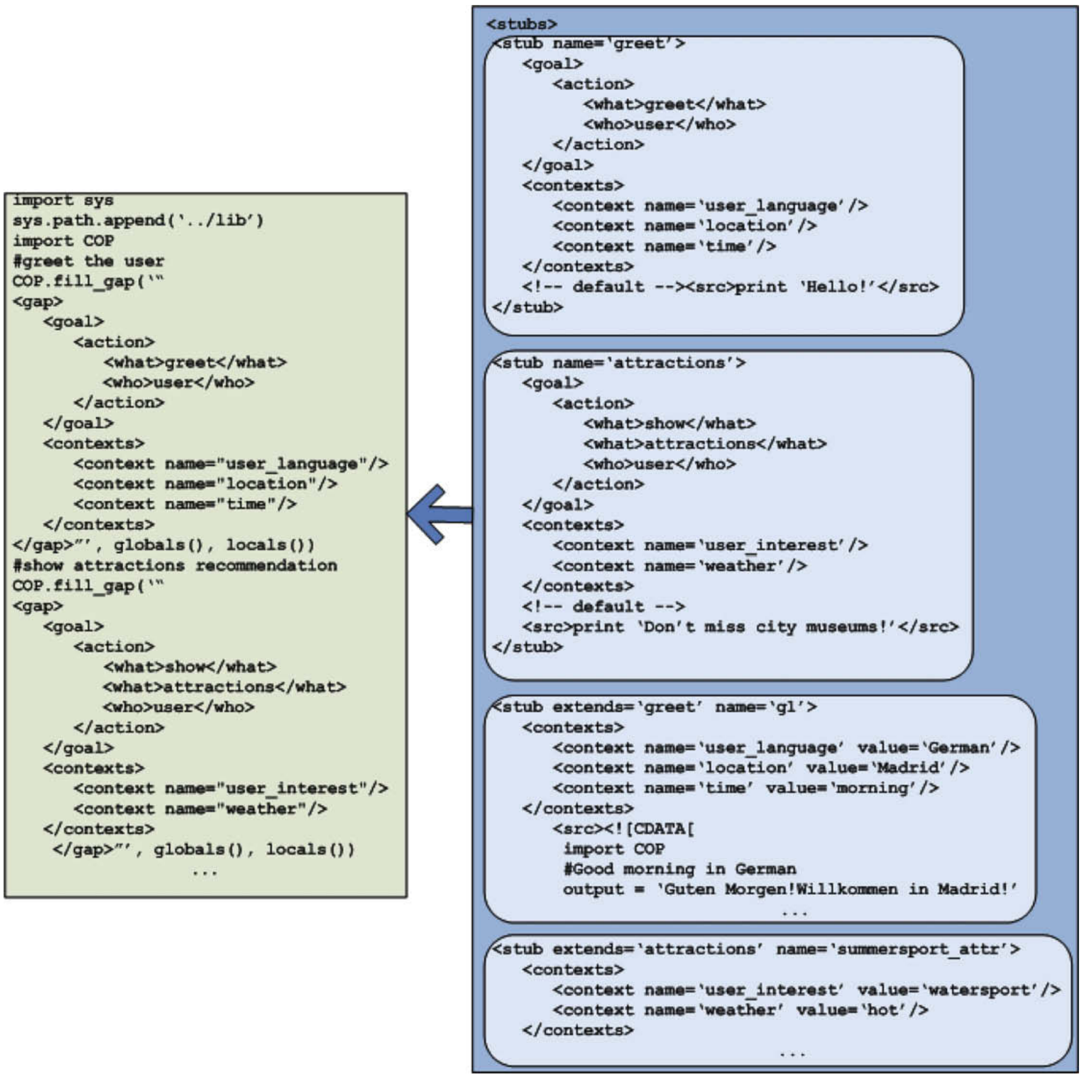
It is a rather simple mashup; nevertheless, there is no loss of generality since the context handling approaches presented in the following sections can be equally applied to more complex services. The tourist service is a typical case of a service offered by a mobile operator. The greeting sub-service could have been developed in-house by the mobile operator, whereas the weather service can be supported by a 3rd party service provider that has a commercial agreement with the operator. The mobile operator owns the user profile and exploits it, in order to execute the greeting service; consequently, it forwards the user's current location (which is known to the mobile network) to the weather service.

4. Source code level approaches

The first significant attempt in source code level context handling was made by Context-Oriented Programming (COP) (Keays and Rakotonirainy, 2003). COP keeps the code skeleton context-free and exploits context-dependent stubs for context manipulation. Context is handled as a first-class construct of the programming language while *goals, open terms* and *stubs* are used in conjunction with context to inject context-related behaviour into the main execution skeleton. Open terms constitute gaps in program's logic that consist of an entity's role (goal), a context information (context) and optionally an event triggering the open term execution. Context-filling is the procedure of selecting the appropriate stub and binding it with the respective program gap, thus providing the desired context-dependent behaviour. This process requires an external entity, named stub repository, from where the relevant code fragments are retrieved by the code skeleton when specific context-related conditions are met. COP has been implemented as an extension to the Python programming language, whose flexible scoping makes it suitable for this purpose. Open terms and stub types are represented in XML and the context-filling is based on these XML representations. It should be noted that COP was not originally directed at context-aware services as they are known today; however, it can be easily applied to the design and development of code for a context-aware service as it can be shown when considering the tourist service example.

Fig. 2 shows the code skeleton in Python where a call to the `fill_gap`() routine triggers the context-filling procedure for the open term. Default stubs are available for both the greeting and attraction services, whereas other defined stubs express more specific context cases; e.g. stub *g1* will be invoked for users in Madrid coming from Germany when it is early in the day. Nevertheless, the implementation of COP in Python is limited, since only statements are regarded as open terms. Moreover, Python is an interpreted language thus it is possible to modify the source code statements once context-filling has taken place and then execute the service without having to re-compile everything. In a different high-level language (e.g., C++, Java) this would not be as easy. In the case of Java this could have been achieved with load-time class transformation of Java class files (Kniesel and Costanza, 2001). Runtime adaptation can also be approached possibly via a specially engineered Java Virtual Machine (JVM) (Cutumisu et al., 2004). Both approaches, however, are far more complex than using an interpreted language such as Python.

Another case of COP is the one discussed in (Costanza and Hirschfeld, 2005; Costanza et al., 2006) where the authors propose the design of context-aware systems following a layered approach. The term "layer" corresponds to a set of partial class and method definitions with specific behaviour. In order to use these layers, code snippets are added to cause the layer activation ("with" construct) or deactivation ("without" construct). This way, the program behaviour can be dynamically modified at runtime in a context-



```
import sys
sys.path.append('../lib')
import COP
#greet the user
COP.fill_gap('"
<gap>
   <goal>
      <action>
         <what>greet</what>
         <who>user</who>
      </action>
   </goal>
   <contexts>
      <context name="user_language"/>
      <context name="location"/>
      <context name="time"/>
   </contexts>
</gap>"', globals(), locals())
#show attractions recommendation
COP.fill_gap('"
<gap>
   <goal>
      <action>
         <what>show</what>
         <what>attractions</what>
         <who>user</who>
      </action>
   </goal>
   <contexts>
      <context name="user_interest"/>
      <context name="weather"/>
   </contexts>
    </gap>"', globals(), locals())
                   ...
```

```
<stubs>
<stub name='greet'>
   <goal>
      <action>
         <what>greet</what>
         <who>user</who>
      </action>
   </goal>
   <contexts>
      <context name='user_language'/>
      <context name='location'/>
      <context name='time'/>
   </contexts>
   <!-- default --><src>print 'Hello!'</src>
</stub>

<stub name='attractions'>
   <goal>
      <action>
         <what>show</what>
         <what>attractions</what>
         <who>user</who>
      </action>
   </goal>
   <contexts>
      <context name='user_interest'/>
      <context name='weather'/>
   </contexts>
   <!-- default -->
   <src>print 'Don't miss city museums!'</src>
</stub>

<stub extends='greet' name='gl'>
   <contexts>
      <context name='user_language' value='German'/>
      <context name='location' value='Madrid'/>
      <context name='time' value='morning'/>
   </contexts>
      <src><![CDATA[
       import COP
       #Good morning in German
       output = 'Guten Morgen!Willkommen in Madrid!'
                                ...

<stub extends='attractions' name='summersport_attr'>
   <contexts>
      <context name='user_interest' value='watersport'/>
      <context name='weather' value='hot'/>
   </contexts>
                                ...
```

Fig. 2. Service skeleton with open terms on the left, matching code stubs on the right.

aware fashion. The system relies on a separate infrastructure to capture context information from sensors and transport it to where the code is running. According to the COP paradigm, context dependencies are kept separate from the base program definition.

Instead of using an external mechanism as in the previous approach (through the stub repository), the context-filling process takes place in the programming language provided that the necessary context information has been retrieved through an appropriate infrastructure and is ready to be used. The approach is presented as an extension of the Common Lisp Object System (CLOS) named ContextL. In more recent work (Hirschfeld et al., 2008) the same layered mechanism was generalised, in order to be combined with other programming languages like Java and Squeak/Smalltalk. For the tourist example different layers need to be defined to express all adaptation cases. Some of them are illustrated in Fig. 3 in the Java prototype of the approach, ContextJ*, e.g. a GoodWeather layer that results in the inclusion of outdoor activities in the recommendation list when activated.

The above COP and layer activation solutions contain characteristics of Feature-Oriented Programming (FOP), where the application can be seen as the collaboration of individual features expressed in requirements, design and implementation, and are sufficient for addressing heterogeneous aspects in the sense that different code fragments are applied to different program parts. For a more complex skeleton, the same context information would be required in different parts of a service and would trigger the invocation of additional operations in the same or even in additional services. This way, context handling becomes a concern that spans across several service units essentially crosscutting into main service execution. A programming paradigm aiming at handling such crosscutting concerns (referred to as *aspects*) is Aspect-Oriented Programming (AOP) (Elrad et al., 2001). AOP has been implemented as an extension to popular programming languages (AspectJ, AspectC++, etc.) where it introduces the notions of *joinpoints* that define the points of the program where aspects can introduce additional behaviour, *pointcuts* that define expressions to detect joinpoints and pieces of *advice* that describe the code to be applied on joinpoints. Aspects themselves consist of a pointcut and the respective advice. In contrast to FOP, AOP is ideal for homogeneous crosscutting concerns meaning that the same code is applied to different points (Apel et al., 2006). Using the AOP paradigm context information can be handled through aspects that interrupt the main service execution, in order to achieve service adaptation to context in a manner similar to COP. This can be an interesting approach for the service engineer especially when combined with methodologies like the Theme approach (Baniassad and Clarke, 2004) that helps in identifying as early as possible which system functionalities consti-

```
//prototype implementation library
import static be.ac.vu.prog.contextj.ContextJ.*;

//File: Layers.java
public class Layers {
  public static final Layer Location = new Layer();
  public static final Layer Spanish = new Layer();
  . . .

  public static final Layer GoodWeather = new Layer();
  . . .
}

//File: IAttractionsService.java
public interface IAttractionsService {
  public List recommendAttractions();
  public List indoorAttractions();
  public List outdoorAttractions();
}
//File: AttractionsService.java
public class AttractionsService implements
iAttractionsService {
  private LayerDefinitions<IAttractionsService> layers =
    new LayerDefinitions<IAttractionsService>();

  public List recommendAttractions(){
    return layers.select().recommendAttractions();
  }
  public List indoorAttractions() { ... }
  public List outdoorAttractions() { ... }

  {
    layers.define(RootLayer,
         new IAttractionsService () {
             public List recommendAttractions() {
                 return indoorAttractions();
             }
         }
    );

    layers.define(Layers.GoodWeather,
         new IAttractionsService () {
             public List recommendAttractions() {
                return
layers.next(this).addAll(outdoorAttractions());
             }
         }
    ); . . . } }
```

```
//File: IGreetingService.java
public interface IGreetingservice {
  public String greetUser();
}

//File: GreetingService.java
public class Greetingservice implements
iGreetingService{
  String userName; String cityName;

  private LayerDefinitions<IGreetingService> layers =
    new LayerDefinitions<IGreetingService>();

  public String greetUser() {
    return layers.select().greetUser();
  }

  {
     layers.define(RootLayer,
          new IGreetingService() {
              public String greetUser() {
                  return "Hello " + userName;
              }
           }
     );

    layers.define(Layers.Location,
       new IGreetingService() {
         public String greetUser() {
return layers.next(this)+".Welcome in " + cityName;
         }
       }
    );

    layers.define(Layers.Spanish,
       new IGreetingService() {
          public String greetUser() {
              return "Hola " + userName;
          }
       }
    );
  . . .
  }
}
```

Fig. 3. Layer definitions for different cases for the Attractions (left) and Greeting (right) services.

tute aspects (*crosscutting* themes) and which base functionalities (*base* themes).

Tanter et al. (2006) go one step further by proposing the adaptation of aspects to context in what is called *context-aware aspects*. This means that the use of aspects is driven by context; a certain aspect may or may not be executed depending on its context of use. The supplementary assumptions made in this approach is that the contexts guiding the aspect invocation can have parameters (e.g. time and location as in the greeting service) and that they can be combined with other contexts. However, what is more important is that service behaviour can be affected not only by current context values but also by past ones. This time-related property of context-aware aspects is referred to as *context snapshotting*.

Context-aware aspects are supported by a framework that has been implemented in Reflex (Tanter and Noye, 2005), an open extension of Java, which provides the necessary constructs for facilitating work on different AOP concepts. Reflex allows first-class pointcuts for dynamic crosscutting. The idea is that *hooksets* of Reflex – acting similarly to AOP pointcuts – can specify conditions of when to send a specific message to a metaobject, achieving this way context-based execution of code fragments or operations. Contexts themselves are represented as objects that manifest a method returning a Boolean value indicating their activation state. Several dedicated language constructs are available in Reflex to express and show activation conditions based either on current or past contexts, such as the abstract classes `CtxActive` and `SnapshotCtxActive`. Returning to our example we can think of a service containing a default message in English and some aspects with customized greetings that depend on user's language (e.g. if the user is from a Castellan speaking city of Spain, the Spanish greeting is invoked). Contexts in this case can also use the local time as a parameter to further adapt the greeting message. Therefore, the time parameter specifies the context state since it does not remain the same for different invocations of the same service (the same applies for location). Similarly, the attractions service can return a default list of all indoor attractions at user's location that can be modified to include outdoor attractions in case of good weather conditions. Fig. 4 contains the definitions of the `SpanishGreetingCtx` and the `GoodWeatherCtx` context that extend the `Context` class to express more specific conditions, as well as the pointcuts indicating the execution of the respective aspects.

An example well suited to demonstrate the *context snapshotting* feature proposed in context-aware aspects is that of the online shop. We assume that a new shopping basket object is created in an online shop each time the user logs in and that at each point of time various offers may be active (active contexts). By exploiting past contexts, special ratings can be offered to the customers purchasing products depending on the context snapshot at login. In other words, a context snapshot is taken during login and the rates offered do not depend on the active discounts at the time of the

```
//time and city specify the context state
Class SpanishGreetingState extends ContextState {
   Time time;
   String city;
   Time getLocalTime() {
      . . .
      return time;
   }
   String getUserCity() {
      . . .
      return city;
   }
}

Class SpanishGreetingCtx extends Context {
//The control flow of the hookset matching requests to greetUser
//operation is exposed
   CFlow cf = CFlowFactory.get(new Hookset(MsgReceive.class,
         new NameCS("WebServiceRequest", true),
         new  NameOS(("greetUser", true)));

   Time time; //with getTime and setTime methods
   String city; // with getCity and setCity methods
   boolean active() {
      return cf.in();
    }

   ContextState getState() {
      if (!cf.in()) return null;
            return new SpanishGreetingState(time, city);
          }
}

Class GoodWeatherCtx extends Context {
   CFlow cf = CFlowFactory.get(new Hookset(MsgReceive.class,
         new NameCS("WebServiceRequest", true),
         new  NameOS(("showAttractions", true)));

   boolean active() {
      return cf.in();
    }
//Relevant pointcuts for our example: they detect the joinpoints
//of calls to greetUser and showAttractions that match the
//SpanishGreetingCtx and GoodWeatherCtx accordingly
pointcut userLoc(): execution(*
*..*.greetUser(..))&&inContext(SpanishGreetingCtx(time));

pointcut weather(): execution(*
     *..*.showAttractions(..))&&inContext(GoodWeatherCtx());
...
```

Fig. 4. Defining the code for the Spanish greeting and good weather context using Context-aware Aspects.

purchase (current context), but on the special rates available when the user logged in (past context). Context snapshotting can be further enhanced by combining the service with a rule engine that can reason over the set of past and current contexts. This is the approach employing *Hybrid Aspects* that combine features from AOP and rule-based reasoning followed by D'Hondt and Jonckers (2004) that can be adapted for context-aware development.

Similar history features are supported in the pointcut language History-based Aspects using LOgic (HALO) (Herzeel et al., 2007) implemented for Lisp that introduces pointcuts on program execution history expressed as joinpoint history and in (Allan et al., 2005) where matching for traces (event patterns recognized in code execution) was added as an extension to the AspectJ language.

A different programming technique that extends object-oriented programming – called Isotope Programming Model (IPM) – is presented by Saiyu et al. (2007). IPM states that object-oriented programming can be changed so that object behaviour is not static, but rather adaptable to different contexts. In that sense the code is changed depending on its context of use and is kept separate from the environmental information. Objects are defined by a number of attributes and default methods as in the traditional object-oriented model (*main element*) and a number of *isotope* elements. Isotope elements describe the code structure in different environments. They are comprised of a context block that specifies the execution conditions and a behaviour part that contains the isotope element's methods. The most recent and most appropriate isotope element is selected in each execution. If no such element exists, the default method – if present – is called. New isotope elements can be added or existing ones can be removed as the environments change. The case of IPM is not demonstrated here further through an example due to the lack of an extended description of the mechanism constructs in the published version of the approach.

5. Model-driven approaches

In the latest years model-driven engineering techniques are being applied with success by many developers. That is the reason why a significant number of recent context management approaches focuses on model-driven development of context-aware applications and services. The modeling language used in the majority of cases is the Unified Modeling Language (UML) (OMG, 2007). UML is the most widely adopted language for object modeling. It allows the introduction of profiles as extensions to the UML metamodel metaclasses. Generally, profiles and metamodels are techniques used for extending the semantics of a language. This way the model representation can be adapted for use in different domains of interest, facilitating at the same time the model transformation to code, based on a number of profile stereotypes and tags imported in the model. These techniques can also be exploited in the framework of context-aware services. An overview of context models – not limited to UML notation – is available from Strang and Linnhoff-Popien (2004).

One of the first approaches towards modeling the interaction between context and service for web services is found in ContextUML (Sheng and Benatallah, 2005). ContextUML is a UML metamodel, which extends the existing UML syntax by introducing appropriate artifacts, in order to enable the creation of context-aware service models. ContextUML-derived models consist of class diagrams where classes correspond to context as well as service constructs, while UML dependency and association relationships express the interaction between the context classes and the respective service ones. ContextUML can be used for expressing (i) parameter injection that is the population of operation parameters based on context and (ii) response manipulation that is manipulation/modification of service responses based on context. The work in ContextUML has been extended by Prezerakos et al. (2007) towards the direction of further decoupling context model from service model design. Apart from class diagrams, UML activity diagrams are used for modeling the core service logic flow in conjunction with MDE transformation techniques and AOP. Indeed many benefits can be gained by combining MDE with AOP as urged also by Carton et al. (2007).

Yet another UML metamodel has been proposed by Ayed and Berbers (2006). This UML metamodel supports structural, architectural and behavioural adaptations of service design based on context values. Structural adaptations are concerned with the extension of classes with additional attributes and methods, whereas architectural adaptations are related with the optional instantiation of objects. Behavioural adaptations extend the UML sequence diagram, in order to support the inclusion of several sequences that can be optionally activated. A UML profile towards Model-driven Development of context-aware applications similar to ContextUML can be found in (Grassi and Sindico, 2007). In this profile context is categorized in (i) state-based context that characterizes the current situation of an entity and (ii) event-based context that represents changes in an entity's state. Constraints are used on both context types to trigger various invocations: state constraints refer to specific points in time, while event constraints exploit historical data of context events. Similarly to the previous metamodel, context information can be either used to be mapped to specific values or modify the structure or the behaviour of the application. *Monitor* elements are responsible for retrieving various context information, while *Adapters* adapt the application based on this information e.g. by causing additional actions to be performed, etc. This UML model is mapped to AspectJ code, where Adapters and Monitors are implemented as aspects.

Modeling techniques can also be used during the software requirements stage of context-aware applications. This is the case of the Context-Oriented Domain Analysis (CODA) by Desmet et al. (2007), which is a systematic approach for gathering requirements of context-aware systems using respective models and can be subsequently mapped to decision tables.

Another important issue of context-awareness is privacy guarantees needed to protect sensitive user data and its dissemination. Privacy and quality issues have been taken into account in the Context Modeling Profile (CMP) of Simons (2007). The profile consists of a number of stereotypes used for representing context items, associations between them and with the context source and context quality (which can be *userprovided*, *sensed* and *derived*). Context validity shows how frequently the context changes, whereas access associations express who has access to the specific context value (e.g. all, specific groups, etc.). The profile is combined with a number of constraints expressing restrictions on context representation expressed in the Object Constraint Language (OCL) (OMG, 2006). The association between context and privacy is also studied by Henricksen et al. (2005). The authors extend their Context Modeling Language (CML) (Henricksen et al., 2002) to include privacy preferences. CML is not based on UML, but on Object-Role Modeling (ORM) and describes context information in terms of object and fact types. Objects represent entities (e.g. person, device), while facts refer to associations and constraints between these objects (e.g. person X "is using" device Y). Ownership is defined for facts and objects; three classes of ownership are used for objects, which can be overridden by fact's ownership, i.e. rules of ownership defined for facts. The ownership is combined with user privacy preferences over context elements. Preferences are expressed in pairs of scope – referring to the context where the preference is to be applied – and scoring expressions – stating whether the context value can be disclosed or not.

Service and context models created using UML and other metamodels can be converted to high-level language source code by

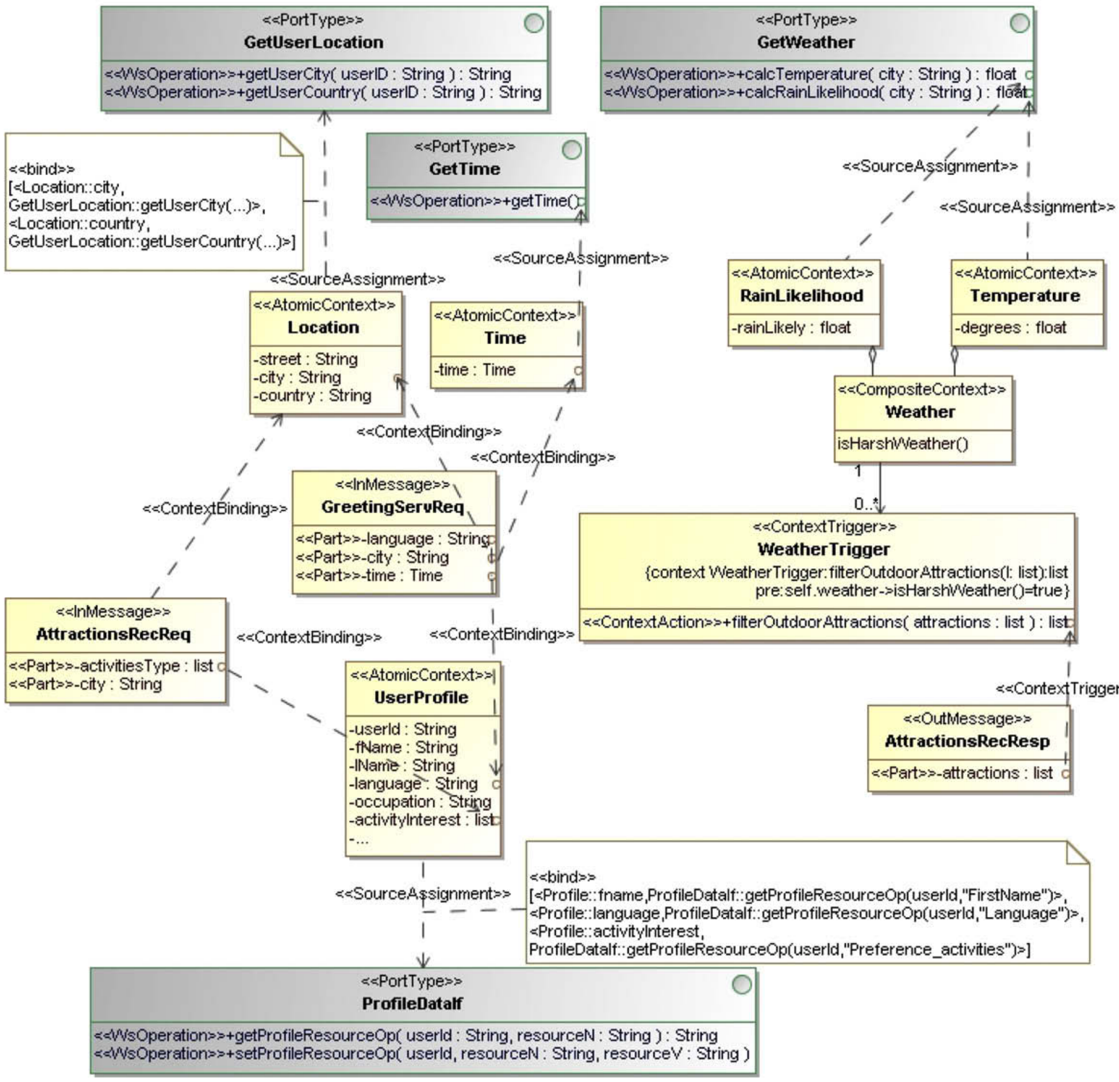

<<PortType>> GetUserLocation
<<WsOperation>>+getUserCity(userID : String) : String
<<WsOperation>>+getUserCountry(userID : String) : String
<<PortType>> GetWeather
<<WsOperation>>+calcTemperature(city : String) : float
<<WsOperation>>+calcRainLikelihood(city : String) : float
<<PortType>> GetTime
<<WsOperation>>+getTime()
<<bind>> [<Location::city, GetUserLocation::getUserCity(...)>, <Location::country, GetUserLocation::getUserCountry(...)>]
<<SourceAssignment>>
<<AtomicContext>> Location -street : String -city : String -country : String
<<AtomicContext>> Time -time : Time
<<AtomicContext>> RainLikelihood -rainLikely : float
<<AtomicContext>> Temperature -degrees : float
<<CompositeContext>> Weather isHarshWeather()
<<ContextBinding>>
<<InMessage>> GreetingServReq <<Part>>-language : String <<Part>>-city : String <<Part>>-time : Time
<<ContextTrigger>> WeatherTrigger {context WeatherTrigger::filterOutdoorAttractions(l: list):list pre:self.weather->isHarshWeather()=true}
<<ContextAction>>+filterOutdoorAttractions(attractions : list) : list
<<InMessage>> AttractionsRecReq <<Part>>-activitiesType : list <<Part>>-city : String
<<ContextTriggering>>
<<AtomicContext>> UserProfile -userId : String -fName : String -lName : String -language : String -occupation : String -activityInterest : list -...
<<OutMessage>> AttractionsRecResp <<Part>>-attractions : list
<<bind>> [<Profile::fname,ProfileDataIf::getProfileResourceOp(userId,"FirstName")>, <Profile::language,ProfileDataIf::getProfileResourceOp(userId,"Language")>, <Profile::activityInterest, ProfileDataIf::getProfileResourceOp(userId,"Preference_activities")>]
<<PortType>> ProfileDataIf
<<WsOperation>>+getProfileResourceOp(userId : String, resourceN : String) : String
<<WsOperation>>+setProfileResourceOp(userId, resourceN : String, resourceV : String)
1
0..*

Fig. 5. Context model and dependencies for the tourist example.

means of either commercial or open source tools responsible for performing the transformation from model to code (e.g. AndroMDA[2], openMDX[3]).

Returning to the tourist service example, the dependencies between the service and the context model in ContextUML notation representing UML modeling are shown in Fig. 5. The web service interfaces are depicted using the <<PortType>> stereotype. Some of these WSs are also used for the context information retrieval (<<SourceAssignment>> dependency), whereas the context dependencies are expressed with <<ContextBinding>> and <<ContextTriggering>>. It would be too optimistic to assert that the process of code generation from models can become completely automatic and that the developer's role lies only on service design. But overall, the development effort can be reduced significantly, since the code structure is generated along with a useful amount of code and the developer may need to intervene only in specific code sections. Corresponding support mechanisms at the code level also have to be present, i.e. the target platform specification needs to be available. The fact that the service author does not have to be fully aware of how those mechanisms should be implemented can be perceived by many developers as an advantage.

6. Message interception

Solutions based on message interception are interesting because they can be applied to WS, which constitute the most popular technology for service-oriented applications on the web and are the most promising candidate also for pervasive environments mainly due to their interoperability properties. Of course it is not limited only to web services, but can be exploited in the framework of other middleware technologies, like Common Object Request Broker Architecture (CORBA) or Simple Middleware Independent LayEr (SMILE) by Salsano et al. (2008) or even in interception of objects and input and output messages in programming languages.

In the latter case of programming languages, less recent approaches have presented the interception of messages exchanged between objects as in the language independent composition filters (CF) approach by Aksit et al. (1993) describing how messages pass through filters where they are processed until they are discarded or dispatched, i.e. activated again. A mechanism more specific to message modification is supported by the work in (Bosch, 1995), where the Layered Object Model (LayOM) implemented in

[2] http://www.andromda.org
[3] http://www.openmdx.org/

```
<!--Greeting Service-->
<soapenv:Envelope>
  ...
 <soapenv:Header>
   <Context xmlns="http://sg.fmi.uni-
passau.de/context">
      <Language>German</Language>
      <Location>
        <address>
          <addressLine keyname="Street"
keyValue="60">San Miguel 18</addressLine>
                          <addressLine keyname="City"
keyValue="40">Madrid</addressLine>
     <addressLine keyname="Country"
keyValue="20">Spain</addressLine>
         </address>
      </Location>
      <Time>
        <localTime>19:00</localTime>
      </Time>
   </Context>
 </soapenv:Header>
 <soapenv:Body>
...
 </soapenv:Body>
</soapenv:Envelope>
```

```
<!--Attractions Service-->
<soapenv:Envelope>
 ...
 <soapenv:Header>
   <Context xmlns="http://sg.fmi.uni-
passau.de/context">
      <ActivityInterest>
         <pref1>History</pref1>
         <pref2>WaterSports</pref2>
      </ActivityInterest>
      <Weather>
        <localTemp>27C</localTemp>
        <rainLikely>0.005</rainLikely>
      </Weather >
   </Context>
 </soapenv:Header>
 <soapenv:Body>
...
 </soapenv:Body>
</soapenv:Envelope>
```

Fig. 6. Examples of SOAP Envelopes with context extensions for location, time, user preferences and weather.

C++ adds layers that perform different actions on the messages. Both these approaches constitute interesting attempts and could be exploited – through some necessary adaptation – in message modification for context-awareness support in code execution.

In the web services case, service requests and responses travel back and forth between WS clients and web services encapsulated in envelopes formulated according to the Simple Object Access Protocol (SOAP). Context information related to a specific service can be handled by (i) defining namespaces to allow SOAP messages to carry context information in the SOAP headers and (ii) intercepting these messages on the way and process the embedded context information, leaving the context handling – at a great extent – outside web services themselves (Keidl and Kemper, 2004). More specifically, all types of context information can be included as SOAP extensions in message headers. The extensions include context blocks containing a unique *context identifier* and a *context value*. Considering again our tourist example the corresponding SOAP messages for each service are depicted in Fig. 6. The language, location and address for the greeting and the weather and activity interest information for the attractions service are visible in the message headers.

Context manipulation in this framework is automatically performed through handlers either included as context plugins in the framework or available remotely as dedicated web services. The handlers intercept SOAP messages exchanged during service execution and modify context extensions accordingly. This processing of context information can be performed either on the client or on the web service side using a specific Context API. In order to specify who modifies which information, context processing instructions are provided, which include references to services responsible for context processing and additional information on how the processing is performed. This approach provides for substantial flexibility in handling context processing independently from the core service logic. The context exchange and processing mechanisms described can be exploited in the WS code to perform the necessary actions leading to context-aware WS provision.

A message interception approach for web service context-awareness is also proposed by Prezerakos et al. (2007), where the SOAP messages are modified based on advices applied during the message interceptor execution. The respective aspects can be turned on and off taking into account the current adaptation needs. In (Kapitsaki et al., 2008) a handler for the Axis2 web service framework[4] has been implemented for the interception of SOAP messages. The handler is responsible for the invocation of plugins loaded on runtime based on an XML configuration file (handler.xml) that associates web service operations with specific plugins. Plugins - separated to *inPlugins* for the modification of requests and *outPlugins* for the responses - retrieve the necessary context information by communicating with context sources exposed as web services and then perform the message modification based on this information. The modification may include request parameter replacement, selection of the correct operation to be performed based on context values, etc. Assuming the two sub-services (Fig. 6) and the corresponding context sources are available as WSs the configuration file for the tourist service example would be as depicted in Fig. 7. The greeting service is associated with one InPlugin that will modify the SOAP request, whereas the attractions recommendation service is associated with two plugins, one for each message direction.

A generic model with inclusion of context information in messages not bound to specific technologies is proposed by the abstract Context-Dependent Role Model (CDR) presented in (Vallejos et al., 2007). In this approach, actors, which can be defined as interacting nodes in a distributed environment, react to exchanged messages by adopting a certain role at each case. This is performed through the context-dependent role selector, a dedicated entity within the actor that selects the appropriate role based on context information regarding the message sender and receiver. Context information is included in the messages by the context ref-

[4] http://ws.apache.org/axis2/

```xml
<?xml version="1.0" encoding="UTF-8"?>
<han:plugins xmlns:handler="http://www.contextplugins.org/handler"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://www.contextplugins.org/handler handler.xsd ">
  <han:plugin id="1" pckg="plugins.greeting" classname="greetingInPlugin" jarfile="plugingreeting.jar"/>
  <han:plugin id="2" pckg="plugins.attractions" classname="attractionsInPlugin"
jarfile="pluginattractions.jar"/>
  <han:plugin id="3" pckg="plugins.attractions" classname="attractionsOutPlugin"
jarfile="pluginattractions.jar"/>
  <han:association>
    <han:serviceEnd name="Greeting" operation="greet"/>
    <han:pluginEnd id="1" direction="in"/>
  </han:association>
  <han:association>
    <han:serviceEnd name="Attractions" operation="attractionsRec"/>
    <han:pluginEnd id="2" direction="in"/>
  </han:association>
  <han:association>
    <han:serviceEnd name=" Attractions" operation=" attractionsRec"/>
    <han:pluginEnd id="3" direction="out"/>
  </han:association>
</han:plugins>
```

Fig. 7. Handler.xml configuration file for the tourist example.

erence proxy. The context reference captures the outgoing messages and may add context data to the message before sending it to the receiver based on rules that indicate in which conditions context information is added.

The use of web services in the management of context information is not limited to the presented approaches. In (Arruda et al., 2003) the Context Kernel web service has been implemented to allow applications to store and exchange context information. Other web services can exploit the retrieval and storage mechanism of the context kernel to gain context information and integrate it in their functionality. Nevertheless, privacy and security issues emerge in the above cases by having context information travel back and forth between different connection points and certainly further work in this direction is highly desirable.

7. Comparison/discussion

From the study of the field of context-awareness and the different issues addressed in the literature we have concluded that useful requirements for efficient context-aware service engineering are:

- *Decoupling from service/application logic*: separation between the management of the main service engineering process, and the context adaptation mechanism.
- *Provision of means for combination and alterations between different models:* since user and provider needs change, the development of context-aware services should be flexible enough to allow the replacement of context or even service models and not be tightly coupled with specific models.
- *Handling of past and composite context data:* historic data needs to be taken into account as well, whereas aggregated and derived information coming from simple context values should also be supported.
- *Support for user data protection:* privacy guarantees need to be taken into account in the sensitive context-aware world.

In this section we show in which degree the above approaches fulfill these main requirements based on the conclusions we have drawn through our survey study. For this reason a summarized presentation of the most representative context-aware approaches in service engineering is illustrated in Table 1.

We observe that all approaches succeed in decoupling the context management from the business logic in a higher or lower degree. Context representation models can be found mainly in model-based approaches, where the use of different models by exploiting generic purpose or domain-specific modeling languages is possible. However, in code-based approaches the model is almost integrated in the approach, depends on the programming language used and cannot be seen independently. The same is the case for message interception techniques. Unfortunately the majority of approaches do not deal with historical and privacy issues, which are important aspects for this kind of services. Different implementation languages or underlying frameworks and middleware are adopted in each approach, but in most cases a certain degree of independence is maintained for the service development (i.e. the adaptation takes place without affecting the underlying middleware technology). The adaptation may take place on either side (client or provider), whereas there are also cases where both are seen as equivalent actors (this happens for example in the context-dependent role model).

As a second step we attempt a more generic comparison of the solutions categories, since an important question for the developer of context-aware services is whether there are specific advantages and disadvantages that would advocate the use of a specific approach for certain types of services. In order to point out the strengths and weaknesses of each approach, the approaches in question are compared in Table 2 according to the following criteria that are meaningful for the service developer when choosing a service engineering approach. These criteria were selected from software engineering evaluation criteria used in other studies (Kan, 2002), as well as from our own experience:

Table 1
A summary with the basic Characteristics of the presented Approaches.

	Context model	Decoupling from business logic	Privacy issues support	Historical data support	Underlying framework/ Implementation	Context adaptation side
COP (Keays and Rakotonirainy, 2003)	Impl.-based	Partial	No	No	Python	Server
Layer activation (Hirschfeld et al., 2008)	Impl.-Based	Partial	No	No	CLOS or Java or Squeak/ Smalltalk	Server
Context-aware aspects (Tanter et al., 2006)	Impl.-based	Partial	No	Yes	Reflex	Server
IPM (Saiyu et al., 2007)	Impl.-based	Partial	No	No	OOP	Server
Hybrid aspects (D'Hondt and Jonckers 2004)	Impl.-based	Yes	No	Partial	AspectS	Unspecified
ContextUML+Aspects (Sheng and Benatallah, 2005), (Prezerakos et al., 2007)	UML	Yes	No	No	Web services	Client
Grassi development (Grassi and Sindico, 2007)	UML	Yes	No	Yes	AspectJ	Unspecified
CML, CMP(Simons, 2007), (Henricksen et al., 2002)	ORM, UML	–	Yes	–	–	–
Towards adaptable web services (Keidl and Kemper, 2004)	UDDI tModel	Yes	No	No	Web services	Both
Handler for web services (Kapitsaki et al., 2008)	Impl.-based	Yes	No	No	Web services (Axis2)	Server
Context-dependent Role model (Vallejos et al., 2007)	Impl.-based	Partial	Yes	No	AmbientTalk	Both

Table 2
Evaluation Criteria for the Context Handling Categories.

	Flexibility	Refactoring ability	Ease of use	Existing system adaptation	Context Model Type	Decoupling from business logic
Source code level handling	High	Limited	Medium	Depends on existing system architecture	OOP-based (including XML-based)	Depends on programming language
Model-based approaches	Medium	High (assuming code re-generation)	Medium	Low (not possible if the system has not been modeled)	UML/MOF Models (mainly)	High
Message interception	Medium	High	High	Medium	External or embedded in messages	High

- *Flexibility*: The degrees of freedom that the context adaptation mechanism provides to the developer.
- *Refactoring ability:* How easy is it to redesign the service, if there are modifications in either the business logic or the context handling logic?
- *Ease of use*: How easy is it for a developer, not very familiar with the specific context adaptation mechanism, to use it?
- *Existing system adaptation:* How easy would it be to apply the specific context adaptation mechanism to an existing system?
- *Context model type*: Is the mechanism bound to a specific context metamodel?
- *Decoupling from business logic:* What is the degree of independence between context information and service logic?

Source code level handling gives more freedom and hence flexibility to the service developer that can implement different kinds of adaptation to context, based on the requirements of the application under development, but inherits the strengths and limitations of the programming language used regarding the other criteria. Moreover, it is quite difficult for a developer to apply these principles in his/her programming language of choice since some features (e.g. aspects) may not be supported in that language. In all cases, a certain degree of familiarization with the adopted programming language is necessary.

Model-driven approaches profit from the advantages of MDE, but require an overall and coherent system modeling that has to be kept updated all times, where the service logic and context models can be introduced. This is not a drawback per se but MDE is best suited for developers with the ability to move between models and from model to code and vice versa without problems. On the other hand, message interception approaches intervene less in the developer's work with respect to the business logic, but require several system modifications in terms of generating and consuming messages to work properly. Furthermore, the message interception mechanism needs to be incorporated in the system; therefore, message interception works better in the presence of a middleware platform where such mechanisms accompany the platform by default.

Although we have divided the approaches in the above main categories, various aspects of context-aware service support can be used at the same time leading to composite approaches, e.g. a model-based approach can be used to generate code that targets a system that relies on message interception. Combination of these aspects into more comprehensive solutions is a very interesting area for future work. Currently there is some interesting work under way in the Java community regarding lightweight enterprise frameworks, such as Spring[5], and languages dedicated to services, such as ServiceJ (Labey et al., 2007), that can be used for context-aware service creation. Nevertheless, we still have a great distance to cover, since all approaches fulfill partially the requirements mentioned in the beginning of the section but none fulfills them all. Moreover, historical and aggregated data as well as privacy issues are only addressed in a small subset of the approaches.

[5] http://www.springframework.org/

8. Conclusions and emerging issues

In this paper we presented a survey on methodologies and solutions for context-aware service engineering. We have described a variety of proposed solutions based on the three main categories that we considered of special interest to service engineering and have not been presented in the literature extensively, namely source code level handling, model-based and message interception approaches. A comparison of the presented work results has exposed the major advantages and disadvantages of each case.

Our main conclusion is that the context handling should be left – if possible – completely outside the service itself in the sense that service logic code should be distinct from the code that performs context adaptation. Nevertheless, services and service requesters must be unaware of the existence of such mechanism and the context information must be available and up to date any time. Furthermore, we find model-based approaches more suitable for service engineering in general, since they profit from the benefits of the MDE paradigm that should not be neglected. Their combination with the novel ideas presented at the implementation level and basically in code-based approaches would lead to interesting results. Our ongoing work is towards the direction of providing a complete context-aware service engineering methodology exploiting web services using MDE techniques and message interception. We are focusing on the specification of the transformation procedures and the complete independence between the business logic and the context mechanisms.

A general observation is that the service engineering community still lacks a set of universally accepted basic design and development principles that can lead to a uniform approach towards efficient context-aware service development. Moreover, the issue of privacy and security is of vital importance and needs to be properly addressed. Indeed, context information and management is related to sensitive user data, such as personal information, current activity, ongoing and past transactions, etc. that should not be revealed unless the end-user has expressed such preference. Unfortunately, privacy is not adequately - or not at all - addressed by the majority of the approaches. Combination with Role Based Access Control (RBAC) techniques could prove useful (Ni et al., 2007).It is our belief that the focus of the future work on context solutions should also be put towards this direction.

Acknowledgements

Georgia Kapitsaki is supported by a Greek State Scholarships Foundation grant. This work has been partially supported by the European Union co-funded IST project Simple Mobile Services (SMS) (http://www.ist-sms.org).

References

Aksit, M., Wakita, K., Bosch, J., Bergmans, L., Yonezawa, A., 1993. Abstracting object interactions using composition filters. Proceedings of the Workshop on Object-Based Distributed Programming 791, 152–184.

Allan, C., Avgustinov, P., Christensen, A.S., Hendren, L., Kuzins, S., Lhoták, O., Moor, O.d., Sereni, D., Sittampalam, G., Tibble, J., 2005, Adding trace matching with free variables to AspectJ. In: Proceedings of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming, Systems, Languages, and Applications, San Diego, CA, USA, pp. 345–364.

Apel, S., Leich, T., Saake, G., 2006. Aspectual mixin layers: aspects and features in concert. In: Proceedings of the 28th International Conference on Software Engineering, Shanghai, China, pp. 122–131.

Ayed, D., Berbers, Y., 2006. UML profile for the design of a platform-independent context-aware applications. In: Proceedings of the First Workshop on Model Driven Development for Middleware (MODDM '06), Melbourne, Australia, pp. 1–5.

Arruda Jr., C.R.E., Neto, R.B., Pimentel, M., da, G., 2003. Open context-aware storage as a web service. International Conference on Distributed Systems Platforms and Open Distributed Processing. Rio de Janeiro, Brazil. 81–87.

Baldauf, M., Dustdar, S., Rosenberg, F., 2007. A survey on context-aware systems. International Journal of Ad Hoc and Ubiquitous Computing 2 (4), 263–277.

Baniassad, E., Clarke, S., 2004. Theme: an approach for aspect-oriented analysis and design. Proceedings of the International Conference on Software Engineering (ICSE'04), 158–167.

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. Scientific American, 34–43.

Bosch, J., 1995. Language support for design patterns. Research Report 11, 1103–1581.

Carton, A., Clarke, S., Senart, A., Cahill, V., 2007. Aspect-oriented model-driven development for mobile context-aware computing. In: Proceedings of the First International Workshop on Software Engineering for Pervasive Computing Applications, Systems, and Environments, (SEPCASE '07), p. 191.

Chen, G., Kotz, D., 2000. A Survey of Context-Aware Mobile Computing Research. Technical Report TR2000-381, Dartmouth College.

Costanza, P., Hirschfeld, R., 2005. Language constructs for context-oriented programming: an overview of contextL. In: Proceedings of the 2005 Conference on Dynamic Languages Symposium, San Diego, California, pp. 1–10.

Costanza, P., Hirschfeld, R., Meuter, W., 2006. Efficient layer activation for switching context-dependent behavior. In: Proceedings of Joint Modular Languages Conference (JMLC'06). Springer Verlag, Oxford, England, pp. 84–103.

Cutumisu, M., Chan, C., Lu. P., Szafron, D., 2004. MCI-java: a modified java virtual machine approach to multiple code inheritance. In: Proceedings of the Third Conference on Virtual Machine Research and Technology Symposium, San Jose, California, vol. 3(2–2).

D'Hondt, M., Jonckers, V., 2004. Hybrid aspects for weaving object-oriented functionality and rule-based knowledge. In Proceedings of the Third International Conference on Aspect-oriented Software Development (AOSD'04), Lancaster, UK, pp. 132–140.

Daniele, L., Dockhorn Costa, P., Ferreira Pires, L., 2007. Towards a rule-based approach for context-aware applications. In: Proceedings of the 13th Open European Summer School and IFIP TC6.6 Workshop (EUNICE'07), Enschede, The Netherlands, pp. 33–43.

David, P.-C., Ledoux, T., 2005. WildCAT: a generic framework for context-aware applications. In: Proceedings of the Third International Workshop on Middleware for Pervasive and Ad-hoc Computing (MPAC'05), Grenoble, France, pp. 1–7.

Desmet, B., Vallejos, J., Costanza, P., Meuter, W.D., D'Hondt, T., 2007. Context-oriented domain analysis. In: Proceedings of the Sixth International and Interdisciplinary Conference on Modeling and Using Context (CONTEXT'07). Springer Verlag, Denmark, pp. 178–191.

Dey, K., Abowd, G.D., 2000. Towards a Better Understanding of Context and Context-Awareness, Workshop on The What, Who, Where, When, and How of Context-Awareness, as part of the 2000 Conference on Human Factors in Computing Systems (CHI'00). The Hague, The Netherlands.

Dockhorn Costa, P., Andrade Almeida, J.P., Ferreira Pires, L., Van Sinderen, M.J., 2008. Evaluation of a rule-based approach for context-aware services. In: Proceedings of the IEEE Global Communications Conference (GLOBECOM 2008), November 30–December 04, 2008. New Orleans, LA, USA.

Elrad, T., Filman, R.E., Bader, A., 2001. Aspect-oriented programming. Communications of the ACM 44 (10), 28–32.

Fahy, P., Clarke, S., 2004. CASS – Middleware for Mobile Context-Aware Applications, Workshop on Context Awareness, MobiSys 2004.

Grassi, V., Sindico, A., 2007. Towards model driven design of service-based context-aware applications. In: International Workshop on Engineering of Software Services for Pervasive Environments: In Conjunction with the Sixth ESEC/FSE joint meeting, Dubrovnik, Croatia, pp. 69–74.

Gu, T., Pung, H.K., Zhang, D.Q., 2004. A middleware for building context-aware mobile services. Vehicular Technology Conference 5, 2656–2660.

Henricksen, K., Indulska, J., Rakotonirainy, A., 2002. Modeling Context Information in Pervasive Computing Systems, Pervasive Computing, LNCS, Springer Verlag, vol. 2414, pp. 79-117.

Henricksen, K., Wishart, R., McFadden, T., Indulska, J., 2005. Extending context models for privacy in pervasive computing environments. In: Proceedings of the Third International Conference on Pervasive Computing and Communication Workshops (PerCom'05 Workshops), pp. 20–24.

Herzeel, C., Gybels, K., Costanza, P., Roover, C.D., D'Hondt, T., 2007. Forward chaining in HALO: an implementation strategy for history-based logic pointcuts. Proceedings of the 2007 International Conference on Dynamic languages: In Conjunction with the 15th International Smalltalk Joint Conference 286, 157–182.

Hirschfeld, R., Costanza, P., Nierstrasz, O., 2008. Context-oriented programming. Journal of Object Technology 7 (3), 125–151 (March/April).

Kan, S.H., 2002. Metrics and Models in Software Quality Engineering, second ed. Addison-Wesley Professional, ISBN-10: 0-201-72915-6.

Kapitsaki, G.M., Kateros, D.A., Venieris, I.S., 2008. Architecture for provision of context-aware web applications based on web services. In: Proceedings of IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'08), Cannes, France, pp. 1–5.

Keays, R., Rakotonirainy, A., 2003. Context-oriented programming. In: Proceedings of the Third ACM International Workshop on Data Engineering for Wireless and Mobile Access, San Diego, CA, USA, pp. 9–16.

Keidl, M., Kemper, A., 2004. Towards context-aware adaptable web services. In: Proceedings of the 13th International World Wide Web Conference (WWẂ04), New York, NY, USA, pp. 55–65.

Kniesel, G., Costanza, P., 2001. JMangler – A framework for load-time transformation of java class files. In: Proceedings of the First IEEE International Workshop on Source Code Analysis and Manipulation, Florence, Italy, pp. 98–108.
Labey, S. De, Dooren, M. van and Steegmans, E., 2007. ServiceJ a java extension for programming web services interactions. In: Proceedings of the IEEE International Conference on Web Services (ICWŚ07), pp. 505–512.
Ni, Q., Trombetta, A., Bertino, E., Lobo, J., 2007. Privacy-aware role based access control. In: Proceedings of the 12th ACM Symposium on Access Control Models and Technologies, Antipolis, France, pp. 41–50.
OMG, 2006. Object Constraint Language OMG Available Specification, v. 2.0, <http://www.omg.org/docs/formal/06-05-01.pdf>.
OMG, 2007. Unified Modeling Language (OMG UML) Infrastructure, v.2.1.2, <http://www.omg.org/docs/formal/07-11-03.pdf>.
Prezerakos, G. N., Tselikas, N., Cortese, G., 2007. Model-driven composition of context-aware web services using ContextUML and aspects, In: Proceedings of the IEEE International Conference on Web Services 2007 (ICWS'07), pp. 320–329.
Saiyu, Q., Min, X., Yong, Q., 2007. Isotope programming model for context aware application. International Journal of Software Engineering and Its Applications 1 (1), 53–66.
Salsano, S., Bartolomeo, G., Trubiani, C., Blefari Melazzi, N., 2008. SMILE, a simple middleware independent layer for distributed mobile applications. In: Proceedings of the IEEE Wireless Communications and Networking Conference (WCNC'08), pp. 3039–3034.
Sheng, Q.Z., Benatallah, B., 2005. ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In: Proceedings of the International Conference on Mobile Business (ICMB'05), pp. 206–212.
Schmidt, A., Beigl, M., Gellersen, H.-W., 1999. There is more to context than location. Computers and Graphics 23 (6), 893–901.
Schmidt, D.C., 2006. Model-driven engineering. IEEE Computer 39 (2), 25–31 (Cover Feature).
Simons, C., 2007. CMP: a UML context modeling profile for mobile distributed systems. In: Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS'07), p. 289b-289b.
Strang, T., Linnhoff-Popien, C., 2004. A context modeling survey. UbiComp First International Workshop on Advanced Context Modelling, Reasoning and Management, Nottingham, pp. 34–41.
Tanter, E., Noye, J., 2005. A versatile kernel for multi-language AOP. In: Proceedings of the ACM SIGPLAN/SIGSOFT Conference on Generative Programming and Component Engineering (GPCE'05). LNCS, Springer Verlag, pp. 173–188.
Tanter, E., Gybels, K., Denker, M., Bergel, A., 2006. Context-aware aspects. In: Proceedings of Software Composition 2006, LNCS 4089, Springer Verlag, pp. 227–242.
Vallejos, J., Ebraert, P., Desmet, B., Cutsem, T. V., Mostinckx, S., Costanza, P. (2007). The context-dependent role model. In: Proceedings of the International Conference on Distributed Applications and Interoperable Systems (DAIS'07), LNCS 4531, Springer Verlag, vol. 4531, pp. 1–16.
W3C, 2004a. RDF/XML Syntax Specification (Revised), <http://www.w3.org/TR/rdf-syntax-grammar/>.
W3C, 2004b. OWL Web Ontology Language Overview, <http://www.w3.org/TR/owl-features/>.
Ying, X., Fu-yuan, X., 2006. Research on context modeling based on ontology. International Conference on Computational Intelligence for Modelling, Control and Automation, and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, 188.

Georgia M. Kapitsaki is a research associate working towards a PhD at the National Technical University of Athens (NTUA). Her research interests include context-aware applications, service-oriented architectures as well as mobile internet and software engineering. She received her Dipl.-Ing from the School of Electrical and Computer Engineering of NTUA in 2005.

Dr. George N. Prezerakos is an Associate Professor in the Electronic Computing Systems Department of the Technological Education Institute (TEI) of Piraeus. He holds a Dipl.-Ing. and a Ph.D. in electrical and computer engineering, both from NTUA. His research interests include context-aware applications and model-driven development. Prior to joining the Academia, he has held various positions in the Greek software and telecoms industry and in the Greek Telecommunications and Posts Commission (EETT).

Dr. Nikolaos D. Tselikas is a lecturer in University of Peloponnese (UoP). His research interests include service-oriented architectures, distributed systems as well as middleware and software engineering. He received both his Dipl.-Ing. degree and his PhD from the School of Electrical and Computer Engineering of NTUA in 1999 and 2004, respectively.

Dr. Iakovos S. Venieris is a Professor in the School of Electrical and Computer Engineering of NTUA. His research interests are in the fields of broadband communications, Internet, mobile networks, Intelligent Networks, internetworking, signaling, service creation and control, distributed processing, agents technology, and performance evaluation. He holds a Dipl.-Ing. degree from the University of Patras and a Ph.D. degree from NTUA. He is a reviewer for several IEEE, ACM, Elsevier, and John Wiley journals, associate editor for the IEEE Communications Letters, member of the editorial staff of Computer Communications (Elsevier), and has been guest editor in the IEEE Communications Magazine.