Providing Context-Aware Adaptations Based on a Semantic Model***

Guido Söldner[1], Rüdiger Kapitza[1] and René Meier[2]

[1] Friedrich–Alexander University Erlangen–Nuremberg
{soeldner,rrkapitz}@cs.fau.de
[2] Trinity College Dublin
rene.meier@cs.tcd.ie

Abstract. Smartphones and tablet PCs are on the verge of revolutionizing the information society by offering high quality applications and almost permanent connectivity to the Internet in a mobile world. They naturally support new applications that take advantage of context information like location, time and other environmental conditions. However, developing these novel context-aware applications is challenging as it is difficult to a priori anticipate their execution context and the adaptations that might be necessary to use new context information. This issue is reinforced by the semantic gap between the low-level technical realization of adaptation mechanisms and the demand to describe adaptations in abstract and comprehensible business terms.
This paper presents programming support for context-aware adaptations based on a semantic model that builds on the AOCI framework. Using such a model, applications and adaptations can be described by means of easy to comprehend business terms. Thereby the model enables the AOCI framework to store and publish both context and domain-specific run-time information and provides a basis for high-level and tailored programming support. This enables to transparently select adaptations based on various criteria and integrate them into applications at run-time. At the level of adaptation mechanisms our approach supports integration for permanent changes using Aspect-Oriented Programming and more importantly for spontaneous and short-time integration of web services by means of interceptors.

1 Introduction and Goals

Smartphones, netbooks and tablet PCs are revolutionizing the information society as we know it today. Due to their increasing computational power and almost permanent connectivity to the Internet they provide an ideal platform for supporting high quality applications in a mobile world. These applications need to act in a context-aware manner

* Part of this work is funded by a research fellowship granted by the German Academic Exchange Service (DAAD).

and therefore have to deal with temporal, spatial, personal or social information provided by the environment. This demands a change of paradigm in application development towards adapting behavior to adjust to the current situation [9]. This is hard to achieve as capturing all possible use cases during design-time is difficult.

As a result, developing adaptive applications remains an arduous task. Most existing approaches provide predefined adaptation mechanisms while adding further adaptation features at run-time is not possible. The reasons for these restrictions [6] are the tight coupling of the application business logic and the adaptation features. In consequence, most approaches omit or only have a limited separate management layer for reasoning and adapting which also impacts expressiveness and flexibility of application-specific adaptation features.

In this paper, we propose programming support for context-aware adaptations that is built upon a semantic model. Our solution is integrated within our Aspect-Oriented Component Infrastructure (AOCI) framework [20] that so far was limited to handle basic annotations using a semantic layer to make AOCI enhanced applications adaptable. We extend this basic support by explicit modelling of the context as well as application-specific domains inside this layer using ontologies of different granularity of abstraction. This allows us to store and to publish context information and provides access to the domain-specific run-time information of the applications. Adaptation developers can now use this information via a simple but yet powerful programming interface, which eases the definition of adaptations. To enable fast dynamic adaptation using external resources, we provide an interceptor-based dynamic adapter for the integration of web services. When executing interceptors, our middleware framework dynamically checks the current context and is able to select and invoke suitable services. This mechanism allows to replace methods at run-time. Hence, using this interceptor-based mechanism, current applications can easily be leveraged and adapted with other services.

Our approach has several advantages compared to other adaptation frameworks. First, due to this work, using the extended semantic layer of AOCI significantly eases the development of adaptive applications. This is due to the fact, that the semantic layer provides access to all the context and domain-specific information using a comprehensive programming interface. Second, based on the semantic layer, the AOCI framework offers different adaptation mechanisms that allow a context-dependent transparent adaptation. These mechanisms can be controlled both by the programmer as well as through configuration options. We support the use of aspect-oriented programming as provided by AOCI for long-term adaptation and newly dynamic and short-term integration of web services on an interceptor-based mechanism. We outline two practical use case scenarios of our approach by extending a Personal Information Manager (PIM) application and show how our framework supports dynamic adaptation in a mobile environment.

The paper is structured as follows: Section 2 discusses related work. Section 3 provides background information and introduces our semantic model. Section 4 describes our approach including a PIM proof-of-concept scenario. In Section 5 we outline the semantic layer and, Section 6 describes the adaptation layer. Section 7 provides an evaluation and finally we conclude this paper.

2 Related Work

In mobile computing scenarios, applications often face the need for adaptation as a result of spontaneous changes in their context. However, providing appropriate adaptation support is still challenging. In the following discussion, we focus on different approaches for dynamic adaptation in mobile environments.

To enable or enhance transparent adaptation, software is often extended by **semantic information** which can be queried by a framework to reason about adaptation. Kiczales et al. [14] propose the use of annotations to define extension points, hence introducing a level of abstraction. However, these extension points are self-defined and do not comply with a semantic model. Consequently the level of abstraction remains low.

The use of so called Crosscut Programming Interfaces (XPI) [22] is similar. XPIs represent explicit and abstract interfaces. Source code and adaptations can be separated by means of design rules. To fulfil this, there are pre-conditions for the source code and post-conditions for the execution of adaptations. XPIs allow to extract semantic information from the implementation, however an explicit meta model is not provided.

Kellens et al. [13] introduce model-based pointcuts, which allow to transfer the matching of adapatations to a more conceptual level. As a semantic language for reasoning they extended the CARMA aspect language combined with the formalism of so called ”intensional views”. CARMA uses SOUL, a language akin to Prolog, to reason about the application of advice. However, compared to our framework, CARMA does neither support dynamic adaptation nor context-awareness.

Research has also been done to provide a common representation format for **semantic modelling**. For this purpose, ontologies can be used. Both SOUPA [8] and SOCAM [12] provide an ontology-based two-layer hierarchical approach. One layer is used to describe general context information and another one for domain-specific ontologies to provide additional vocabularies for supporting specific types of services. To address domain-specific issues during design-time, feature models are widely used. Such feature models are compatible with ontologies as they can be used as subset of the later.

Based on such semantic layers, frameworks can cope with context-awareness. The Context Broker Architecture [7] is based on SOUPA and provides a programming interface to access the context. However, in contrast to AOCI, the focus is mainly on a common semantic model and consequently automatic-adaptation is not supported. Gaia [17], Aura [21] and DoAmI [2] also support context-aware services and can cope with adaptation, but they neither support fine-grained nor transparent adaptation of services.

Unanticipated dynamic adaptation of mobile applications is also addressed in the MUSIC [18] project, which recently addresses aspect-oriented adaptations. Using this framework, planning-based self-adaptation is possible. To accomplish this, the authors use a meta-model and a sophisticated reasoning engine, however this model is not ontology-based. Instead they concentrate on system-level issues like quality-of-service. In [23], the authors propose an aspect-based variability model for representing cross-organizational features. Based on a feature ontology, they can cope with the adaptation requirements at run-time and customize existing features to the client needs.

3 Background: The AOCI Framework

This section introduces the AOCI framework that provides the basis for our semantic model, its programming support and the dynamic integration of web services. The framework enables dynamic adaptation of component-based applications in a distributed environment, e.g., to account for context changes. In order to allow transparent adaptation, component developers have to introduce particular statements, so called *ontology-based annotations*, which adhere to an ontology-based model. This information indicates to the middleware, where adaptations can be applied and which kind of adaptation is suitable. Thus, our system supports a *greybox component* [5] approach, while preserving encapsulation and the concepts of a black box to a certain degree. Conceptually, the AOCI framework can be split into two layers: A semantic layer, which is responsible for context-aware selection of adaptations and a low-level adaptation layer, which performs the necessary steps to adapt the application accordingly to the selection.

The ontology-based *semantic layer* enables to reason about adaptation. In general, the use of ontologies provides a means to gain a shared understanding of a set of entities, relations and functions. Ontologies are widely used in knowledge management and due to their expressiveness, the possibility for context representation and extensibility they are well suited to model application domains and context information. In AOCI, ontologies are used, first, to attach semantic information to both applications and adaptations, and second, to provide basic reasoning capabilities.

The adaptation itself is performed by the *adaptation layer*. It takes all the necessary steps to adapt the application. The adaptation process uses the semantic information attached to the implementation in order to identify the code which needs to be adapted. So far AOCI uses aspect-oriented programming techniques and allows an invasive change of applications typically needed for adaptations with a long-term perspective.

Our implementation specifically targets the adaptation of OSGi-based applications and therefore is implemented using Equinox [10]. As an adaptation mechanism it utilizes Equinox Aspects which supports basic aspect-oriented programming for OSGi.

4 Adapting Service Ecosystems

Next, we motivate how to enable the adaptation of current applications with context-dependent web services in a dynamic fashion. We start by introducing our developer role model that eases the provision of context-aware adaptations. Then, we present use case scenarios for our approach.

4.1 Developer Role Model

To ease writing adaptive applications, our approach provides a *developer role model* and distinguishes three different roles, which are depicted in Figure 1. *Service developers* attach additional semantic information to their source code. This information has to adhere to a semantic model. The semantic model itself is modelled by the *model authors*, which are experts in different domain of services. They build a set of entities which represent the structure of a particular domain and its activities. *Adaptation developers*

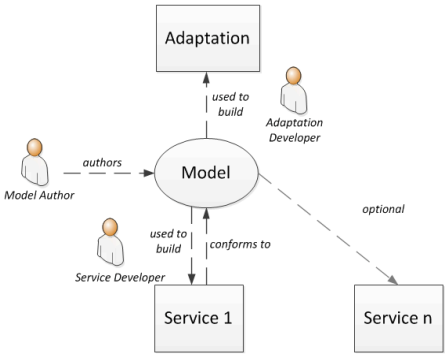

Fig. 1. Developer roles

use the model and its semantic API to write adaptations and determine in terms of the model where and when the adaptation takes place.

4.2 Use Case Scenarios

In our scenario, a student is using a smartphone (e.g. MeeGo-based [19]) with a PIM application running on it. She can use this application to write emails, schedule appointments and manage contacts. Moreover, her university is providing a variety of web services with attached semantic information, among others, a web service to book rooms for learning groups. However, this web service can only be used if a) the student's location is within the university campus and b) the student is member of a tutor group because there is only a small pool of rooms that can be booked on short notice. If these conditions are met, appointments can result in a booking of a room. The corresponding call to the appointment-method will be intercepted, and in case of a room booking, the method call will be transparently replaced by the invocation of the web service. The university also offers a translation web service along with a semantic description, which can be registered in the semantic layer. Using this translation-service, the application can leverage the PIM application with automatic translation capabilities when needed. These use cases can also be expressed more formally. For each use case, we first define the conditions in the context that have to be fulfilled, followed by the adaptation that should be used.

- **RoomBooking:** Appointment.To = 'RoomX' and UserGroup = 'Tutor' $\rightarrow$ Invoke RoomBooking
- **Translate:** Appointment.To.Language != Me.Language $\rightarrow$ Invoke Translation Service

To realize these use cases with our framework, the different developers have to provide semantic information. The service developer attaches the PIM application with semantic annotations created by the model author, and the adaptation developer defines the web service to be invoked and sets the conditions, which have to be fulfilled to invoke the web service. Before adaptations can be applied they have to be deployed to a target

system (e.g., the student's smartphone) in form of conditions and associated web services references. This can for example be performed by accessing a well-known university website. However, other discovery approaches can be imagined. From a conceptual point of view, the two use cases are not anticipated during the design of an off-the-shelf PIM application. However, by means of our middleware, these use cases can be realized at run-time by using our platform. The use cases depict the main goals of our enhancements:

- To provide a high-level semantic model in order to describe applications, adaptation conditions and the adaptation itself.
- A transparent context-dependent adaptation mechanism.

5 Semantic Layer

In this section, we describe the architecture of our extended semantic layer in detail. First, we show how the semantic model is realized, followed by a description of how developers can use this model to attach semantic information.

5.1 Semantic Model

As stated, adaptation usually happens due to changes in the context, hence, it is a key requirement to model the different aspects within the environment like location (e.g. position, orientation), time, identity (preferences, profile) or activities (e.g. sleeping, walking, etc.) or computing resources (device, network, bandwidth, etc.)

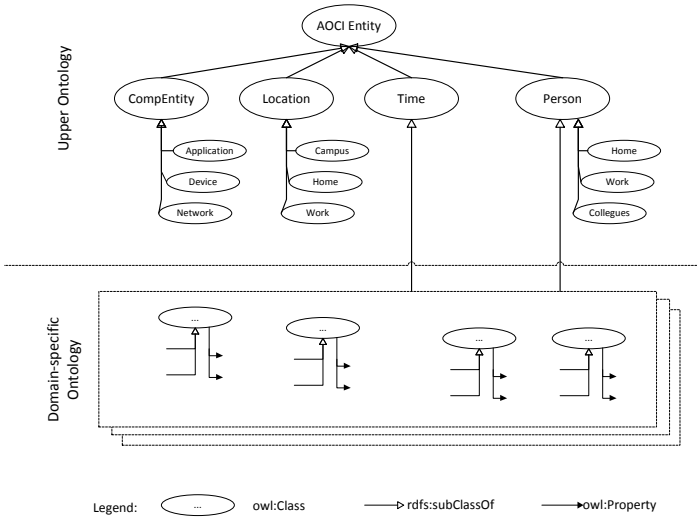

Fig. 2. Upper and domain-specific ontology

For *context modelling*, we use the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [8]. SOUPA provides a core set of entities for the environment and is highly extensible. Our ontology is based on the Resource Description Framework (RDF) [15] and the Web Ontology Language (OWL) [3]. RDF was originally specified as a data model to describe meta data and is now used as a general method for conceptual description. OWL is based on RDF and represents a knowledge representation language for authoring and reasoning about ontologies. Furthermore, context ontologies are used to describe and define the conditions when adaptation can occur. To describe the actual adaptations and applications, our approach uses *domain ontologies*. Domain models are integrated together with context models in the semantic model: The SOUPA ontology serves as upper ontology for the context and is extended with ontologies for domain-specific services as lower ontology (see Figure 2).

Hence, we model the structure and its provided activities of different application domains by means of these lower-ontologies. Consequently, we introduce several OWL entities; we distinguish so called *StructureNodes* and *ActivityNodes*. To put these elements into relationships, our model provides generalization / specialization-, association- and aggregation constructs. Both the StructureNodes and the ActivityNodes span up a separate tree, which can be connected by means of an association if a structure provides a certain functionality. A subset of an ontology for the PIM is depicted in Figure 3. As stated, there are two different subtrees. The upper subtree describes the structure of the PIM domain including modules like email or calendar, the lower subtree models the different functionalities being applicable in that domain. For example, the message node supports sending functionality; consequently these two nodes are connected to each other. The `sendMessage()` text within the rectangle indicates that there is an instance of a reference to an adaptation stored within the ontology which adheres to a sending semantics.

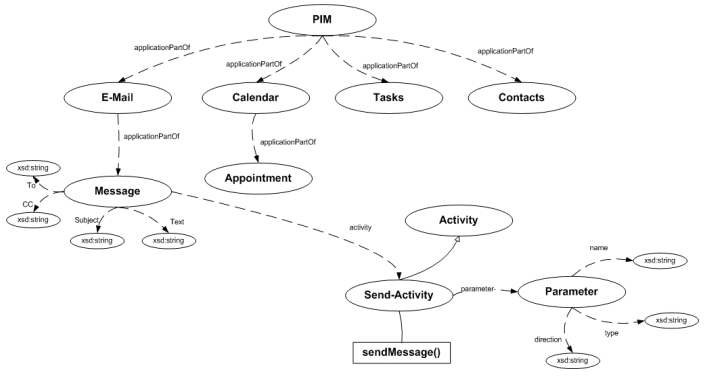

Fig. 3. Graphical representation of a PIM ontology

5.2 Semantic Programming

To implement context-aware applications, we provide an API for managing semantic information. Application developers use annotations to attach semantics to their implementation. Figure 4 shows how to do this according to our second use case.

```
@AOCI.PIM.Appointment
public class Appointment{
        @AOCI.PIM.Calendar.Appointment.To
        User toUser;
        @AOCI.SendActivity.SendMessage
        public void sendAppointment(){...}}
```

Fig. 4. Appointment class annotated with PIM annotations

As the annotations adhere to the semantic model, the information can be extracted at run-time by means of our API, which extracts values based on the ontology and the Java reflection API. This API is used by the dynamic adapter of the adaptation layer, but can also be used in aspect-oriented adaptation code. Figure 5 depicts how to use this API. In our API, the `AOCIContext` is used as an entry point to the ontology and to access the actual execution context, in this case, the `sendAppointment()`-operation. The `Appointment`-class further contains a property referring to the recipient. The first statement extracts the current location from the upper context ontology. The second statement uses the domain ontology to navigate to the recipient user by selecting the corresponding class property and extracts his language.

```
String loc = AOCIContext.Location.toString();
String curLang = AOCIContext.currentNode.getStructureNode().
    getProperty("").getLanguage().toString();
```

Fig. 5. Querying semantic information at run-time

Finally, adaptations can be enabled and disabled at run-time by means of policy files. They can reference concepts within the ontology and define values in the context or the domain, which have to be met to enable or disable the adaptation. This provides flexibility for controlling adaptations as the adaptation process can be defined by users or administrators.

A sample scenario is depicted in Figure 6. First, it defines to use adaptations in classes that are attached with appointments semantics. The next lines specify that the adaptation is only enabled if sender and recipient have different languages and the application is used within the university. If all conditions are fulfilled, the framework will query the ontology for suitable adaptations.

```
<advice type="pre">
    <jp>Activity.Parents='PIM.Calender.Appointment'</jp>
    <condition>
        PIM.Calendar.Appointment.To.Language != User.Language
    </condition>
    <condition>Location='University'</condition>
</advice>
```

Fig. 6. Context condition for the appointment adaptation

6 Adaptation Layer

Based on the outcome of the semantic reasoning, the **adaptation layer** transparently chooses the adaptation method and takes the appropriate steps to perform the adaptation. In this section, we initially discuss the differences between the two supported adaptation techniques from a conceptual point of view, next we detail the adaptation process using our interceptor-based approach enabling dynamic integration and invocation of web services.

6.1 Comparison of adaptation techniques

We support adaptation via aspect-oriented programming and enhanced our framework with interceptors. While the former can be understood as a generalization of the latter, we used different implementations for AOP and interceptors and due to its properties we combined the interceptors-based solution with the integration of web services.

The aspect-oriented programming support is based on the Equinox Aspect framework which features load-time adaptation of components. Adaptation code can be loaded from remote peers. While this approach provides great flexibility, it might require integration of code from an untrusted provider. Furthermore, many adaptations are location-dependent and therefore often of short-term nature, hence from an implementation point of view using load-time weaving seems unreasonable as modified application parts need not only to be reloaded for the integration of an adaptation but also for the removal of an adaptation. Of course, one could deactivate adaptations that are no longer needed but this would lead to an ever-growing number of unused adaptations thereby wasting resources. The code size of adaptations is also an important factor as the code has to be loaded via the network which can take considerable time if an adaptation provides a complex functionality that is attached with a large state (e.g., a vocabulary file in case of the translation service).

Therefore, we enhance our framework with method-level interceptors, representing an approach to easily integrate adaptations into current applications. Such interceptors can be executed before or after or even instead of a selected method. We chose to integrate generic interceptors at application startup as described below which avoids reloading or restarting actions at runtime and supports the dynamic integration and invocation of web services. Thus, a more spontaneous adaptation using services is permitted that has lower security risks compared to the aforementioned approach as loading of remote code is avoided.

Depending on the context and the required adaptation, our framework dynamically selects the appropriate mechanism at run-time if it is available. Table 1 summarizes the main differences of the two approaches.

	Aspect-weaving	Interceptors
Applicationn	Load-time	Run-time
Spontanous interaction	Restart required	Ad-hoc
Perfomance	Very good	Medium
Granularity	Class, method, field	Method
Distributed Adaptation	Yes	No
Availability	Permanent	Only in context
Security risks	Local, network	Network
Coupling	Tight	Loose

Table 1. Comparison of adaptation approaches

6.2 Adaptation Process

Our interceptor-based adaptation is based on a dynamic adapter. Our middleware uses the ASM [4] framework to automatically create hooks at the beginning (respectively at the end) of annotated method calls. ASM is an all-purpose Java bytecode manipulation and analysis framework used to modify existing classes or dynamically generate classes, directly in binary form. Within these hooks, the semantic layer is queried to reason about possible adaptations. In case of an adaptation, the method can be intercepted or redirected to a web service. Internally, the process consists of two subsequent phases, the *transformation* phase and the *invocation* phase.

Transformation Phase In the transformation phase, hooks are integrated within the application to enable the use of interceptors. This phase happens before loading classes into the Java virtual machine. The framework uses the hook-mechanism of the Equinox framework to intercept the classloading. The hook-mechanism allows to register a callback function which is responsible for adapting the classes. The full process is described in Figure 7. First, the `processClass`-method of the `AOCIExtensionHook` is called. Within the body, the configuration for the bundle where the class resides is requested. Upon receipt of the information, a `TransformerService` is requested. This service is used to transform the actual class. We use ASM to inject a call to a dynamic adapter function. Within this function, the current context is evaluated, conditions are checked and web services are invoked dynamically. The transformed class is then stored along with the original class in the `AOCIBundleRegistry`.

The dynamic adapter supports three strategies: pre-invoke, post-invoke and replacement. Pre-invoke is usually used before the method execution with the goal to change the values of the input parameters, hence changing the semantics of the method or in order to validate the ingoing parameters. Post-invoke is executed after the annotated method and can change the result of the function. The replacement directive

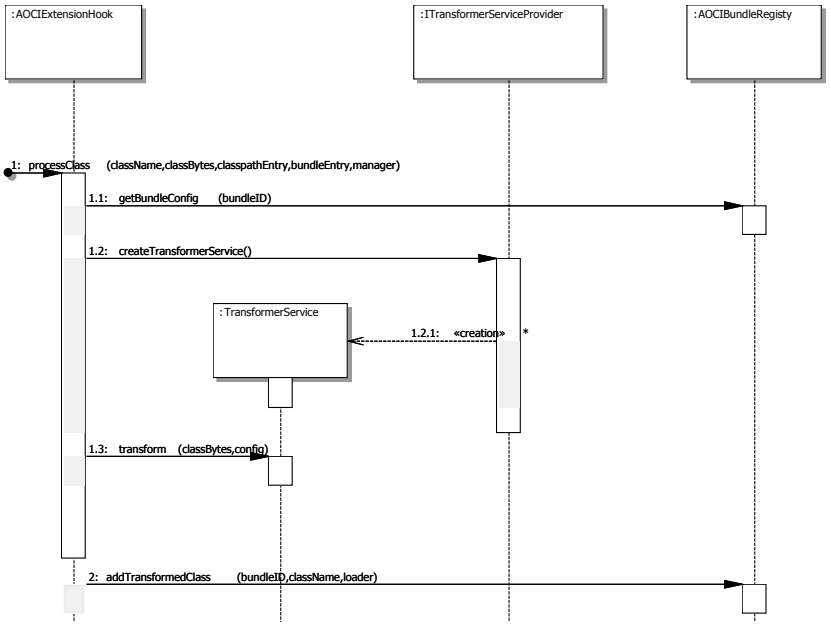

Fig. 7. Transformation phase

provides the possibility to skip the execution of the method. This allows for a transparent way to interchange the behavior of services.

Invocation Phase The invocation phase is performed at run-time and represents the execution of the interceptors. In the first phase of the invocation phase, context conditions have to be evaluated, and based on the result, a decision whether to perform adaptations or not is taken. If the conditions are met, the dynamic adapter reasons by means of the `OntologyService`. This service encapsulates the ontology and provides reasoning capabilities to determine what kind of services should be integrated. The ontology provides a model for describing the structure and behavior of services and applications. This information is used at run-time to dynamically invoke web services. The ontology also contains references to these web services. For the dynamic invocation of the web services we use the DAIOS [16] framework. Compared to other dynamic invocation frameworks, DAIOS is easy to use, performs well and supports a wide variety of web service techniques. DAIOS uses a generic technique based on similarities of input parameters for the mapping between WSDL and DAIOS types and supports both simple data types as well as complex data types.

The next step is to determine the calling semantics of the web service communication. We support two different styles: blocking and non-blocking communication. The standard way to invoke web services is synchronous blocking. However, as web services are often deployed within a Service Oriented Architecture (SOA), some services may take some time to process a request. Consequently, the standard request/response style is not

suitable for such long-running method invocations. Therefore we provide the support for asynchronous (non-blocking) communication.

In the last step, the actual dynamic invocation is being performed. DAIOS supports the core SOA principles: It uses dynamic services invocation, hence no static component like client-side stubs is needed, and instead arbitrary web services can be invoked using a single interface with generic data structures. Furthermore, our invocation engine is protocol-independent and supports a message-driven approach, hence leading to looser coupling of services; both SOAP-based as well REST-based [11] services are supported. The process is shown in Figure 8: First, within the application's dynamic adapter, a reference to the *InvocationService* is fetched. Second, the invoke method is called, which in turn checks the conditions, fetches a suitable web service reference and makes a dynamic call to the web service.

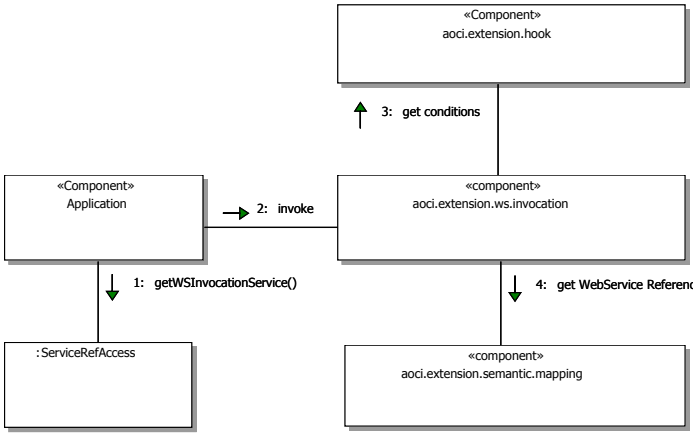

Fig. 8. Invocation phase

7 Evaluation

For the evaluation, we conducted experiments to adapt our application in a short-term style by means of an interceptor-based invocation of a web service and compared the results to our previous aspect-oriented adaptation support. We attached the appointment module of the Nomad PIM [1] (an OSGi-based open-source PIM) with semantic information as described in Section 5 and provided an adaptation to translate the text body of emails by means of a translation web service. For the interceptor-based variant the evaluation includes the transformation during the start of the PIM application, the semantic meta data extraction at run-time and the dynamic invocation. Additionally, we measured the aspect-oriented weaving that is performed during class loading requiring a bundle restart of the effected OSGi components. All experiments were performed 50 times for each type of measure and the web service was co-located with the application on the same physical machine.

Type of measure	Min. [ms]	Max. [ms]	Avg. [ms]
Interceptor integration during application startup	987	1731	1403
Semantic meta data extraction at run-time	0,49	1,28	0,67
Ontological reasoning at run-time	3,7	4,6	4,1
Dynamic invocation at run-time	7,21	7,87	7,39
Aspect-oriented adaptation	1803	2006	2357

Table 2. Benchmark results for interceptors

Table 2 shows that our interceptor-based approach is executing with low overhead, however, due to the transformation phase, the startup of the application takes 1400 ms longer. Subsequent reasoning and invocations only need a few milliseconds. Aspect-oriented adaptation, in contrast, needs a restart of the affected bundles if adaptations are integrated or removed. In both cases this results in a application downtime of 2000 ms at run-time and bears significant startup costs. Compared to the interceptor approach, subsequent invocations do not need additional reasoning, hence the approach provides low invocation costs.

In a second step, we measured the different variants of how dynamic invocation of web services can be applied. We evaluated the pre-invoke, the post-invoke and the replace style. Furthermore we conducted measures for asynchronous and synchronous calls (see Table 3). The replace strategy performs slightly better than the other styles as less reflection-based method calls are needed. Furthermore, due to the described rule-based adaption selection process of the scenario, the ontological reasoning time is quite low.

	Average asynchronous [ms]	Average synchronous [ms]
Pre-invoke	7,21	7,39
Post-invoke	7,65	8,12
Replace	6,31	6,88

Table 3. Benchmark results for dynamic invocation

In summary, an interceptor-based approach can support various scenarios where an external and code-wise unknown service needs to be integrated on demand. Furthermore, the approach is faster in respect of the integration of adaptations than our previous solution using AOP, because the components affected by the adaptations do not need to be restarted.

8 Conclusion

To cope with the requirements of dynamic applications, the ability to adapt due to changes in the context is essential. Existing approaches for adaptive software lack a semantic model to define and perform adaptations in a business-oriented way. In this paper we filled this gap by integrating context and domain modelling in one combined ontology. Based on this model, our application can ease the questions of “when”, “what” and “how” to adapt. In combination with a flexible API to access the semantic model and

the support for dynamic and possibly short-time integration of web services we provide a powerful tool set for dynamic adaptation of applications in a mobile world.

References

1. Nomad PIM. http://nomadpim.sourceforge.net/
2. Anastasopoulos, M., Klus, H., Koch, J., Niebuhr, D., Werkman, E.: DoAmI-a middleware platform facilitating (re-) configuration in ubiquitous systems. In: System Support for Ubiquitous Computing Workshop. At the 8th Annual Conference on Ubiquitous Computing (Ubicomp 2006) (2006)
3. Bechhofer, S., Van Harmelen, F., Hendler, J., Horrocks, I., McGuinness, D., Patel-Schneider, P., Stein, L., et al.: OWL web ontology language reference. W3C recommendation 10, 2006–01 (2004)
4. Bruneton, E., Lenglet, R., Coupaye, T.: ASM: a code manipulation tool to implement adaptable systems. Adaptable and extensible component systems (2002)
5. Büchi, M., Weck, W.: A plea for Grey-Box components. Tech. Rep. 122, Turku Centre for Computer Science (1997)
6. Charfi, A., Dinkelaker, T., Mezini, M.: A plug-in architecture for self-adaptive web service compositions. In: Web Services, 2009. ICWS 2009. IEEE International Conference on. pp. 35–42. IEEE (2009)
7. Chen, H., Finin, T., Joshi, A.: Semantic Web in the Context Broker Architecture. In: Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04). p. 277. IEEE (2004)
8. Chen, H., Perich, F., Finin, T., Joshi, A.: SOUPA: Standard ontology for ubiquitous and pervasive applications. In: Mobile and Ubiquitous Systems: Networking and Services. pp. 258–267. IEEE (2004)
9. Coutaz, J., Crowley, J., Dobson, S., Garlan, D.: Context is key. Communications of the ACM 48(3), 49–53 (2005)
10. Eclipse Foundation: Equinox OSGi framework. http://www.eclipse.org/equinox (2008)
11. Fielding, R.: Architectural styles and the design of network-based software architectures. Ph.D. thesis (2000)
12. Gu, T., Pung, H., Zhang, D.: A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications 28(1), 1–18 (2005)
13. Kellens, A., Mens, K., Brichau, J., Gybels, K.: Managing the evolution of aspect-oriented software with model-based pointcuts. ECOOP 2006–Object-Oriented Programming pp. 501–525 (2006)
14. Kiczales, G., Mezini, M.: Separation of concerns with procedures, annotations, advice and pointcuts. ECOOP 2005-Object-Oriented Programming pp. 195–213 (2005)
15. Klyne, G., Carroll, J., McBride, B.: Resource description framework (RDF): Concepts and abstract syntax. Changes (2004)
16. Leitner, P., Rosenberg, F., Dustdar, S.: Daios: Efficient Dynamic Web Service Invocation. Internet Computing, IEEE 13(3), 72–80 (2009)
17. Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R., Nahrstedt, K.: Gaia: a middleware platform for active spaces. ACM SIGMOBILE Mobile Computing and Communications Review 6(4), 65–67 (2002)
18. Rouvoy, R., Eliassen, F., Beauvois, M.: Dynamic planning and weaving of dependability concerns for self-adaptive ubiquitous services. In: Proceedings of the 2009 ACM symposium on Applied Computing. pp. 1021–1028. ACM (2009)
19. Schroeder, S.: Introduction to MeeGo. Pervasive Computing, IEEE 9(4), 4–7 (2010)

20. Söldner, G., Schober, S., Schröder-Preikschat, W., Kapitza, R.: AOCI: Weaving Components in a Distributed Environment. On the Move to Meaningful Internet Systems: OTM 2008 pp. 535–552 (2008)
21. Sousa, J., Garlan, D., et al.: Aura: an architectural framework for user mobility in ubiquitous computing environments. In: Proceedings of the 3rd Working IEEE/IFIP Conference on Software Architecture. pp. 29–43 (2002)
22. Sullivan, K., Griswold, W., Song, Y., Cai, Y., Shonle, M., Tewari, N., Rajan, H.: Information hiding interfaces for aspect-oriented design. ACM SIGSOFT Software Engineering Notes 30(5), 166–175 (2005)
23. Walraven, S., Lagaisse, B., Truyen, E., Joosen, W.: Aspect-based variability model for cross-organizational features in service networks. status: published