

Layers of Collaboration Aspects for Pervasive Computing

Thomas Cottenier, Tzilla Elrad
Illinois Institute of Technology, USA
{cotttho, elrad}@iit.edu

Abstract. Pervasive Computing scenarios require smooth collaboration and coordinated behavior within large-scale heterogeneous distributed systems. Software entities need to be able to reconfigure themselves seamlessly and without user intervention to respond to changes in their environment or serve customized requests. While Object-Oriented Designs focus on the capabilities of individual objects, they lack expressiveness in capturing collective behaviors. Design Patterns such as the Role Object Pattern are intended to address those issues but did not succeed in elevating Roles to first class entities. There is a need for reusable units of modularity that have the ability to capture interactions that cut across class structures and containers, while accommodating high adaptability and scalability requirements and maintaining low coupling between the collaborating entities. The authors present a distributed Aspect-Oriented collaboration model that extends role models and tackles those requirements. The model supports client-context dependent service instance adaptations for deployment in context-aware environments by combining remote pointcuts and sophisticated crosscuts expressions. An implementation of the model built upon the Globus Toolkit container for grid computing, and the AspectWerkz dynamic AOP Framework is presented.

1. Introduction

Pervasive computing environments interconnect heterogeneous nomadic devices that collaborate and interact seamlessly to create an immerse environment, and provide value added customized services to the user. From the software engineering point of view, such environments exhibit challenging requirements in terms of adaptability, scalability and deployability:

- **Dynamic adaptability**
Because of the nomadic nature of device services in a pervasive environment, the model needs to accommodate dynamic role assignments, at the instance level and entities need the capability to self-tune as to automatically adjust their behavior to fit circumstances.
- **Scalability**
Nomadic devices need to simultaneously interact with various other devices, while playing different roles. The large amount of simultaneous customized interactions an entity might be involved in poses important scalability challenges.
- **Deployability**
Nomadic entities need to be discovered and integrated in existing collaborations dynamically. For incremental deployment, it should be easy to deploy new devices and declare new contexts/features without having to put the system down. Entities should have the ability to seamlessly integrate a new environment. They should make themselves known; discover the other entities they are susceptible to collaborate with as well as their context.
- **Context-awareness and proactivity**
In current client-server interaction models, offered services are oblivious with respect to their clients. It is up to the clients to formulate well-formed requests. In pervasive computing scenarios, services should exhibit proactivity and awareness about the specific contexts of the requests. Therefore, software entities need to be transparently informed one way or another of the state of other software entities so they can anticipate their requests and modify their behavior based on this information.

Based on the above requirements, the authors' position is that software designs for pervasive computing environments should address the following issues.

- **Client-context dependent service instance refinement**
In pervasive computing interaction models, a same service instance might need to simultaneously offer customized services to other entities, adapted to their profile and their context.

- **Encapsulation of context-specific collective behavior**

The most variable elements in pervasive environments are not the entities themselves, but their interactions. There is therefore a need for reusable units of encapsulation that have the ability to capture complex interactions that cut across class structures and containers.

- **Loose-coupling between entities**

Software entities need the ability to join and take an active part in previously initiated collaborations. As such, loose coupling between entities, dynamic invocations and the ability to non-invasively integrate a running system are fundamental.

- **Non-invasive Interaction composability**

To avoid the combinatorial blowup of parameterization expressions and deal with the incremental integration of new entities and new context features, it should be possible to non-invasively compose layers of interactions that are not independent one of each other.

Simultaneously take up those issues is not an easy task. While the relation between Objects and collaboration roles has been studied thoroughly, a key challenge is to reconcile loose coupling between the entities taking part in collaborations while encapsulating the interaction-specific features. The authors' position is that the aptitude to use quantification when assigning roles to collaborating entities might play a main role in providing a scalable solution.

2. Role Models and Service Instance refinement Models

Role models stress the importance of collaborations in Software design. Role models strive for incrementally designing a system by composing independently defined collaborations. A collaboration is a view of an object-oriented design from the perspective of a single interaction concern. They capture the protocols the Objects should implement to fulfill the interaction. A role is a view of a collaboration from the perspective of a core object that takes part in the interaction. Collaborations can be instantiated more than once, and a same object can play different roles. Roles can be assigned to object dynamically.

The role object design pattern [2] implements the role model by using the decorator design pattern. Role instances can dynamically be attached to core objects. Thus, the role object design pattern supports context dependent service instance refinement. Yet, it does not provide a way to encapsulate collaborations. The limitations of the pattern are well known. First, the role and the core object are two different entities, which poses object identity problems. Second, all role specific public methods need to be provided in the interface of the core object, which clashes with the scalability and loose coupling requirements of Pervasive Computing environments. Finally, roles are not composable.

Mixin layers [4] aim at encapsulating independent collaborations. Roles are assigned to objects by inserting their class as mixins into a hierarchy of collaborations specific classes. They offer a non-invasive solution for the implementation of large-scale refinements. However, Mixin layers operate at the class level. As such, they don't allow refinement of interactions at the instance level.

The Lasagne model [5][6] takes a similar approach, and achieves dynamic interaction refinement at the instance level through collaboration layers that encapsulate object wrappers. It offers a scalable solution to the management of collective object behavior in the presence of simultaneous client-specific views.

Mixin and Lasagne layers are encapsulation modules that have the ability to capture interactions that cut across class structures and allow non-invasive integration. Yet, they introduce tight coupling; the mixins and role wrappers are assigned individually on a per-instance base. It should be possible to assign collaboration roles on a per-profile and per-context basis, without having to reference instance handles explicitly.

The authors propose to implement the model of collaboration layers using Aspect-Oriented. The advantage an aspect-oriented language offers over the Object-Oriented Mixin and Lasagne layers approaches is the capability to do quantification. The context in which a particular role should be assigned to a particular instance can therefore be specified in a very flexible way, without having to enumerate those explicitly on a per-instance base.

3. Layers of Collaboration Aspects for Pervasive Computing

Aspect-Oriented implementation of the Role Object pattern (cf. Kendal [3]) offers the following advantages over an Object-Oriented implementation. Object schizophrenia is avoided because collaborating entities have only one identity, to which various roles can be transparently assigned to by aspects. It avoids having to bloat the interface, as new role-specific members can be assigned to the core object through aspect introduction. Moreover, it reduces the coupling between collaborating entities, as collaboration specific protocols are encapsulated in the aspect. Aspects are themselves potentially subject to weaving. Collaborations can therefore be dynamically and non-invasively refined and composed.

When designing pervasive computing applications, we need the ability to express awareness of remote events, as well as awareness to remote context. In that purpose, we adopt two additional concepts: remote pointcuts [8] and sophisticated crosscuts [7]. We adopt a XML aspect definition syntax inspired from the Aspectwerkz [9] dynamic AOP framework for the definition of pointcuts and advice bindings. The corresponding aspect implementation is a regular java class.

Remote pointcuts

Remote pointcuts allow us to locally specify events of interest occurring on remote hosts. When the pointcut is triggered, the metadata of the joinpoint is passed to the local host and the corresponding advice execute locally. For scalability issues, the remote pointcuts are declared as being asynchronous by default. The remote thread that triggers the pointcut does not wait for the advice to be executed. The type attribute (before, around, etc) of the bound advices is therefore ignored.

In mobile environments, the handles to services that participate in an interaction need to be discovered dynamically. We assume the entities publish the services they offer to a registry hierarchy, so that a classification attribute can be used in the remote pointcut specification. The following remote pointcut is triggered on the execution of the `set_context` method on all service instances whose definition is registered with classification concept "Luminosity Sensor", and for which the `ContextManager` Mixin is available.

```
<rpointcut name="luminosity_context" classification="Luminosity Sensor"
          expression="execution(* *$ContextManager.set_context(*))" type="asynchronous"/>
```

Sophisticated crosscuts

Sophisticated pointcuts were introduced by the Event-based AOP framework [7]. EAOP argues that AOP languages should provide support for declaring pointcut expressions that match a sequence of execution points instead of only a single event. By doing so, a cleaner separation between crosscut and action (advice) definition is obtained. Sophisticated crosscuts avoid having to add bookkeeping code into the advices that specify additional conditions under which code should or should not run. Indeed, such code is relevant to the definition of the crosscut, and not of the advice. Sophisticated pointcuts are particularly useful for discriminating contexts, and assign roles.

The following pointcut declaration discriminates between the first time method `m()` is executed and the next times.

```
<pointcut name="doit_first_time" expression="execution(B.m(*))"/>
  <pointcut name="doit_next" expression="execution(B.m(*))"/>
  <advice name="doit" type="around" bind-to="doit_next"/>
</pointcut >
```

The metadata of the `doit_first_time` pointcut can be retrieved in advices bound to the `doit_next` pointcut by calling `joinpoint.getMasterJoinPoint()`, and the sophisticated crosscut can be reinitialized through `joinpoint.reset()`.

The following code sample provides an aspect-oriented solution to the simultaneous client-context dependent service instance refinement problem. Method `m()` of service B is refined according to the luminosity context of the calling instances registered as being local.

```
<aspect name="LuminosityRoleManager" deployment-model="perInstance">
  <param name="context" value="bright"/>
  <rpointcut name="luminosity_setcontext" classification="Local"
    expression="execution(* *$ContextManager.set_context(*))">

    <pointcut name="doit_pc" expression="execution(B.m(*))/>
    <rpointcut name="luminosity_removecontext" classification="Local"
      expression="execution(* *$ContextManager.remove_context(*))"/>

    <advice name="doit" type="around" bind-to="doit_pc" />
    <advice name="resetcontext" bind-to="luminosity_removecontext"/>

  </rpointcut >
  <advice name="setcontext" type="after" bind-to="luminosity_setcontext"/>
</aspect>
```

Fig. 1 LuminosityRoleManager aspect definition file

```
public class LuminosityRoleManager {
  private final CrossCuttingInfo m_info;
  private String role ;
  public LuminosityRoleManager (final CrossCuttingInfo info) {
    m_info = info;
    role = m_info.getInfo("localsystem", this).getParameter("context");
  }
  public Object doit(final JoinPoint joinPoint) throws Throwable {
    //do some stuff
    final Object result = joinpoint.proceed() ;
    //do some other stuff
    return result ;
  }
  public void setcontext(final JoinPoint joinPoint) throws Throwable {
    //do some stuff
    if ((String)joinpoint.getParameter() != role)
      joinpoint.reset();
  }
  public void resetcontext(final JoinPoint joinPoint) throws Throwable {
    //do some stuff
    if ((String)joinpoint.getParameter() == role)
      ((JoinPoint)joinpoint.getMasterJoinPoint()).reset();
  }
}
```

Fig. 2 LuminosityRoleManager aspect

By opposition to the Role object pattern implementations, the protocol used in the collaboration is completely encapsulated in one module and the role can be applied non-invasively to any entity that contains the role manager mixin. As advices can be added and removed at runtime, roles can dynamically be assigned and unanticipated collaborations can be initiated.

4. Implementation and Architectural Issues

We applied the the Collaboration Aspect Layers model to services running in the Globus Toolkit [10] container for Grid Computing using the Aspectwerkz [9] dynamic AOP framework. Beside the capability to dynamically adapt grid service instances, we experienced that AOP techniques facilitate the development of Globus Grid Services, and lead to a modular design. Intuitively, each feature the toolkit adds to Web Service functionality crosscuts its implementation. We were able to modularize service lifecycle management, service data and notification code in aspects, which makes those features transparently deployable through weaving and corresponding WSDL XSL transformations.

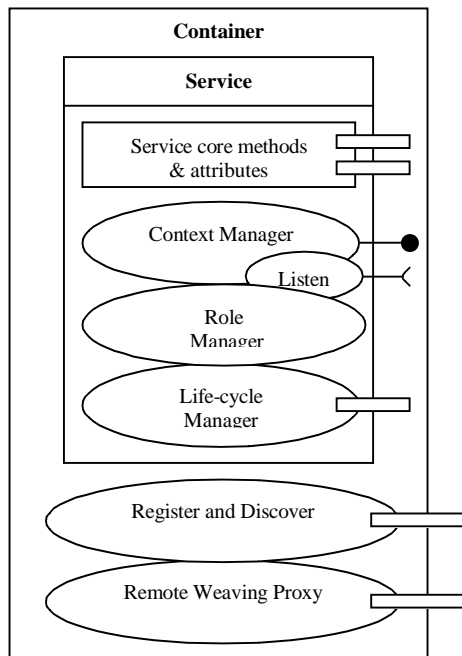


Fig 3. Container/Service components

Lifecycle Management: Contrarily to Web Service, Grid Service instances can be created on demand through requests to a service instance factory.

Service Data: Service Data is a structured collection of information that is associated to a Grid Service to classify and index them according to their characteristics. Service Data is ideal to keep track of the context of a service instance. The Context Manager previously described is implemented as a service Data aspect.

Notification: The Globus Toolkit implements service notifications using the Subject-Observer design pattern. Through aspect weaving, services can register to remote events of interest dynamically and non-invasively. The asynchronous remote pointcut construct is implemented by weaving notification code.

The **context manager** mixin includes a listener mixin, so that the entities' context can remotely be adapted. In addition, the container runs a **remote weaving proxy** and a **registry** management interface.

5. Conclusions

The authors showed the need for reusable units of modularity that have the ability to capture interactions that cut across class structures and containers, while accommodating high adaptability and scalability requirements and maintaining low coupling between the collaborating entities. The authors presented a distributed Aspect-Oriented collaboration model that extends role models and tackles those requirements. The model supports client-context dependent service instance adaptations for deployment in context-aware environments by combining remote pointcuts and sophisticated crosscuts expressions. An implementation of the model built upon the Globus Toolkit container for grid computing, and the AspectWerkz dynamic AOP Framework was presented.

References

- [1] M. Satyanarayanan. *Pervasive Computing: Vision and Challenges*. IEEE Communications, 2001
- [2] Dirk Bäumer, Dirk Rielhe, Wolf Siberski, Martina Wulf. *The Role Object Pattern*, Proceedings of the 4th Pattern Languages of Programming Conference, Monticello, Illinois, USA, 1997
- [3] Elizabeth A. Kendall. *Role Model Designs and Implementations with Aspect-Oriented Programming*, Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, Denver, Colorado, United States, 1999
- [4] Yannis Smaragdakis and Don Batory. *Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs*. ACM Transactions on Software Engineering and Methodologies (TOSEM) , 2002,
- [5] Bo Norregaard Jorgensen, Eddy Truyen. *Evolution of Collective Object Behavior in Presence of Simultaneous Client-Specific Views*. Proceedings of the 9th international Conference on Object-Oriented Information OOIS'03. LCNS 2817. Springer Verlag, 2003
- [6] Eddy Truyen, Bart Vanhaute, Wouter Joosen, Pierre Verbaeten, Bo Norregaard Jorgensen. *Dynamic and Selective Combination of Extensions in Component-Based Application*, Proceedings of the 23rd international conference on Software engineering, Toronto, Ontario, Canada , 2001
- [7] Rémi Douence, Olivier Motelet, Mario Südholt. *A formal definition of crosscuts*. Technical Report 01/2/INFO, Ecole des Mines de Nantes, 2001
- [8] Muga Nishizawa, Shigeru Chiba, Michiaki Tatsubori. *Remote Pointcut – A language Construct for Distributed AOP*. Proceedings of the 3rd international conference on Aspect-oriented software development Lancaster, UK, 2004
- [9] Aspectwerkz homepage <http://aspectwerkz.codehaus.org>
- [10] Globus Toolkit homepage <http://www.globus.org>