Context Distribution for Supporting Composition of Applications in Ubiquitous Computing

Carlos Noguera, Ellen van Paesschen, Carlos Andrés Parra, Johan Fabry

Context Distribution for Supporting Composition of Applications in Ubiquitous Computing

Carlos Noguera,Ellen Van Paesschen, Carlos Parra
University Lille 1, INRIA Futurs - LIFL CNRS UMR 8022, ADAM Team
{noguera|parra|vanpaesschen}@lifl.fr

Johan Fabry
PLEIAD Lab
Computer Science Department (DCC)
University of Chile
jfabry@dcc.uchile.cl

1. INTRODUCTION

Devices and applications reacting to each other and to the environment in which they reside is a very important part of Ambient Intelligent systems. Such *context-aware* systems require mechanisms to react to changes in their surrounding environment.

Context-awareness can be divided into two complementary aspects. On the other hand, a static part that describes the context and, on the other hand, a dynamic part that specifies the actions to undertake in response to changes in that context. In [4] we proposed a generative, model driven framework called CARBO (*Context-Awareness Rule-Based Orchestration*). It uses a model to specify the static part of the context, and condition-action rules to define the dynamic part.

In this poster we present an extension to the CARBO framework that distributes the context state into a set of *context slices*, located on each of the participating entities. In this way, rather than having the complete context state in the device that runs the orchestration engine, each device will keep the state of the entity that represents it.

2. INCLUDING CONTEXT AWARENESS

In CARBO, context awareness in applications is achieved by specifying a model of the context, and writing a set of rules. From these inputs, CARBO generates an orchestration engine that executes services on the applications participating in the context as dictated by the rules. The development of CARBO applications is composed of two phases, a modeling of context and specification of behavior phase followed of a code generation phase. In the first phase, a model of context is defined with a set of abstractions as Descriptors and Entities. In addition to the model, CARBO allows the definition of a set of *condition-action* rules, which represent the behavior of the applications in response to changes in the context.

As a specification language, CARBO uses a limited subset of the Java programming Language. Entities and context descriptors are represented as classes sporting annotations that state their role, and rules are represented as methods containing a single `if` statement. Annotations are used to state the role of the code elements both on the model and rule set. Supose that we are working on a scenario in which a smart home turns on the lights of a room whenever a user identified through its cellphone comes with in range, the model and rules explained previously would be represented by the following Java code.

```
@Entity public class Cellphone{
  @Attribute static String phoneNumber;
}

@RuleSet class LightRules{
  @Rule public void rule1(){
    if(Cellphone.phoneNumber == MYNUMBER
      Light.turnOn();
  }
}
```

From the set of annotated Java classes, CARBO generates a custom orchestration engine that listens to events from an underlying middleware, and keeps the state of the context in an in-memory model. Form this context state, rules and triggered if the conditions are met.

3. CONTEXT DISTRIBUTION

The initial CARBO context model centralizes the context on the device that contains the rule engine. This however does not cleanly map to the fact that we are dealing with different devices when performing application composition. Because of this, additional work must be performed by the CARBO engine to assure that the context representation is synchronized with the actual values of the context, as for example that information is no longer available due to disconnection of devices. It is more sensible to consider that each device contains a slice of the overall context, more specifically the slice that is directly related to the device itself. Here we discuss how we have extended CARBO to support distributed context.

3.1 Handling Offline Context Slices

We regard the main issue of offline context slices to lie in evaluating the condition of the rule. Consider the case of an action of a rule that invokes a service that is unavailable at that time. We should not delay invoking that service until it becomes available. This is because this delayed invocation may happen at a moment where the condition of the rule has become false, which leads to erroneous behavior. We have therefore chosen to simply skip service requests to offline

SAC'08 March 16-20, 2008, Fortaleza, Ceará, Brazil

context slices. Hence, we need to consider the impact of any of these slices being offline when evaluating the condition.

In our approach we return a programmer-specified value when a context slice is offline and expect the condition expression to be constructed so that it handles such errors appropriately. The idea is to treat offline context slices *as if they were online* but contribute negatively to the truth value of the condition.

To enable this approach to work, the rule creator must be able to specify what context information should be returned when a slice is offline. This ensures that the conditions that use this information evaluate correctly. One way in which this can be realized is using tagged futures.

3.2 Tagged Futures

Futures [5], are intended to act as placeholder values for an as yet undetermined object. Futures can be passed around in an application, as if they are the object for which they are the placeholder, but attempting to read the actual value *blocks* until the future is resolved. Our concept of *Tagged Futures* [3] enhances futures by allowing an unresolved future to be read. The programmer specifies a mock value to be used instead of the actual value, as an extra tag added to the future. Read accesses to the future will return this mock value, and as soon as the future is resolved, read accesses will return the actual value. For more information we refer to [3].

The behavior of tagged futures meets the needs for our chosen approach to deal with offline context slices. We can have a context slice as a tagged future of which the different attributes are tagged with their offline value. When online, the context slice works as normal, and when offline, the programmer-specified value for these slices is returned. For example, consider the code below, for the light room example. Here the attributes of the different entities have been given an extra annotation that specifies the value to be returned when offline.

```
@Entity public class Cellphone{
  @Future("\"0\"")
  @Attribute static String phoneNumber;
}
@RuleSet class LightRules{
  @Rule public void rule1(){
    if(Cellphone.phoneNumber == MYNUMBER)
      Light.turnOn();
  }
}
```

The Cellphone entity specifies `0` as a mock value when offline. We assume there is no cellphone with number 0. Consequently, when the user is out of range, the corresponding part of the if-test will fail. As a result of this, the test of the rule does not need to manually take into account the offline nature of context slices. In fact, the rule would be identical if it was written without taking distribution into account. This shows the advantage of using tagged futures: the ability to deal with distributed context **does not come at a cost of higher complexity** of the code of the rule. If sensible mock values are used for when offline, the rule will behave correctly.

4. RELATED WORK

There are several related works in the domain of context-aware systems. Here we describe how some of these approaches deal with disconnection issues. `EgoSpaces` [1] defines a safe-distance protocol, that is, for every transaction to begin, the system should verify that every agent is close enough so that the transaction will be finished before the agent may be out of range. In our approach, we do not need to verify a safe-distance. By using futures, we have the semantics needed to deal with offline devices. The rules will evaluate to false, until the device is online again.

In `Gaia` [6], disconnection is handled by using a presence service. This service is periodically sending heartbeats to verify if an entity is in the active space. If the entity does not respond to the heartbeat, the presence service assumes the entity is no longer available and notifies the rest of the space. Spoon Graffiti uses a similar approach to detect disconnections. Nevertheless, Gaia limits itself to notify the disconnection of an entity. In our approach we also define the actions to follow when an entity goes offline.

Another related project is `CARISMA` [2] in which a set of policies are created to deal with conflicts regarding the information stored in profiles across different entities. Nevertheless, they do not address the problem of disconnection directly and trust an underlying middleware to resolve conflicts related to elements going out of range.

5. CONCLUSIONS

In this poster we present an evolution to the CARBO orchestration engine that distributes context state amongst the different participants by using context slices. As a result, CARBO is a better match to the domain of Ubiquitous Computing setting, where each device is responsible for its own context representation. To implement context distribution we use tagged futures as implemented in Spoon Graffiti. This allowed us to elegantly manage the connection and disconnection of each of the slices that compose the whole context. The small price to pay for this context distribution, is that default values have to be specified for each of the attributes of the entities.

6. REFERENCES

[1] Egospaces: Facilitating rapid development of context-aware mobile applications. *IEEE Trans. Softw. Eng.*, 32(5):281–298, 2006.

[2] L. Capra, W. Emmerich, and C. Mascolo. Carisma: Context-aware reflective middleware system for mobile applications. *IEEE Transactions on Software Engineering*, 29(10):929–945, Sep 2003.

[3] J. Fabry and C. Noguera. Abstracting connection volatility through tagged futures. In *Proceedings of the 2nd Ambient Intelligence Developments (AmI.d) Conference*, sept 2007.

[4] C. A. Parra, M. D'Hondt, C. Noguera, and E. V. Paesschen. Introducing context-awareness in applications by transforming high-level rules. In *Object Technology for Ambient Intelligence Workshop (OT4AmI)*, Berlin, Germany, 2007.

[5] J. R. Halstead. Multilisp: a language for concurrent symbolic computation. *ACM Trans. Program. Lang. Syst.*, 7(4):501–538, 1985.

[6] M. RomĞn, C. K. Hess, R. Cerqueira, A. Ranganathan, R. H. Campbell, and K. Nahrstedt. Gaia: A Middleware Infrastructure to Enable Active Spaces. *IEEE Pervasive Computing*, pages 74–83, Oct–Dec 2002.