A Role-Based Implementation of Context-Dependent Communications Using Split Objects

Jorge Vallejos, Peter Ebraert*, and Brecht Desmet

Programming Technology Lab – Vrije Universiteit Brussel
Pleinlaan 2 - 1050 Brussels - Belgium
{jvallejo, pebraert, bdesmet} @vub.ac.be

Abstract. This position paper focusses on the context-awareness feature in the domain of pervasive computing. Our particular interest is to investigate how context information may influence the communication between applications in this domain. We identify the problem of tangling context information with the definition of functional behaviour, and propose a solution based on a role-model to overcome this problem.

1 Introduction

Within the domain of pervasive computing, context-awareness has commonly been defined as the ability of an application to adapt itself to its dynamic environment. The context of an application is defined as any information in the application's surroundings that may influence its current state or behaviour. Upon a change in the context, an application can either respond directly by invoking some functionality, or modify its behaviour for future interactions. The latter shows the need for context-dependent dynamic adaptation of application's behaviour [1].

In an interaction between two entities, the behaviour of the message receiver does not only depend on the context itself, but also on the context of the sender. We can illustrate this by envisaging a context-aware cell phone which notifies differently the incoming calls according to both the location of its user and the identity of the caller. These two facts belong to different contexts: the caller's identity is part of the caller's context whereas the callee's location is located in the callee's context.

In order for the receiver of a message to adapt its behaviour according to the context of the communication partners, the sender's context has to be passed in the message. However, this context should not get entangled with the functional behaviour of the receiver since this would inevitably lead to less understandable and maintainable code.

* Author funded by a doctoral scholarship of the Institute for the Promotion of Innovation through Science and Technology in Flanders (IWT- Vlaanderen)

We propose the use of a *role-based* model to circumvent the entanglement problem. Such a model allows us to cleanly separate the declaration of the behaviour from its selection. In order to make this selection context-dependent, we present a *context-dependent role selection* mechanism that identifies the most appropriate role for the current context conditions of the participants in a communication process. We describe an implementation of context-dependent roles based on *split objects* [2], a model that uses the object delegation mechanism to represent the different roles of an entity.

2 Context-Dependent Communications in Pervasive Computing

In order to illustrate how the context influences the communication between two applications, we will focus on a scenario in the domain of pervasive wireless networks. For this purpose, we consider a system composed of context-aware entities (objects, components, applications, etc.) residing on different mobile and embedded devices. In this system, the context may vary from one entity to another. Moreover, different entities may be aware of different context information. We go out from the premise that every computer system is aware of its context, and that it gets notified when its context changes (like in the ContextToolkit framework [3]). Figure 1 illustrates the communication between two entities with different surrounding context information.

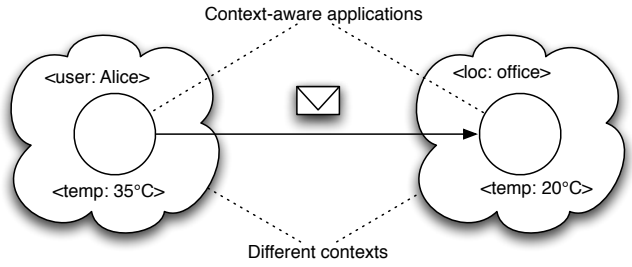

Fig. 1. A context-dependent communication

The most common scenario of context-awareness envisages an entity being influenced by its direct surrounding context. Upon reception of a message, the entity may tailor its response to its context. As an example, we recall the context-aware cell phone example. This device may use, for instance, its physical location to provide location-dependent behaviour (e.g. different ways of notifying incoming calls). This implies that the cell phone application has to dynamically adapt its behaviour in response to changes of location. Note that this adaptation does

not imply a complete change of behaviour. In this example, the location information should just modify the part of the behaviour related to signaling incoming calls.

In a different scenario, the receiver's behaviour might not only be influenced by its own context, but also by the context of the sender. For instance, the cell phone application could also notify incoming calls according to the context information of the caller (identity, location or even some kind of "urgency level of the call"). This context dependency requires the information of the sender to be included somehow in the message. Nevertheless, this information should not interfere with the original method invocation (e.g. passing this information as new parameters in the method invocation), as this would harm the understandability and the maintainability of the code.

Both scenarios show the strong relation between the context information and the behaviour of an application. However, this does not imply that context information and behaviour should be entangled with one another. Reasoning about context information inside the behaviour's implementation would lead to undesirable situations such as scattered context-dependent if-statements, resulting in cluttered and less readable code.

An alternative solution could be the use of polymorphism. However, in a setting of pervasive computing, we can have many context parameters. Expressing all the combinations of context parameters in a hierarchical way, would inevitably lead to a combinatorial explosion of class definitions. This is why we conclude that in order to deal with context-dependent communications, we need a model that satisfies the following conditions:

Dynamic behaviour composition The model should enable the implementation of applications based on composable parts, in order to represent partial adaptation of behaviours.

Dynamic context adaptation The model should enable an application to adapt to the context by dynamically switching its behaviour. With context, we mean the contexts of both participants in the communication.

Context-dependent messages The model should provide a message passing mechanism that allows programmers to include context information without coupling it with original method invocation.

3 A Role Model for Context-Dependent Behaviour

In this section, we show how a model based on roles can satisfy the three conditions for context-dependent behaviour. In such a model, application entities are composed of roles which represent the different behaviours the entity can adopt according to the context. A role is a particular point of view on an entity that has an identification and a partial definition of that entity. In addition, a role can specialise another role so that different roles can share behaviour. The model's message sending protocol allows the specification of the role the receiver should adopt to respond to the message. Hence, if context-dependent behaviour

is modelled as roles, an application could adapt to its context by just assuming the appropriate role.

In the object-oriented literature, the essence of role-based model has been previously studied by [4–6]. In what follows, we focus on the implementation of roles using split objects [2], evaluating its suitability for context adaptation.

We are currently exploring the development of role-based applications in a prototype-based language called AmbientTalk [7]. This language adheres to the ambient-oriented programming (acronym AmOP) paradigm which was specially invented for pervasive computing. One of claim the AmOP paradigm makes is that prototypes are better suited for software development in this domain. This is why we opted for split objects; a prototype-based approach to the role model.

3.1 Modelling Roles with Split Objects

The Split objects approach uses the delegation mechanism of prototype-based languages to represent the different roles of an entity (called *viewpoints*). A split object is defined by a collection of pieces (represented by objects in the delegation hierarchy) that hold part of the description of the state and behaviour of the split object. Each piece denotes a role and the delegation hierarchy denotes the specialisation of roles. Split objects are first class entities, whereas pieces are not. It means that only split objects are directly accessible. Messages to split objects are sent on a *per viewpoint* basis, which means that a message has to specify the role(s) from which the split object should respond to this message (no specification of the role implies for the split object to adopt the most general role found at the root of the delegation hierarchy).

We now evaluate the convenience of representing context-aware applications as split objects based on the conditions of context-dependent communications we identified in Section 2:

Dynamic behaviour composition Using split objects, context-dependent behaviours can conveniently be modelled as pieces. Since these pieces are structured in an object delegation hierarchy, programmers can benefit from the flexibility of this structure for performing dynamic behaviour composition. In a split object:

- pieces can dynamically be added, eliminated or modified.
- properties of a piece (variables and methods stored in the slots of the object representing the piece) can be dynamically added, eliminated or modified.

Figure 2 illustrates a basic role-based implementation of the cell phone application using split objects. The application is modelled as a split object where the pieces represent the different context-dependent behaviours. Partially modifying the behaviour is easy to achieve since split objects can share and specialise behaviour. We can see that the #`loud` and #`discreet` roles, are sharing the behaviour of the #`call` method, while they are specialising the default behaviour of the `signal` method.

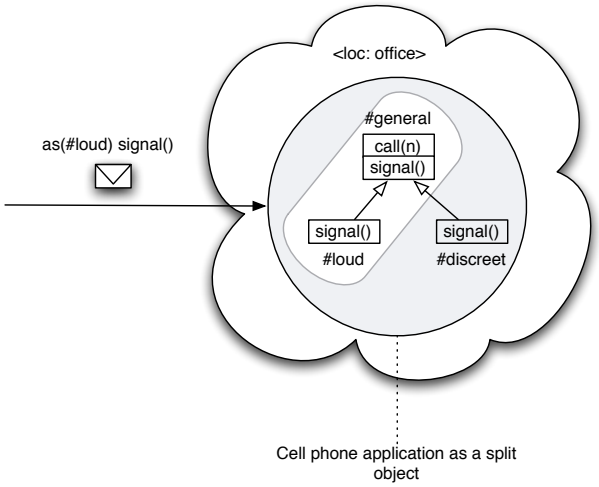

Fig. 2. A location-aware cell phone application as a split object

Dynamic context adaptation A split object only adopts a specific behaviour if it receives a message indicating a role. In such a case, the split object responds to the message based on the delegation chain of the piece denoting such role. In the example of the cell phone, if the `#loud` role is specified in a incoming message, the application will respond according to the delegation chain composed of the pieces `#loud` and `#general`.
However, the specification of a role in the messages is not always possible, since the sender may not necessarily know the different roles of the receiver. We need, therefore, an additional process to select the role an application should adopt according to the current context conditions. We have called this process the *context-dependent role selection*, which is discussed in the following section.

Context-dependent messages Split objects provide a message sending mechanism that separates the specification of the role from the method invocation (e.g. `cellPhone as(#loud) signal()`). We believe that a similar scheme should be used to pass context information of the sender without coupling it to the method invocation. We also discuss about this abstraction in the message sending mechanism in the following section.

3.2 Context-Dependent Role Selection

In the beginning of this paper, we have argued that in a communication between two entities, the response may depend on the contexts of both participants. For coping with these context dependencies, we subsequently proposed a role-based model, in which different kinds of behaviour are shaped into roles. Because we

also want dynamic adaptation of behaviour depending on context, we need a mechanism to use the context information for selecting a role. We have shown that a good separation between context information and behaviour specification is a must for ensuring understandability and maintainability. As such, we need a mechanism that bridges the gap between context information and role selection.

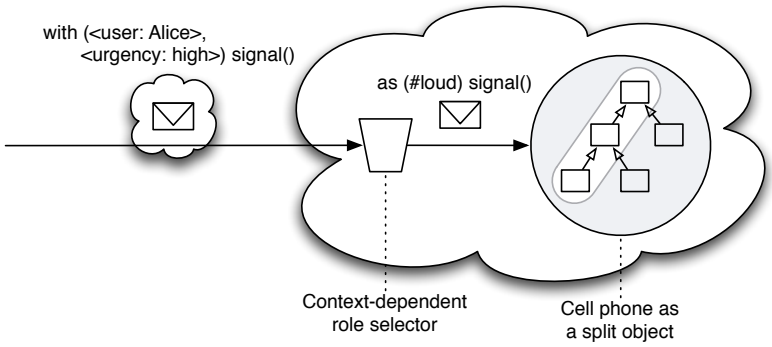

Fig. 3. Context-dependent role selection

We envisage a *context-dependent role selection* mechanism, that uses the context information of all the communication partners for deciding which role the receiver should take. Figure 3 sketches the communication process of our model. We can see that the context-dependent role selector takes as input a contextualised message and consequently calls the message in a split-object way (by specifying which role the receiver should take). Note that we need a way to get the context information from both the sender and the receiver to the role selector. This will be further discussed in section 4.

In order to improve the understanding of the message sending protocol, we describe step by step how a message is delivered from caller to callee in our cellular phone example:

1. The caller sends a message that contains the method invocation and (part of) the context information of the sender. We could represent this "contextualised" message as follows:

   ```
   cellPhone with(<user:Alice>,<urgency:high>) signal()
   ```

2. The message is received by the context-dependent role selector which is also aware of the context information of the receiver.
3. The role selector searches for the appropriate role according to the contexts of both sender and receiver.
4. Once the role has been selected, the role selector sends a new message with similar sender, receiver and method invocation to the original message, but

now also including the role that the receiver should adopt in order to produce a response. The translated message has the same form as the messages in split objects:

```
cellPhone as(#loud) signal()
```

5. Finally, the receiver (which is a split object) adopts the role and produces a response.

4 Discussion and Future Work

In this section, we give an overview of some issues that were raised throughout this paper. For each of these issues, we propose some solutions and discuss about the consequences they might have. Because we did not take any final decision yet on these issues, each one of them presents a topic for future work.

Where should the context-dependent role selector be located? We identify three possible positions for locating the selector: at the *sender-side*, at the *receiver-side* or in the *connection between both*. Throughout this paper, we assumed that the role selector resides on the receiver side. Since the role selector needs to be aware of the different roles the receiver can take, it seems reasonable to locate it on the receiver side. Moreover, this would also ease the process of getting the context of the receive to the role-selector.

How do we get the sender's context information to the role selector? Because contexts need to passed around in our model, we have to choose wether this is done by using a *pull-* or a *push-strategy*. Throughout this paper, we assumed a push-strategy, in which the context information is pushed to the receiver together with the message. However, one could opt for a pull-strategy, in which the initiating message does not contain any context-information. In that case, the role selector would query the communication participants for the information it requires for selecting the appropriate role for the receiver. This discussion boils down to the classic differences between push- and pull-models (identified in the observer pattern in [8]). A push-model requires less communications, but might have unnecessary overhead, as some context information could not be necessary to select a role.

How would the decision making process look like? The role selector takes as input the context parameters of all the communication partners. From the moment that all the context parameters are present, the role selector can start the role selection process. This process may be supported by a rule-based engine which reasons about the context parameters and determines the role that corresponds to such parameters. The rules should be implemented in a declarative way, so that they are easy to write, understand and expand.

A similar rule-based system for behaviour selection can be found in the context of subject-oriented programming [9]

When should we pass information as context and when as method parameters? The difference between the context and method parameters lies in the role selection process. While context parameters are used for selecting a role, method parameters are just passed on to the callee. The decision of using a context or method parameter is thus application-dependent and should be taken by the application designer.

Can we model communication between more than two participants? Throughout this paper, we always exemplified our model by communications between two parties. Our model, however, could easily be extended to support more communication partners. In a previous implementation of role models, this extension has been successfully accomplished by including the notion of activites [10]. Nevertheless, this would rule out the positioning of the role selector at the caller and the connector sides.

5 Conclusion

This paper tackles a problem that was identified in the domain of *pervasive computing*. In this domain, applications are said to be *context-aware*, since they are aware of their surroundings, and can act upon changes in these surroundings. A context-aware application acts differently on a message it receives, depending on its context. We showed however, that in an communication between two or more context-aware applications, the behaviour of the receiver does not only depend on its own context, but also of the context of the other communication partners. We showed that in such a setting, we need (i) a way to cleanly separate the context information from the functional behaviour, (ii) a mechanism to send context information around, and (iii) a way to dynamically adopt to the appropriate role in a certain context.

We propose the use of a *role-based* model, because it allows us to cleanly separate the declaration of the behavior from its selection (satisfying the first requirement). The model's message sending protocol allows the specification of the role the receiver should adopt to respond to the message. Modelling context-dependent behaviour as roles, would allow the application to adopt to its context by just assuming the appropriate role. This is how the role-based model satisfies the second requirement.

In order to make sure that the receiving entity adopts the appropriate role, we present a *context-dependent role selection* mechanism that identifies the most appropriate role for the current context conditions of the participants in a communication process. This mechanism should reside on the receiver's side, and should be implemented as a rule-based reasoning system that is composed of declarative rules, making them easier to write, understand and expand.

We describe an implementation of this model based on *split objects* [2], and show how this implementation satisfies the three requirements that were stated above, and as such solves the problem that was identified.

References

1. Oreizy, P., Gorlick, M., Taylor, R., Heimbigner, D., Johnson, G., Medvidovic, N., Quilici, A., Rosenblum, D., Wolf, A.: An architecture-based approach to self-adaptive software. IEEE Intelligent Systems (1999) 54–62
2. Bardou, D., Dony, C.: Split objects: a disciplined use of delegation within objects. In: Proceedings of the 11th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, ACM Press (1996) 122–137
3. Salber, D., Dey, A.K., Abowd, G.D.: The context toolkit: aiding the development of context-enabled applications. In Press, A., ed.: CHI 99: Proceedings of the SIGCHI conference on Humon factors in computing systems, New York, USA (1999) 434 – 441
4. Kristensen, B.B., Osterbye, K.: Roles: Conceptual abstraction theory and practical language issues. Theory and Practice of Object Systems **2**(3) (1996) 143–160
5. Anderson, E., Reenskaug, T.: System design by composing structures of interacting objects (1992)
6. Smith, R.B., Ungar, D.: A simple and unifying approach to subjective objects. Theory and Practice of Object Systems **2**(3) (1996) 161–178
7. Dedecker, J., Van Cutsem, T., Mostinckx, S., D'Hondt, T., De Meuter, W.: Ambient-Oriented Programming in ambienttalk. In: Proceedings of the 20th European Conference on Object-Oriented Programming (ECOOP), Dave Thomas ed., Lecture Notes in Computer Science, Springer, to appear. (2006)
8. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
9. Harrison, W., Ossher, H.: Subject-oriented programming: a critique of pure objects. In: OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications, New York, NY, USA, ACM Press (1993) 411–428
10. Kristensen, B.B., May, D.C.M.: Activities: Abstractions for collective behavior. Lecture Notes in Computer Science **1098** (1996) 472–??