Towards Context-Aware Components

Antoine Beugnard [†] Sophie Chabridon [‡] Denis Conan [‡]
Chantal Taconet [‡] Fabien Dagnat [†] Eveline Kaboré [†]
{Antoine.Beugnard,Sophie.Chabridon,Denis.Conan,Chantal.Taconet,Fabien.Dagnat,Eveline.Kabore}
@institut-telecom.fr
[†]Institut Télécom, Télécom Bretagne, Brest, France
[‡]Institut Télécom, Télécom & Management SudParis / UMR CNRS SAMOVAR, Évry, France

ABSTRACT

Making component self-adaptable requires observation abilities. Observation features are usually intricated within the functional code. We propose to consider observation as an aspect. The solution we present in this paper allows an explicit specification of which observation data are required by a business component and which observation data this component offers to the other entities of the system. We illustrate the advantages of this separation of concerns for a self-adaptable web server.

Categories and Subject Descriptors

D.2.2 [**Design Tools and Techniques**]: Modules and interfaces; D.2.11 [**Software architectures**]: Domain-specific architectures

General Terms

Design, Experimentation

1. INTRODUCTION

Software components seem to be a promising way of building software thanks to their assembling features. Many component models have been proposed from academic ones (*e.g.*, Fractal, OpenCOM) to already widely used industrial ones (*e.g.*, CCM, J2EE, .NET, OSGi, SCA). All these models propose a quite similar architecture: offered and required ports, configuration attributes and control ports.

In parallel to this architectural branch of progress, the software engineering field proposes another way of expressing modularity thanks to aspects (or more generally, preoccupations).

Some works have already tried to merge both approaches: components and aspects: FAC [14], AspectCCM [7], FuseJ configuration language [17].

We propose in this article to tackle a specific preoccupation which is central to adaptation: observation. We consider the ability to observe its context as a specific aspect that can be isolated from the core functionality of any component. In addition, we propose a development process based on model engineering to cope with this aspect: The observation is specified thanks to a model (that conforms to a meta-model of observation) and is integrated into the component with a model transformation that generates a component with all the specified features of observation. The observations are reusable and can be merged with other components. We have experimented this approach in a preexisting component architecture in order to improve its adaptive features thanks to observations. This shows how simple it is to separate the core functionality of the component from its observation abilities.

The remainder of this paper is organised as follows. Section 2 motivates the promotion of an observation aspect. Section 3 presents the component model integrating the observation contracts which are more thoroughly defined in Section 4 and experimented in Section 5. Finally, Sections 6 and 7 discuss the contribution with regard to related work and conclude the paper respectively.

CASTA'09, August 24, 2009, Amsterdam, The Netherlands.

2. OBSERVATION ASPECT

It is now widely accepted that a software component is functionally defined by the services it offers and requires. When such a component needs to be adaptable, all models of adaptability ([1, 5, 6, 10, 11, 16, 19]) are based on a 4-steps process [12]: *1)* observe the context; *2)* analyse the context to decide whether to adapt (applying a strategy); *3)* plan the change (that may be predetermined); and *4)* execute the reconfiguration plan.

In this article, we focus on the first step. We propose a process that makes functional aspects of components independent from observational ones. Current component models do no separate the functional interface from the observation one, even if it could be a good practice. Note that the relation is asymmetric since observations may rely on the functional services of the component. This separation of concerns gives the opportunity to change observations without modifying the core part of components. The context-aware component is then built from these models by a kind of transformation that weaves both descriptions and proposes an implementation of the observation infrastructure.

We propose to study the following scenario. A web server has been developed without any adaptation consideration. This server is deployed on a small old-fashioned reused computer. Its administrator observes that the web server suffers denial of service and decides to detect high rates of in-

complete attempted connections so that the server applies counter-measures such as blocking requests from incriminated IP domains. The first possible solution is to make a patch to the server source code. But, the administrator also notices that the server does not scale well when the CPU load exceeds some threshold. Wondering how many different observations may be of interest and would necessitate to modify the server source code, the administrator freezes and eventually lets the server unmodified.

We propose to solve this problem with three elements: a declarative way of specifying observations of interest; a kind of weaver to integrate the observation code into the core application; and a set of predefined components dedicated to observation. The first two elements constitute the contributions presented in this paper while the last one relies on the COSMOS [8, 15] context management framework.

3. CONTEXT-AWARE COMPONENT

In this research work, we have chosen FRACTAL [3] as the demonstrating component model. FRACTAL is a general-purpose component model of the OW2 Consortium[1]. This is a hierarchical and reflexive component model with sharing. As we will see, we use FRACTAL for its flexibility and its extensibility: one of the main features of this component model is that it is dynamic and reflexive.

Figure 1 illustrates the different entities in a typical FRACTAL component. The thick black box denotes the controller part of a component, while the interior of the box corresponds to the content part of the component. T symbols protruding from the box are interfaces. Interfaces appearing on the left and right sides depict server and client interfaces, respectively. Interfaces appearing at the top of the box represent reflexive control (extra-functional) interfaces such as the life-cycle controller, the binding controller, the attribute controller or the content controller interfaces.

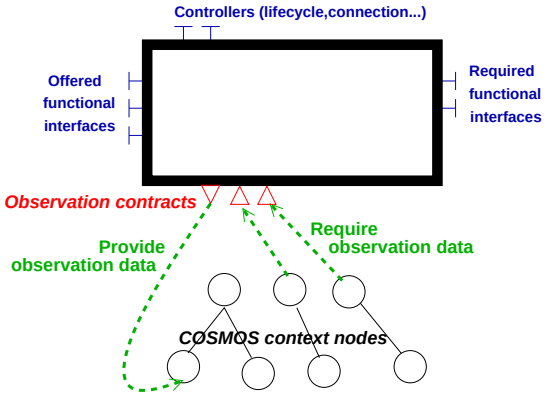

Figure 1: Self-adaptable component with observation contracts

We complement the FRACTAL component model described so far with observation contracts which appear at the bottom of the component as drawn in Figure 1. Observation contracts get translated into extra-functional interfaces. Contrary to control interfaces at the top that are exclusively server interfaces, the interfaces we add can be client or server interfaces connected to the infrastructure. In our case, the infrastructure is the COSMOS context manager which is itself implemented using FRACTAL.

COSMOS[2] is a component-based framework for managing context information in ubiquitous environments for context-aware applications. In particular, COSMOS identifies the contextual situations to which a context-aware application is expected to react. These situations are modelled as *context policies* that are hierarchically decomposed into fine-grained units called *context nodes*. A context node is a context information modelled as a FRACTAL component. The relationships between context nodes are *sharing* and *encapsulation*. The sharing of a context node —and, by implication, of a partial or complete hierarchy— corresponds to the sharing of a part of or a whole context policy. Context nodes at a hierarchy's leaves (the bottom-most elements, with no descendants) encapsulate raw context data obtained from context providers that in this paper are either legacy framework entities or other application components. Communication between context nodes through the hierarchy may be bottom-up or top-down. The former case corresponds to notifications sent by context nodes to their parents, whereas the latter case corresponds to observations triggered by a parent node.

On the one hand, observation contracts translate into client interfaces to express the fact that the component *i)* sends a notification of some of its observation data to the context manager or *ii)* asks for an observation of the context. On the other hand, server interfaces correspond *i)* to the receipt of a notification from the context manager or *ii)* to the answering of an observation of its observation data by the context manager. For a better separation of concerns, the context manager mediates all the exchanges of observation data between application components, that is we do not envisage that observation contract interfaces are directly connected.

4. OBSERVATION CONTRACTS

Since we target context-aware applications, that is applications that are designed and implemented with context-aware business capabilities, we propose that designers specify observation contracts in models (*à la* UML). In the case of our Web server example, designers specify in UML class diagrams that the Web server requires the "CPU load"context data and provides the denial-of-service data containing the list of IP domains from which too numerous requests are issued. Either an administrator's console or an adaptation service collects these denial-of-service data from the Web servers they monitor through the context manager. In the former case, the administrator can then manually intervene. In the latter case, the adaptation service analyses the observation data, then plan counter-measures and apply them : this MAPE-K autonomic control loop defines an autonomic system of Web servers [12].

Part[3] of the meta-model which allows a component to define its observation contracts is presented in Figure 2. Being a primitive or composite component, a ContextAwareSystem defines its ObservationContracts through which it is linked to contextual Entities of the observable world via Observables either required or provided. An example of Entity in

[1] `http://fractal.ow2.org`

[2] `http://picoforge.int-evry.fr/projects/svn/cosmos`

[3] Due to space limitation, the complete meta-model is not presented in this paper.

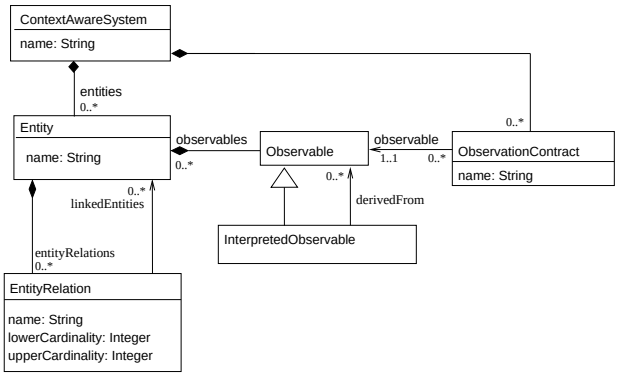

Figure 2: Observation Contract Meta-Model

the Web server scenario is the local Computer. The presence of the Entity concept expresses the fact that we need to state the origin of the observation data. For instance, as depicted in Figure 3, the Web server of the scenario requires the CPULoad of its computer, not from a remote computer. Note also that since the Web server provides observation data it appears in the model of Figure 3 as an observable Entity. In addition, observable Entities are linked thanks to EntityRelationships and define Observables that can be simple or aggregated (InterpretableObservable). Therefore, observe the EntityRelationship between WebServer and Computer in Figure 3.

An ObservationContract (not detailed in this simplified model) describes how a component and an observable are associated. Firstly, it defines if the observation data are provided or required by the component. In case of required observation data, it also defines the mode: observation or notification. For a notification, it defines what event triggers this notification (*e.g.*, for the CPUload Observable, the notification may be triggered periodically, or if the CPUload exceeds a given threshold).

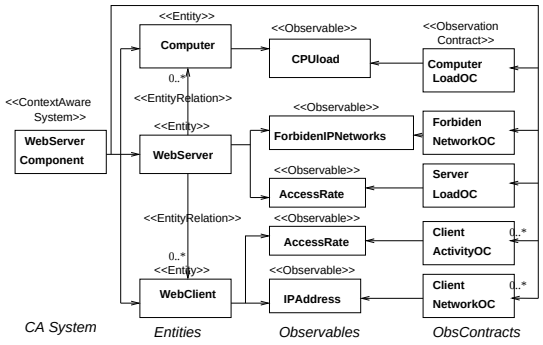

Figure 3: Web Server Observation Model

Furthermore, the contract defines the QoC (Quality of Context) characteristics required for the observations. Indeed, since context-aware applications are highly dependent upon context information, a precise knowledge of its QoC is required. Meta-information can therefore be associated to observations defining their QoC. In our work, we focus on some generic QoC parameters which are used in most applications [4], such as precision (gives bounds to better mirror reality), correctness (probability there exist unintentional internal errors), resolution (granularity), up-to-dateness (age and lifetime correlation) and trust-worthiness (how much confidence can be put in a context source). This part of our work is out of the scope of this article.

5. EXPERIMENTATION

We have experimented this separation of preoccupations in the Fractal implementation of a Web server called *Comanche* [4]. As shown in Figure 3, we have specified the fact that we were interested in the observation of the used CPU and the access rate (other observables have not been used).

We have developed a set of models and model transformations (in the Kermeta language [13]) that use the UML models describing the business functionalities of the Web server and the context-awareness model as depicted in Figure 3. We have then generated a FRACTAL ADL description of the system. The resulting description is directly usable to deploy the Web server with its observation abilities.

Without any observation integration, the Web server does not trace any observation. In a first evolution, we introduce a first observation, the used CPU for instance, and regenerate the component architecture. The system now traces the percent of CPU used. In a second evolution, we add the access rate observation and regenerate the whole system automatically. The system eventually traces both the used CPU and the access rate.

This scenario shows how explicit observations can be weaved and reused with the component thanks to model transformations. The study needs to be generalized since only two observables have been modelled and tested.

6. RELATED WORKS

To our best knowledge, none of the current component models propose an explicit model of observation. This prevents composition, reuse and some verification of observation contracts. The usual approach is to merge observation aspects with the business part of the system.

In the SAFRAN approach [10], the observation is mixed with the adaptation controller and cannot therefore be reused. If two controllers require the same observation, they both have to describe it. The context-aware extension of CORBA IDL called CA-IDL (context-aware IDL) proposed in the RCSM middleware [18] allows the definition of observation contracts comparable to our proposal. However, it is restricted to a set of predefined observables.

The CORTEX project [2] has defined the concept of sentient objects. A sentient object is an entity that collects context information, processes it and produces software events. Although sentient objects enable the development of context-aware applications, they do not envision observation as an extra-functionality and are based on software objects rather than components limiting the ease of composition.

OSA [9] relies on the Fractal component model to hide the component observation in the controller part of the components. This work is the closest to our proposition in that the mechanisms to get observation information correspond to aspect weaving. However, OSA can only express the fact that observation data are provided by the component, not that they are required by the component, that is in only one direction, limiting reusability in other application domains.

[4]http://fractal.ow2.org/tutorials/comanche.html

In addition, these data are only provided by observation: the notification mode is not available.

7. CONCLUSION AND PERSPECTIVES

We have considered observations as a first-class aspect that can be separately specified from the functional part of components. This approach has been successfully experimented in a Web-server application. We claim that observations can be weaved at design-time with components, leading to the constitution of separate repositories for business components and observations whose elements can be combined to build context-aware applications that both observe and are observable, the first step towards truly adaptable systems.

For the time being, the observation contracts are described with a UML-like model, but we believe that a domain specific language could be defined in order to improve the usability and the checking of consistency.

In the short term, we intend to increase the number of observables that can be weaved in the FRACTAL component model. Beyond, we would like to explore remote (distributed) observation contracts and their use for the adaptation of context-aware applications.

8. REFERENCES

[1] Dhouha Ayed and Yolande Berbers. Dynamic adaptation of CORBA component-based applications. In *Proceedings of the 2007 ACM symposium on Applied Computing (SAC'07)*, pages 580–585. ACM Press, 2007.

[2] G. Biegel and V. Cahill. A framework for developing mobile, context–aware applications. In *Conference on Pervasive Computing and Communications PerCom 2004*, pages 361–365, Orlando, USA, March 2004.

[3] É. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. The FRACTAL Component Model and Its Support in Java. *Software—Practice and Experience, Special issue on Experiences with Auto-adaptive and Reconfigurable Systems*, 36(11):1257–1284, September 2006.

[4] T. Buchholz, A. Kupper, and M. Schiffers. Quality of context information: What it is and why we need it. In *10th International Workshop of the HP OpenView University Association*, Geneva, Switzerland, July 2003.

[5] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. A framework for dynamic adaptation of parallel components. In *Parallel Computing Conference (ParCo)*, 2005.

[6] Djalel Chefrour. Developing component-based adaptive applications in mobile environments. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1146–1150, Santa Fe, New Mexico, USA.

[7] Pedro J. Clemente, Juan Hernadez, Juan M. Murillo, Miguel A. Perez, and Fernado Sanchez. *Component-Based Software Development: Case Studies*, chapter Chapter 5 : Component-based System Design and Composition: An Aspect-oriented Approach, pages 109–128. World Scientific, 2004.

[8] D. Conan, R. Rouvoy, and L. Seinturier. Scalable Processing of Context Information with COSMOS. In J. Indulska and K. Raymonds, editors, *Proc. 6th IFIP WG 6.1 International Conference on Distributed Applications and Interoperable Systems*, volume 4531 of *Lecture Notes in Computer Science*, pages 210–224, Paphos, Cyprus, June 2007. Springer-Verlag.

[9] Olivier Dalle. Component-based discrete event simulation using the fractal component model. In *AI, Simulation and Planning in High Autonomy Systems (AIS) - Conceptual Modeling and Simulation (CMS) Joint Conference*, pages 213–218, Buenos Aires, AR, February 2007.

[10] Pierre-Charles David and Thomas Ledoux. Towards a framework for self-adaptive component-based applications. In Jean-Bernard Stefani, Isabelle Demeure, and Daniel Hagimont, editors, *Proceedings of Distributed Applications and Interoperable Systems 2003 DAIS2003)*, volume 2893 of *Lecture Notes in Computer Science*, pages 1–14, Paris, 2003. Federated Conferences, Springer-Verlag.

[11] M. Oussalah Gautier Bastide, A.-D. Seriai. Self-adaptation of software component structures in ubiquitous environments. In *ICPS'08: Proceedings of the International Conference on Pervasive Services*, Juillet 2008.

[12] J.O. Kephart and D.M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1), January 2003.

[13] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel. Weaving executability into object-oriented meta-languages. In S. Kent L. Briand, editor, *MODELS/UML'2005*, pages 264–278, Montego Bay, Jamaica, 2005. Springer.

[14] Nicolas Pessemier, Lionel Seinturier, Thierry Coupaye, and Laurence Duchien. A model for developing component-based and aspect-oriented systems. In *5th International Symposium on Software Composition*, Vienna, Austria, 2006.

[15] R. Rouvoy, D. Conan, and L. Seinturier. Software Architecture Patterns for a Context Processing Middleware Framework. *IEEE Distributed Systems Online*, 9(6), June 2008.

[16] Maria-Teresa Segarra and Françoise André. A framework for dynamic adaptation in wireless environments. In *TOOLS '00: Proceedings of the Technology of Object-Oriented Languages and Systems (TOOLS 33)*, page 336, Washington, DC, USA, 2000. IEEE Computer Society.

[17] Davy Suvée, Bruno De Fraine, and Wim Vanderperren. *Component-Based Software Development*, volume 4063 of *Lecture Notes in Computer Science*, chapter A Symmetric and Unified Approach Towards Combining Aspect-Oriented and Component-Based Software Development, pages 114–122. Springer Berlin / Heidelberg, 2006.

[18] S.S. Yau, F. Karim, Yu Wang, Bin Wang, and S.K.S Gupta. Reconfigurable context-sensitive middleware for pervasive computing. 1(3):33–40, June 2002.

[19] Ji Zhang and Betty H. C. Cheng. Model-based development of dynamically adaptive software. In *IEEE International Conference on Software Engineering (ICSE06)*, Shanghai, China, May 2006.