Reconciling Context-Oriented Programming and User Interface Adaptation

Benoît Duhoux
Université catholique de Louvain
Louvain-la-Neuve, Belgium
benoit.duhoux@uclouvain.be

ABSTRACT

Context-oriented programs take into account detected contexts to alter an application so that it exhibits the most appropriate behaviour for that particular context. However Context-Oriented Programming (COP) focuses mostly on adapting behavioural aspects while mostly ignoring adaptation of the user interface aspect. In the HCI community, on the other hand, much research has been done on user interface adaptation but without paying much attention to the behavioural aspect. This PhD work seeks to reconcile both communities to allow programmers to build realistic applications that are more sensitive to their surrounding environment, as well as to evaluate the user acceptance of such systems.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing systems and tools**;

EICS '18, June 19–22, 2018, Paris, France

ACM ISBN 978-1-4503-5897-2/18/06.
https://doi.org/10.1145/3220134.3220150

KEYWORDS

Context-Oriented Programming; User Interface Adaptation; Programmers as Users

ACM Reference Format:
Benoît Duhoux. 2018. Reconciling Context-Oriented Programming and User Interface Adaptation. In *EICS '18: EICS '18: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, June 19–22, 2018, Paris, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3220134.3220150

INTRODUCTION

A lot of information about the surrounding environment of applications is available but little of it is exploited to refine and customize the behaviour of applications to provide the most appropriate services according to their surrounding environment. This environment (represented as a set of "contexts") is derived partly from user settings such as the preferred language, from internal sensors of the computing device such as the battery level, and from external information such as the location, actual wheather conditions and so on [1]. With contemporary programming technologies, it is not easy to create such context-aware systems due to the high-level of dynamicity and exponential combination of features they can offer. Context-Oriented Programming (COP) attempts to solve this problem [6].

Even if the software engineering community has widely explored COP [11], and the HCI community does the same for user interface adaptation research [4, 8, 12], few interactions between both are present in literature [5]. It seems interesting to reconcile these two domains. This is a main objective of our research. Another objective is to develop supporting interactive tools to help programmers build and debug context-oriented systems. Finally, we are interested in evaluating some user acceptance aspects of context-aware systems with real end-users.

In the following section, we describe the case study of a risk information system to illustrate what we mean by context-oriented applications. Then we explain the objectives of our research and finally conclude with our current progress.

CASE STUDY

Before explaining our research and its objectives, we exemplify a context-aware application to give the reader a better intuition of such applications. Our case study example is inspired by two Belgian applications: *INFO-R!SQUES.be*[1] and *.be alert*[2]. The former is a web application providing information to citizens about the actions they need to take before, during and after a risk (e.g., an earthquake or a heatwave). These instructions vary with the type of persons (children, adults, elderly persons) and their location (for instance if an adult is inside his car). The latter application is an alert system to which citizens can subscribe to receive notifications with general instructions by sms, email and so

[1] http://www.info-risques.be

[2] http://be-alert.be/

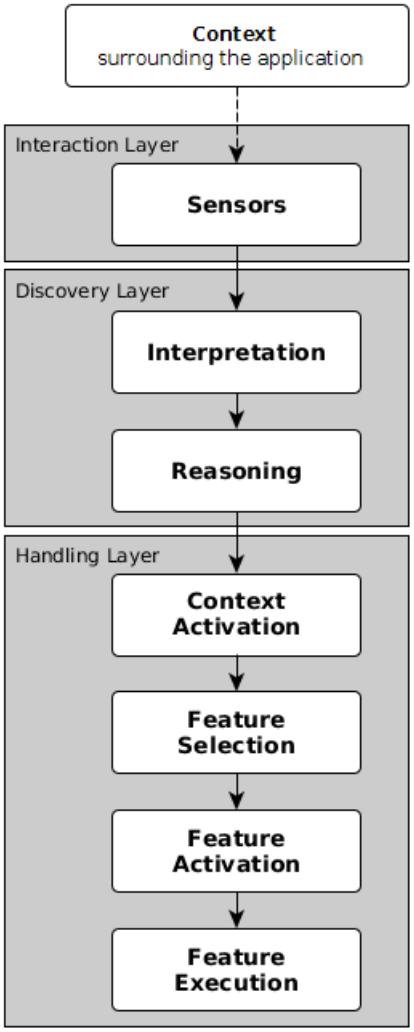

Figure 1: Overview of our context-oriented software architecture. For more details, see [9].

on, when a risk is detected. However, both systems have their limitations. *INFO-R!SQUES.be* gives a lot of information but doesn't adapt it dynamically to help citizens in urgent need. While *.be alert* is more dynamic but doesn't directly target the concerned persons, or floods the citizens with information when several risks appear simultaneously. Despite their limitations, the two systems are complementary, which motivated us to conceive a single solution with a high-level of dynamicity to display only the needed information to help citizens in crisis situations. In addition to these features, we extend the application with a feature guiding the citizens to a safe place through either a navigable map when they have a good connectivity and a high battery level or rather a textual navigation when the connectivity is weak or the battery level is low. This case study is an example of a context-oriented application that can adapt its behaviour at runtime when contexts change. To build such applications, we need to use context-oriented programming tools and technologies, such as those explained in the next section.

RESEARCH OBJECTIVES

The primary goal of our research is to design an environment to help programmers build context-oriented applications. A first objective towards reaching that goal is to create an architecture for building such systems that can reconcile the behavioural and user interface adaptations. A second objective is to provide interactive tool support to help programmers in the creation of such applications. Finally, we want to study the user acceptance of such context-oriented systems. In this section, all given examples are directly inspired from our case study above.

Reconciling contexts and features

To build context-oriented software systems, we need frameworks and/or languages enabling a high-level of dynamicity required to adapt the behaviour of such systems when contexts change. This research question has been explored since ten years by the software engineering community [11]. In these previous studies, each behavioural adaptation is attached to a context in order to execute it when this context gets activated. For example, assume an earthquake is detected and the user needs to receive dedicated instructions about this, this information is immediately adapted as soon as the context `Earthquake` gets activated. However, in COP we do not have a clear separation of concerns yet between the contexts and their corresponding behavioural adaptations. A first contribution of our work is to increase this separation between contexts and their corresponding features (previously called behavioural adaptations), by exploiting some of the similarities between COP and Feature-Oriented Programming (FOP) [3]. We created a software architecture [9] that reconciles contexts and features, as depicted in Figure 1. When a sensor detects a new value, a message is sent to the Interpretation component to interpret it and then the Reasoning component may reify this new message into a potential context object. For example, when the battery level is less than 20%, a context `LowBattery` is

reified. The Context Activation component will then verify whether this context can be (de)activated taking into account the possible dependencies among contexts. For instance, some contexts can be declared as mutually exclusive (e.g., `LowBattery` and `HighBattery`). Once the contexts have been (de)activated, the Feature Selection component picks which features must be (un)installed based on a declared context-feature mapping (e.g. the context `LowBattery` implies the installation of the feature `TextualNavigation`). After this selection of features, the Feature Activation component actually (de)activates the selected features but again taking into account existing dependencies with already installed ones. Finally after the features have been (de)activated, the Feature Execution component will actually deploy them dynamically to alter the behaviour of the application. So far, however this architecture was implemented only for handling the behavioural adaptations. What remains to be done is to integrate the user interface adaptation in a similar way, explained in the next objective.

Reconciling behavioural and user interface adaptations

As stated before, for now we considered a feature mostly as a behavioural entity. But this definition could be extended as suggested by Kang et al. They define a feature as *"a prominent or distinctive user-visible aspect, quality, or characteristic of a software sytem or systems"* [7], i.e. a functionality directly relevant to the user. So, we could consider that a feature also comprises user interface aspects. In the HCI community, we can relate this notion of feature to a user task model [10]. Task models have been used in HCI to help define user interface adaptive systems as exemplified by UsiXML [8] and COMET [4], which are both based on the Cameleon reference framework [2]. This framework proposes a four layered approach. To generate adaptive user interfaces, developers first define the tasks and concepts of the system, in order to design the different user interactions with the system. The user tasks are then transformed into abstract user interfaces (AUI), which are still independent of the concrete interaction modalities. After abstracting the user tasks, a model then allows the creation of more concrete user interfaces (CUI), which are interactor-dependent but not dependent on the platform. To obtain the final user interfaces (FUI) specific to the platform, a last transformation is applied to the CUIs. However, such generative layered approaches make it complex to generate user interfaces on the fly because when a new target platform or modification is defined, the whole generation process typically needs to be run again. This motivates us to explore a more flexible alternative for user interface adaptation. Modelling user adaptation in a similar way as behavioural adaptation, we could reconcile both adaptations in a same context-oriented architecture [9]. Even if our approach currently focuses mostly on behavioural components, nothing in the architecture prohibits those components to comprise user interface fragments that could thus be composed or altered on the fly depending on the current context.

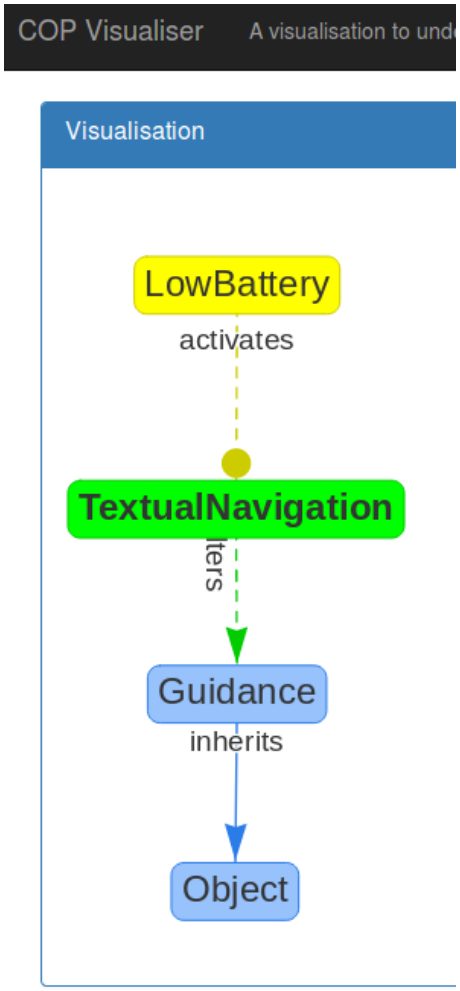

Figure 2: Screenshot of the COP visualization tool.

[3]https://www.nngroup.com/articles/ten-usability-heuristics/

Programmers as users: getting the right tools

Building, debugging or simulating context-aware applications is not easy for programmers due to their high-level of dynamicity. To provide them with the right support tools is thus an interesting challenge. We already created such a tool for the visualization of active entities (i.e., the active contexts, the installed features and how and where they interact with the application) for the implementation and debugging phase as illustrated by Figure 2. Another useful tool would be an interactive IDE helping programmers to create the contexts, the features and their mappings in a more visual way. As we suggested earlier [5], it would also be useful to have a simulator to simulate such systems and their surrounding environments. To prototype and assess these tools, we will conduct several evaluations with programmers to evaluate the comprehensibility and usefulness of our tools.

Getting the end-user perspective: user acceptance

Due to the non-static nature of the interaction they provide, at times, context-oriented systems may seem awkward to use by end-users. For example, when a user is using a navigable map to reach a safe destination during an earthquake and low battery is detected, the visual map may get substitued by a textual navigation mode. In such cases, that user should be gracefully informed about this substitution and should even have the possibility to revert to the previous behaviour, if he desires so, as suggested by Nielsen´s heuristics[3]. As part of the prototype systems we will build we will start to assess such user acceptance issues with actual end-users.

CURRENT PROGRESS

I developped a first prototype of the implementation of our COP architecture (Fig. 1), a first prototype of the visualization tool (Fig. 2) for programmers, as well as a COP-based language with a clear separation of contexts and features.

ACKNOWLEDGMENTS

I would like to thank prof. Kim Mens (UCL) and prof. Bruno Dumas (UNamur) for their constructive reviews of this paper.

REFERENCES

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. 1999. Towards a Better Understanding of Context and Context-Awareness. In *Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99)*. Springer, 304–307.

[2] Gaëlle Calvary, Joëlle Coutaz, David Thevenin, Quentin Limbourg, Laurent Bouillon, and Jean Vanderdonckt. 2003. A unifying reference framework for multi-target user interfaces. *Interacting with Computers* 15 (2003), 289–308.

[3] Nicolás Cardozo, Sebastian Günther, Theo D´Hondt, and Kim Mens. 2011. Feature-Oriented Programming and Context-Oriented Programming: Comparing Paradigm Characteristics by Example Implementations. In *International Conference On Software Engineering Advances (ICSEA'11)*. IARIA, 130–135.

[4] Alexandre Demeure, Gaëlle Calvary, and Karin Coninx. 2008. *COMET(s), A Software Architecture Style and an Interactors Toolkit for Plastic User Interfaces*. Springer, 225–237.

[5] Benoît Duhoux. 2016. *L´intégration des adaptations interfaces utilisateur dans une approche de développement logiciel orientée contexte*. Master's thesis. Université catholique de Louvain, Belgium.

[6] Robert Hirschfeld, Pascal Costanza, and Oscar Nierstrasz. 2008. Context-Oriented Programming. *Journal of Object Technology, March-April 2008, ETH Zurich* 7, 3 (2008), 125–151.

[7] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon University.

[8] Quentin Limbourg, Jean Vanderdonckt, Benjamin Michotte, Laurent Bouillon, Murielle Florins, and Daniela Trevisan. 2004. UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces. In *Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages"*. ACM, 55–62.

[9] Kim Mens, Nicolás Cardozo, and Benoît Duhoux. 2016. A Context-Oriented Software Architecture. In *Proceedings of the 8th International Workshop on Context-Oriented Programming (COP'16)*. ACM, 7–12.

[10] Fabio Paternò, Cristiano Mancini, and Silvia Meniconi. 1997. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. In *Human-Computer Interaction INTERACT '97*. Springer, 362–369.

[11] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. 2012. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software* 85, 8 (2012), 1801 – 1817.

[12] David Thevenin and Joëlle Coutaz. 1999. Plasticity of User Interfaces: Framework and Research Agenda. In *Interact'99*, Vol. 99. Press, 110–117.