Implementing Self-Adaptability in Context-Aware Systems ⋆

Boris Mejías[1] and Jorge Vallejos[2]

[1] Université catholique de Louvain, Louvain-la-Neuve, Belgium
boris.mejias@uclouvain.be
[2] Vrije Universiteit Brussel, Brussels, Belgium
jvallejo@vub.ac.be

1 Introduction

Context-awareness is the property that defines the ability of a computing system to dynamically adapt to its context of use [1]. Systems that feature this property should be able to monitor their context, to reason about the changes in this context and to perform a corresponding adaptation. Programming this three activities can become cumbersome as they are tangled and scattered all over in the system programs.

We propose to model context-aware systems using feedback loops [2]. A feedback loop is an element of system theory that has been previously proposed for modelling self-managing systems. A context-aware system modelled as a feedback loop ensures that the activities of monitoring, reasoning and adapting to the context are modularised in independent components. In this work, we take advantage of such modularisation to explore different programming paradigms for each component of the loop.

2 Feedback Loops for Self-Adaptable Context-Aware Systems

Modelling software systems using feedback loops implies for the developers to identify which kind of information needs to be monitored, dedicating particular agents for this task. Once the monitored information is collected, another component is in charge of deciding correcting actions, using an actuator agent to apply the corrections to the system.

Consider the case of a computer-assisted system for managing the lights of a so called *intelligent house*. This system consists of a set of lights and sensors that detect the presence of people in the house. The detection of a person is monitored by a specialised component that decides whether to turn on or off the lights, or simply modify their intensity. The loop is depicted at the left side of figure 1.

Since the use of mobile devices such as phones, PDAs, media players or GPSs are becoming very common, we can expect that users will use her/his mobile device to communicate with the house. We also expect that these devices can adapt their behaviour according to their context. The context can represent locality, CPU use, battery load,

⋆ This work has been partially funded by the European projects EVERGROW and SELFMAN, and by the flemish project of Context-Driven Adaptation of Mobile Services (CoDAMoS).

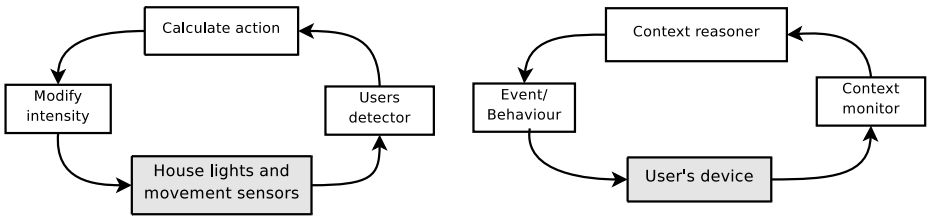

Fig. 1. Feedback loops modelling an automated light system and a context aware mobile device.

or a particular situation such as being busy, in a meeting, etc. The context is constantly monitored by a *context reasoner*, which decides the behaviour of the device in order to react to external events, or to trigger certain events to communicate with other devices.

These simple loops already provide self-adaptability to the house lights system and to the user's mobile device. The former adapts light's intensity according to the detection of users, and the later adapts its behaviour depending on the context. Consider now both models collaborating as a self-organising system. We first extend the house lights system to also monitor context. Having a context reasoner, lights are able to adapt their behaviour not only to users' movement, but also to particular context dependent scenarios. For instance, you do not want to turn on the lights and wake up the kids when they are in the sleeping context. We also add other sensors in order to receive message from users' devices.

Figure 2 depicts the interaction between both loops. User's device monitors the intensity of the lights while still monitors context. Being in the context of *arriving home* may triggers an event to turn on the lights. The context *watching a film* with *high light intensity* may triggers the event of lowing the intensity of the lights.

Since the house lights system is enriched with a context reasoner, some events triggered from user's device may not have always the same result. For instance, turning on the lights when arriving home may not work as expected if kids are in the sleeping context. Like this, two users can communicate through the lights systems as stigmergy. We can also observe that sensors and lights serve as stigmergy for the communication of user's device, and the controller of the house, because both of them monitor the system, and trigger events to modify the intensity of lights.

3 Implementing Feedback Loops

We have started to implement a prototype of the system using Mozart [3], a multi-paradigm programming system implementing the Oz language [4]. We have identified several ways of communicating components of a loop, which can be done using an event-driven approach, or stream communication, which can achieve by pulling or pushing information (lazy or eager execution). To communicate distributed components, message passing seems to be the most appropriated paradigm.

User's devices follow naturally the actor model [5], but inside the actor we can introduce other paradigms as well. For instance, the context reasoner applies a set of rules to

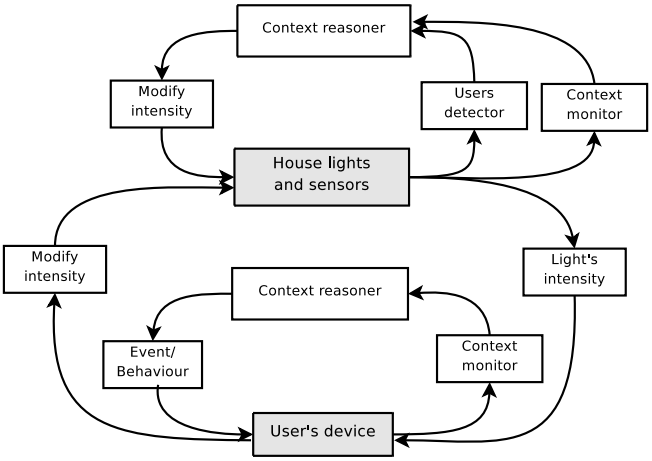

Fig. 2. Communicating two feedback loops.

the monitor information in order to determine the correspondent rule. This component fits better logic or declarative programming. To implement adaptive behaviour, we have chosen a model representing roles [6], where split objects [7] are used as the general architecture.

Since every component communicate with other by events or messages, they are quite independent, and the decision of the implementation of each of them, do no affect the implementation of the others. We still need to investigate more about the explicitness of the components matching the design and the implementation, because sometimes they appear clearly at the conceptual level, by they integrated to other components in the implementation.

References

1. Group, I.A.: Ambient intelligence: from vision to reality (2003)
2. Van Roy, P.: Self management and the future of software design. In: Formal Aspects of Component Software (FACS '06). (2006)
3. Consortium, M.: The mozart-oz programming system. *http://www.mozart-oz.org* (2007)
4. Van Roy, P., Haridi, S.: Concepts, Techniques, and Models of Computer Programming. MIT Press (2004)
5. Hewitt, C., Bishop, P., Steiger, R.: A universal modular actor formalism for artificial intelligence. In: Proc. of the 3rd IJCAI, Stanford, MA (1973) 235–245
6. Vallejos, J., Ebraert, P., Desmet, B., Cutsem, T.V., Mostinckx, S., Costanza, P.: The context-dependent role model. In Indulska, J., Raymond, K., eds.: 7th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS '07). Lecture Notes in Computer Science, Springer-Verlag (2007) 277–299
7. Bardou, D., Dony, C.: Split Objects: a Disciplined Use of Delegation within Objects. In: Proceedings of the 11th Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'96), San Jose, California, USA (1996) 122–137