Context Petri Nets: Definition, Analysis, Debugging and Execution of Context-aware Systems

Nicolás Cardozo1,2, Kim Mens1, Theo D’Hondt2

1 ICTEAM, Université catholique de Louvain
2 Software Languages lab, Vrije Universiteit Brussel
{nicolas.cardozo,kim.mens}@uclouvain.be, tjdhondt@vub.ac.be

1. POSITION STATEMENT

Context-Oriented Programming (COP) [8] is a programming paradigm that allows dynamic software adaptations to the surrounding execution environment of a system. Over the past few years effort has been set into the definition of programming languages and their abstractions that enable the definition of such dynamic adaptations.

All existing COP languages, through different abstractions and techniques, provide one way or another to reify the situations on the surrounding execution environment of the system, being the most prominent contexts and layers. This abstractions are usually first-class entities of the system to which partial behavior definitions can be associated. We regard contexts (or layers) together with such definitions by the more general term of adaptation. Adaptations are made dynamically available or unavailable in the system by means of the activation or deactivation constructs respectively. Whenever an adaptation is activated (in the dynamic scope of the activation construct or globally) the behavior defined for the adaptation can be used by the system. If an adaptation is deactivated, its associated behavior will no longer be visible in the system.

We have identified some issues that still need to be tackled by the COP community, and in our vision is holding it back from moving forward. First, the dynamicity with which adaptations are made available, based on the execution environment of the system, can rise inconsistencies in the system’s behavior. Different approaches have been proposed to mitigate behavior inconsistencies by defining dependency relations between adaptations [3, 4, 6, 7]. Such approaches allow to define constraints on interaction between adaptations when they are activated. However, none, or only manual verification of the interaction between adaptations is provided, which means that the definition of the interaction between adaptations needs to be foreseen before hand. Bearing in mind that COP systems may define multiple adaptations and that their activation occurs dynamically and unannounced, foreseeing all possible interactions between adaptations might be to cumbersome or even unfeasible in a system of considerable size.

We argue that to deal with such inconsistencies it is necessary, not only to be able to constraint interaction between adaptations, but also to provide the means to automatically verify that constraints imposed between adaptations are satisfied at run-time. Moreover, it should be possible to analyze the correctness and completeness of adaptation interaction.

Second, most COP languages have been developed as proof-of-concept prototypes, where only the basic features of COP are available. Integration between COP languages and their host language has not been a concern. This creates a difficult entrance barrier and acceptance of COP systems.

We believe that since the basic requirements of COP have been defined and extensively tested, the effort for the next generation of COP systems should concentrate on providing developers with the necessary tools to program, analyze and test their applications as an integrated COP tool suite.

2. CONTEXT PETRI NETS

As part of our research in context Petri nets (CoPN), we have been concentrating in these two points in order to improve the COP developers with the appropriate support to ensure that their applications are consistent and correct. With this objective in mind we have developed CoPN, a language independent model for context-oriented programming [9]. The CoPN model targets to be an integrated tool suite that allows the definition, analysis, debugging and execution of COP systems. We discuss the building blocks of our proposed tool suite in the following.

2.1 Definition

The CoPN model is a Petri net-based model that allows the definition of behavior adaptations with respect to the execution environment of a system. Taking from the formal semantics provided by Petri nets [2, 2, 1], adaptations can be easily expressed in CoPN.

CoPNs use the already existing language constructs for the definition of adaptations, dependency relations between adaptations and the dynamic activation of adaptations. These are translated in terms of a Petri net, and can be used as an underlying execution model of a COP system at run-time. The visual representation of an adaptation defined as a CoPN is shown in Figure [9].

CoPNs provide an explicit view on the different states in the life cycle of an adaptation activation (circles in Figure [9]), how many times has an adaptation been activated and in which state a context is (black dots in Figure [1]), and the available actions over an adaptation (squares in Figure [9]).
2.2 Execution
At run-time CoPNs allow the automatic verification of the interaction between adaptations. Given the dynamic characteristics of Petri nets used in CoPN it is possible to ensure that all constraints imposed by the interaction between adaptations are satisfied at all time. Even more, when an activation violates one of these constraints, the model is able to automatically backtrack the operation and provide feedback to the user with the cause of the constraint violation.

All this process is automatically build as part of the CoPN model. Developers do not need to manually define the interaction between adaptations or verify the consistency of the system. Rather, developers only use the already provided dependency relations within the CoPN model. Definition of the dependency relations is done by means of language abstractions as well as with the definition of adaptations or their dynamic activations. Developers do not need to interact directly with the CoPN model.

2.3 Debugging
The tool suite currently provides a visualization of the CoPN defined form the COP language constructs. The purpose of this visualization is to be able to activate the different contexts to check if their interaction is indeed the desired one. Clearly this process should be automatize as much as possible, though, developers should be given the opportunity to perform their own tests for the cases they are interested in. Manipulation and changes in the CoPN view, should also modify the definition of the constraints between adaptations.

3. FUTURE WORK
As mentioned before, it is not sufficient to provide means for the definition of the interaction between adaptations. It is also important to allow to identify the completeness and correctness of the interaction definitions. Luckily, the CoPN tool suite can also be used for such purposes. Here we present how CoPNs could be used for the identification of inconsistencies and the development of COP systems. This work is still in its preliminary stages.

3.1 Analysis
Petri nets analysis techniques could be used to analyze the system before hand. In this way via reachability analysis, for example, it could be possible to identify missing or unnecessary constraints between adaptations. Analysis and identification of inconsistencies in the system should be an automatic process.

3.2 Development
The CoPN tool suite could be used for programming COP systems themselves. That is, given the visual representation of the model, context-dependent behavior could be directly defined to the corresponding adaptation it belongs to. In such a way, the complete system can be programmed, analyzed, tested and executed in the same environment.

4. CONCLUSION
We believe that in order to push the COP paradigm forward into the mainstream, effort should be made into providing the appropriate programming support for developers. Providing a tool suite that allows to define behavioral adaptations to the execution environment of the system, and that ensures that the interaction between adaptations is correct and consistent.

We argue that throughout the CoPN model we are setting the first milestones to have such a tool suite. CoPNs already allow to automatically manage the interaction between adaptations at run-time. Additionally, the tool already provides a visual representation of the COP system that enables programmers to test and debug the interaction between adaptations before they are ever run.

The CoPN model and tool suite here presented are an initial step forward towards a new generation of dynamically adaptable software systems development.

5. REFERENCES