Managing Runtime Evolution in Dynamic Software Systems

Extended Abstract

Nicolás Cardozo[1,2], Sebastán González[1], Kim Mens[1], and Theo D'Hondt[2]

[1] ICTEAM Institute, Université catholique de Louvain
Place Sainte-Barbe 2, 1348 Louvain-la-Neuve, Belgium
[2] Software Languages Lab, Vrije Universiteit Brussel
Pleinlaan 2, 1050 Brussels, Belgium

In the context of mobile and pervasive computing [7] contextual information (like personalization, location, internal device's state, and on-spot environmental information) is becoming central to application development. Current-day systems are required to incorporate and react to contextual information, which emphasizes the growing importance of runtime software evolution [3]. To address this need, the context-oriented programming (COP) paradigm has been proposed to provide the ability of writing programs that can adapt, correct, or extend their behavior dynamically at runtime according to the surrounding execution environment. Different COP languages have been defined as either new languages or extensions of existing languages [1,4].

COP languages introduce different language abstractions that enable the definition, dynamic (de)activation, and composition of contexts and context-dependent behavior. Let us illustrate the idea by means of a particular COP language, *Subjective-C* [5], although similar concepts can be found in most COP languages. Contexts are defined as first-class program entities and are usually given by assigning a semantic meaning to internal/external characteristics of the environment (e.g., the GPS coordinates 50°51'0"N 4°21'0"E correspond to the `Brussels` context), which is defined as `@context(Brussels)`. Behavior is associated to a context by annotating partial method definitions with the corresponding contexts for which the method is applicable as follows, `@contexts Brussels -(void) getCoordinates{...}`. Such method definitions become available dynamically in the main application only if their context of definition is active. Contexts can be activated and deactivated dynamically, using respectively the `@activate(Brussels)` and `@deactivate(Brussels)` constructs.[3]

In the context of COP applications, activations and deactivations of contexts are assumed to happen concurrently and without warning, which may lead to incoherences or inconsistencies with respect to the expected application behavior. To deal with this, different proposals have been made to manage the definition and composition of dependencies between contexts [2,5]. Such proposals only provide models that constrain the dynamics of (de)activating a context, according to the state of its related contexts. Nonetheless, in most cases, the

[3] Activations and deactivations are triggered by a sensed change in the internal or external information.

corresponding language abstractions or runtime support for the model is not provided. Furthermore, the informal and high-level definition of such constraints makes their verification difficult and computationally expensive especially in the highly dynamic settings encountered in mobile and pervasive computing.

We propose to address the problem of consistency management in systems that dynamically evolve at runtime, along two fronts. First, to cope with the dynamic nature of COP systems, we propose a module for the precise definition and management of interaction between context dependencies. We use the Petri net [6] based formalisms for this. In addition to the advantages given by the formal definition, the model also provides a first-hand view on the dynamics and state of COP systems. Moreover, it serves as an underlying implementation in our context management system, thus providing a lightweight verification mechanism for the activation and deactivation of contexts.

Second, an static analysis module could be used to provide an upfront fine-grained reasoning about consistency and validity properties of the application. The introduced formalism of Petri nets already provides analysis and verification mechanisms that could be used to, for example, find whether an application may reach a conflicting configuration of active contexts. Alongside the analysis tools provided by Petri nets, another verification module could identify possible inconsistencies at the method level. Based on the static information for each method (contexts in which it may be applicable, and methods which it may call), an analysis can be performed to ensure that application behavior remains consistent whenever methods are switched. For example, in a localization application, detecting faulty message sends to `getCoordinates` on a GPS device, just as the location context changes from `GPS` to `Extrapolate`, where the method is not defined. This module thus gives meaningful feedback about potential errors, and different possibilities on how to solve them for example, by creating a dependency between the two contexts.

The modules proposed here constitute a clear step forward in the support for runtime evolution of COP applications. We provide a lightweight runtime system for managing dependencies between adaptations, and an upfront analysis to identify possible runtime conflicts at a fine-grained level of granularity. Other modules will be explore in the future, to widen the family of problems addressed.

References

1. Costanza, P., Hirschfeld, R.: Language constructs for context-oriented programming: an overview of ContextL. In: Proceedings of the Dynamic Languages Symposium. pp. 1–10. ACM Press (Oct 2005), collocated with OOPSLA'05
2. Desmet, B., Vallejos, J., Costanza, P., De Meuter, W., D'Hondt, T.: Context-oriented domain analysis. In: Modeling and Using Context. pp. 178–191. Lecture Notes in Computer Science, Springer-Verlag (2007)
3. Gabriel, R.P., Goldman, R.: Conscientious software. In: Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications. pp. 433–450. OOPSLA'06, ACM Press, New York, NY, USA (2006)

4. González Montesinos, S.: Programming in Ambience: Gearing up for dynamic adaption to context. Ph.D. thesis, Université catholique de Louvain (October 2008)
5. Gonzlez, S., Cardozo, N., Mens, K., Cdiz, A., Libbrecht, J.C., Goffaux, J.: Subjective-C: Bringing context to mobile platform programming. In: Proceedings of the International Conference on Software Language Engineering. Lecture Notes in Computer Science, vol. 6563, pp. 246–265. Springer-Verlag (2011)
6. Murata, T.: Petri nets: Properties, analysis and applications. Proceedings of the IEEE 77(4), 541 – 580 (April 1989)
7. Satyanarayanan, M.: Pervasive computing: Vision and challenges. IEEE Personal Communications 8(4), 10–17 (Aug 2001)