Summary of the Second Workshop on Domain-Specific Aspect Languages

Johan Fabry
INRIA Futurs - LIFL, ADAM Team
40, avenue Halley,
59655 Villeneuve d'Ascq, France
Johan.Fabry@lifl.fr

Damijan Rebernak
University of Maribor
Smetanova ulica 17
2000 Maribor, Slovenia
damijan.rebernak@uni-mb.si

Thomas Cleenewerck
Vrije Universiteit Brussel, PROG
Pleinlaan 2,
1050 Brussel, Belgium
tcleenew@vub.ac.be

Anne-Francoise Lemeur
LIFL, ADAM Team
40, avenue Halley
59655 Villeneuve d'Ascq, France
lemeur@lifl.fr

Jacques Noyé
Ecole des Mines de Nantes
4, rue Alfred Kastler, BP 20722
44307 NANTES Cedex 3, France
Jacques.Noye@emn.fr

Éric Tanter
DCC - University of Chile
Blanco Encalada 2120,
Santiago, Chile
etanter@dcc.uchile.cl

1. Introduction to the workshop

Although the majority of work in the AOSD community focuses on general-purpose aspect languages (e.g. AspectJ), seminal work on AOSD proposed a number of domain-specific aspect languages, such as COOL for concurrency management and RIDL for serialization, RG, AML, and others. A growing trend of research in the AOSD community is returning to this seminal work, as witnessed by the high attendance rate at the DSAL06 workshop, held as part of GPCE06/OOPSLA06.

The workshop aimed to bring the research communities of domain-specific language engineering and domain-specific aspect design together. In the previous successful edition we approached domain-specific aspect languages from a language implementation point of view, where advances in the field of domain-specific language engineering were investigated to answer the implementation challenges of aspect languages. In this second edition, we approached the design and implementation of new domain-specific aspect languages, as well as the composition at all levels (from design to implementation) of these languages or individual features.

Workshop DSAL '07 March 12, 2007 Vancouver, British Columbia, Canada.

The workshop sought contributions related to domain-specific aspect languages, more particularly (but not limited to):

- design of DSALs
- successful DSALs and their applications
- issues in both design and implementation of DSALs
- methodologies and tools suitable for creating DSALs
- mechanisms for interaction detection and handling in DSALs
- theoretical foundations for DSALs
- analysis about the specificity spectrum in aspect languages
- key challenges for future work in the area

The workshop was comprised of two parts: paper presentation sessions and freeform discussion sessions. The former were held in the morning, the latter in the afternoon.

2. Contributions

The papers presented at the workshop were the following:

- ERTSAL: A Prototype of a Domain-Specific Aspect Language for Analysis of Embedded Real-Time Systems, by William L. Sousan, Victor Winter, Mansour Zand and Harvey Siy
- A Distribution Definition Language for the Automated Distribution of Java Objects, by Paul Soule, Tom Carnduff and Stuart Lewis

- ReLAx: Implementing KALA over the Reflex AOP Kernel, by Johan Fabry, Éric Tanter and Theo D'Hondt
- ALPH: A Domain-Specific Language for Crosscutting Pervasive Healthcare Concerns, by Jennifer Munnelly and Shiobán Clarke
- Aspect Oriented DSLs for Business Process Implementation, by Arno Schmidmeier

Each of these papers is also contained in this workshop proceedings volume.

3. Discussions

3.1 Definitions

The discussion in the afternoon started by seeking a definition of the term Domain-Specific Aspect Language. The participants agreed that a suitable definition would be the following: **A DSAL is a domain-specific language that is used to express a concern that cuts across multiple concerns.** Furthermore a defining property of a DSAL is **The use of a DSAL in programming an application invasively changes the behavior or structure of other modules of the application.** This can be achieved, e.g., by inserting code in these modules. Also, it was deemed that the DSAL programmer is not required to know about the cross-cutting nature, nor to know that the DSAL makes use of some form of aspects as an implementation strategy.

The above definitions then raised the question of join-point models for such languages. Some participants expressed their reservations with giving these domain-specific join-point models the name 'join-point models' as there is a strong implicit connection to the general-purpose join-point model. Other participants argued that a domain-specific join-point model can be considered as a specialization of a hypothetical 'most general' join-point model, and that therefore the naming should be kept. It was decided that the term 'join-point model' was indeed appropriate.

3.2 DSAL Infrastructure

A second topic of discussion was whether it is possible to create an infrastructure for the definition of DSALs that is general enough to enable a wide variety of DSALs to be implemented, and whether a set of design criteria can be established for such an infrastructure. It was established early on that due to the wide variety of domains, one general infrastructure would be difficult to make. As examples of this wide variety were considered the KALA DSAL as discussed in the ReLAx presentation versus an example DSAL that would manipulate the dynamic representation of web-pages. It seemed initially that because of this wide variety of domains it would also be impossible to establish design criteria beyond the use of common sense guidelines. However, by considering the differences between DSLs and DSALs, and the definition of a DSAL previously established, a number of criteria could be established.

Design Criteria for a DSAL infrastructure are:

- it should enable the easy creation of a DSL translator and other language-based tools such as, e.g., debuggers
- it should take care of exposing join-points to the DSAL
- it should be open, to enable new join-points to be exposed
- it should provide for a form of conflict detection and resolution, both for conflicts within programs in one DSAL as conflicts for programs written in multiple DSALS.

The first of these criteria is due to the fact that a DSAL is a DSL, and the remainder of the criteria are due to the cross-cutting nature of DSALs. The discussion closed with a brief look at the design of DSALs themselves. The participants agreed that well designed DSALs are at the core well designed DSLs and therefore require largely the same design criteria.

3.3 Use Of DSALs

Lastly, the use of DSALs was discussed. It was hypothesized that there could be a link between horizontal and vertical DSALs and asymmetrical and symmetrical AOP.

Vertical DSALs are created for a vertical domain, i.e., domains that correspond with a particular kind of applications: banking, e-commerce, et cetera. Horizontal domains, in contrast, span multiple vertical domains. Examples are transaction management, graphical user interfaces, and so on.

The hypothesis is that horizontal DSALs would have an asymmetrical nature, and vertical DSALs would have a symmetrical nature. Verifying this hypothesis was considered as an interesting avenue for future work. Considering the types of users for DSALs, the workshop concluded that the target audience for DSALs would be the average programmers, while general-purpose aspect languages should be reserved for power-users.

Acknowledgments

The organizers wish to thank the workshop attendants for an enjoyable and productive workshop.