Summary of the Third Workshop on Domain-Specific Aspect Languages

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

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

Johan Fabry
PLEIAD Lab
Computer Science Department
(DCC)
University of Chile
jfabry@dcc.uchile.cl

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

Éric Tanter
PLEIAD Lab
Computer Science Department (DCC)
University of Chile
etanter@dcc.uchile.cl

1. Introduction to the workshop

The tendency to raise the abstraction level in programming languages towards a particular domain is also a major driving force in the research domain of aspect-oriented programming languages. As a matter of fact, pioneering work in this field was conducted by devising small domain-specific aspect languages (DSALs) such as COOL for concurrency management, RIDL for serialization, RG, AML, and others. After a dominating focus on general-purpose languages, research in the AOSD community is again taking this path in search of innovative approaches, insights and a deeper understanding of fundamentals behind AOP. Based on the successful DSAL06 and DSAL07 workshops, and the special issue of IET Software journal on Domain-Specific Aspect Languages, this workshop series continues to support a growing trend in AOSD research.

The workshop aims to bring the research communities of domain-specific language engineering and domain-specific aspect design together. In the previous successful editions held at GPCE06/OOPSLA06 and AOSD07 we approached domain-specific aspect languages both from a design and a language implementation point of view. New for this edition is that we explicitly invited contributions of work on adding domain-specific extensions (DSXs) to general-purpose aspect languages (GPALs). The focus on language embedding raises specific issues for language designers, such as proper symbiosis between, and composition of, DSXs.

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

- design of DSALs and DSXs
- successful DSALs, DSXs and their applications
- issues in both design and implementation of DSALs and DSXs
- methodologies and tools suitable for creating DSALs and DSXs
- semantics and composition of DSALs and DSXs
- disciplined approaches for invasive metaprogramming
- error reporting in DSALs and debugging of DSALs
- approaches for composable language embeddings
- 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 remainder of this workshop summary is structured as follows. Section 2 gives an overview of all the papers presented at the workshop, which are divided in two groups: design and implementation of DSALs, and experience reports.

Workshop DSAL '08 April 1st, 2008 Brussels, Belgium.

The workshop discussions are summarized in Section 3. In the three main discussions we defined what constitutes a DSAL, we investigated how domain-specificness surfaces in DSALs and what design criteria are used when an implementation approach for DSALs is selected.

2. Presented Papers

The papers which were presented at the workshop can be divided into two groups.The first group of papers deals with issues in the design and implementation of DSALs and DSXs, while the second group discusses about experiences of designing particular DSALs and DSXs.

2.1 Design & Implementation Issues

Common to the papers in this group is they all are concerned with the issue of composability. This quality is approached from different perspectives ranging from a single to multiple DSALs and from an implementation point of view to a language design point of view. Each paper offers a tool in which a disciplined approach for invasive metaprogramming is used to design and implement DSALs and DSX.

- "Prototyping and Composing Aspect Languages using an Aspect Interpreter Framework" by Wilke Havinga, Lodewijk Bergmans, Mehmet Aksit [HBA08].
  This invited talk investigates the composability among different DSALs by offering an aspect interpreter framework. The challenge is to offer support that is general enough to express the DSALs and their interactions.
- "Modularizing Invasive Aspect Languages" by Thomas Cleenewerck, Theo D'Hondt (Technical Paper) [CD08].
  In this paper composability of domain-specific aspect language constructs of a single DSAL is investigated by focussing on the modularization of the implementation of each language construct. The challenge of modularizing the constructs lies in expressing the composition of the invasive semantics of each language construct in a modular fashion.
- "Dynamically Linked Domain-Specific Extensions for Advice Languages" by Tom Dinkelaker, Mira Mezini (Technical paper) [DM08].
  Composability is approached in this paper from the extensibility point of view. An implementation technique is demonstrated in which specific DSX can be easily defined to serve as advice languages in aspect languages. The challenge is to embed the language extensions and coordinate their execution.
- "A DSL to Declare Aspect Execution Order" by Antoine Marot, Roel Wuyts (Short paper) [MW08].
  One particular facet of composability is the execution order. In this position paper, the authors identified some problems and postulated a possible solution for the design of DSALs that should improve composibility.

2.2 Design & Implementation Experience

Experiences drawn from the designing of particular DSALs and DSXs have again proven to be very valuable for assessing the pros and cons of DSALs and DSX and for driving the workshop discussions.

- "Towards a DSAL for Object Layout in Virtual Machines" by Stijn Timbermont, Bram Adams, Michael Haupt (Short paper) [TAH08].
  This position paper argues in favor of the design of a DSAL to handle the tangled object layout concern in virtual machines. The language being proposed explores the boundaries of DSALs as it does not prevent the scattering but rather allows modular reasoning over scattered code fragments.
- "Towards a Domain-specific Aspect Language for Leasing in Mobile Ad hoc Networks" by Elisa Gonzalez Boix, Thomas Cleenewerk, Jessie Dedecker, Wolfgang De Meuter (Short paper) [GBCDM08].
  This paper motivates that leasing code in distributed applications is a complex crosscutting concern, consisting of many subconcerns which, in turn, are also tangled and scattered. The authors present a DSX rather than a new DSAL and shows that event-based AOP (a natural fit in an asynchronous computational model) introduces challenging design issues.
- "A Domain-specific Language for Parallel and Grid Computing" by João L. Sobral, Miguel P. Monteiro (Short Paper) [SM08].
  The DSAL presented in this paper aims to promote the localization of parallelization and gridification issues into well-defined modules. Interestingly, it also contains the main motivations for implementing the DASL on top of AspectJ.

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

3. Discussions

The workshop hosted three main discussions. First, we defined what constitutes a DSAL, second, we investigated how domain-specificity surfaces in DSALs and, third, we listed the design criteria that are used when an implementation approach for DSALs is selected.

3.1 Defining DSALs

There were a couple of papers that challenged the boundaries of what constitutes a DSAL and what not. Timbermont et.al [TAH08] argued that that DSALs can be used to merge and reason about domain-specific object layout descriptions of VMs. This language does not have pointcuts and does not separate the concern. So the question, also stated by the authors, is: when is a DSAL not a just a plain DSL? A similar question was raised through the work of Dinkelaker and

Mezini [DM08] where DSLs are used to define advices of aspects. In contrast, Gonzalez et.al [GBCDM08] based their language on general-purpose event-based AOP. So when is a DSAL not just a GPAL? Lastly, Sobral et.al. [SM08] implemented their DSAL on top of AspectJ. Does this imply that these DSAL programs, from an implementation point of view, are no longer crosscutting in nature?

These questions led to quite vivid discussions on what is a DSAL. The outcome of this discussion led to the following definition of a DSAL.

Conjecture 1. *A DSAL is a DSL for expressing crosscutting concerns, more formally a DSL whose programs are non-functionally composed with other programs.*

In function(al) composition defined by the mathematical operator $\circ$, values are provided to a unit of computation (i.e. a module, a function, etc.) and are subsequently processed in order to produce several other values. Computations are thus solely parameterized. In non-functional composition it is not sufficient to only pass values to change the outcome of the computation but the computation itself needs to be changed. Aspects are an example of this style. They cannot be composed in a "mere" functional style, because they need to change the behavior of other computations.

The main characteristic of the implementation techniques for aspects is that they all operate at the meta-level in order to change the behavior of other modules. In the discussion some border cases were discussed, e.g. monads. Monads capture a frequently reoccurring pattern in functional programs to control the composition of functional computations. It has been shown that monads are expressive enough to implement aspects [DM97]. Moreover, monads are implemented with higher-order functional programming. So, monads seem to implement aspects by *"mere"* functional composition. However, this is not entirely true because in order to benefit from the improved composition possibilities provided by monads, programs need to be written in monadic style. More concretely, a program needs to be rewritten using the monad abstraction so that it can be composed with monads implementing extensions [Wad92]. Monads thus introduce an extra "parameter" to the program, and are used to expose hooks within the execution of a functional program. By exploiting these hooks, aspects can be implemented as a monad that extends the execution of the base program. Clearly, although monads are functional, an additional meta operation is necessary to transform the programs involved in a composition.

Further elements of discussion were the following:

DSL From our conjecture of what constitutes a DSAL we can conclude that composition is an inherent and distinguishing characteristic of DSALs in comparison to DSLs. Despite the emphasis on composition with another program, the conjecture does not distinguish between symmetrical and asymmetrical AOP approaches. Like most GPALs, most DSALs are asymmetric. In this case, the program produced by a DSAL cannot be executed. However, a DSAL may also yield an executable program but which is then nevertheless composed with another program.

GPAL The conjecture does not explicitly mention GPALs. Instead we get a unambiguous definition of DSALs. Moreover, it avoids a number of confusions. The first confusion may arise when using a GPAL to implement the DSAL. In such a case, although a domain-specific aspect is compiled into a single module the DSAL nevertheless remains an aspect language dealing with a crosscutting concern. The second confusion may arise when comparing other common characteristics of GPALs with those of DSLs. One of them is declarativity. DSALs are not always more declarative. Like in DSLs declarativeness is not a necessary condition nor a sufficient one. The third and last remark is about the separation between pointcut and advice. In GPALs these are commonly very clearly separated. In DSALs this is not always the case. See the next section for a more elaborate discussion.

Inversion of Control Dependency inversion and dependency injection are patterns to invert the dependency relations between modules. However, in each of those patterns, modules are still parameterized with other dependent modules or execution control is directed by new modules.

Separation Separation of concerns is an important quality of GPALs. This is also the case for DSAL. It is implicitly part of the conjecture as an external specification is non-functionally composed with another module by changing the behavior of that module.

CSL Concern-specific languages (CSL) are often confused with DSALs. Like DSALs, concerns can be stand-alone programs. Unlike DSALs, concern-specific languages also encompass non crosscutting concerns which can be composed using functional composition techniques. This means that DSALs are CSLs but not the other way around.

Other relations with existing terminology and techniques may need to be investigated. One of them, which was identified, but not further discussed, is invasive composition.

3.2 Joinpoints, Pointcuts and Advice in DSALs

We observed that, in the past edition, including this one, only a small amount of work in the DSAL community has focused on domain-specific pointcuts. DSALs use either simple domain-specific pointcuts like in [SM08], or revert to GPAL solutions like in [DM08], or mix both like in [GBCDM08]. One exception this year was the work in [CD08] where pointcuts are defined in terms of the concepts from the do-

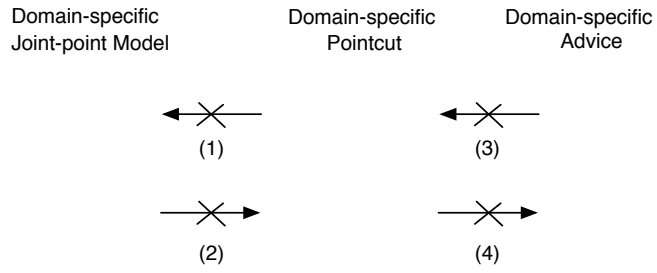

Figure 1. Overview of the argumentation for Conjecture 2.

main of the aspect language in order to modularly compose the semantics of DSAL constructs. Moreover, similarly to the early aspect languages such as COOL [Lop97], in some current DSAL proposals such as [TAH08] the separation between poincuts and advice is not very clear.

To summarize, the question is *how does the domain-specific nature surface in DSALs*. Concerning the relation of joinpoints, pointcuts and advice in DSALs it is our conjecture that:

Conjecture 2. *A DSAL must at least provide a domain-specific join point model, domain-specific pointcuts or domain-specific advice.*

Joint-point Model A joint-point model defines a set of execution events e.g. execution of a method within an object. In some cases, these models also capture the time when an event occurs e.g. before the execution of a method.

Pointcuts A pointcut is a predicate selecting a number of join points. Quantification is a part of a pointcut offering operators to specify the quantity of individual join points.

Advice An advice is the action that is taken when an event occurs. In case of execution events, advices also need to stipulate when they are executed e.g. before or after.

Figure 1 sketches an argumentation for the above conjecture. Each crossed arrow depicts a "does not imply" relationship. For example, a domain-specific pointcut does not imply a domain-specific join point model. Given all these relationships, we can conclude that a domain-specific joint-point model, pointcut or advice can independently be made domain specific from one another. Hence, it suffices for a DSAL to make at least one of these three domain specific.

These three parts of a DSAL can be independently made domain-specific because:

(1) The expression of a pointcut can be domain specific, while relying on a GPAL pointcut model. An example of this can be found in KALA [FTD08]. A KALA pointcut has the form of a method signature (with AspectJ-like wildcard support). KALA however uses the AspectJ joinpoint model: this pointcut identifies both the execution of the corresponding method, as well as calls to getter and setter methods in this method.

(2) GPAL predicates are domain independent. They can thus also be applied to domain-specific join-point models. For example, trivial sets can be computed like the entire set or the empty set. A less academic example is the selection of one join point by referring to its name e.g. in a simple DSAL for workflows a single action can be selected.

(3) Domain-specific advices are often used in conjunction with GPAL predicates. An example of such a language is COOL [Lop97].

(4) Predicates select elements from the domain-specific joint-point model. This does not imply any relationship to how the advice is specified.

Let us remark that during the workshop we *did not* discuss the relation between domain-specific advices and domain-specific joint-point models. In particular, does a domain-specific joint-point model imply a domain-specific advice?

3.3 Tool support

In the past, DSALs have been implemented from scratch by ad-hoc implementation approaches. It is only quite recently that implementation toolkits are becoming available for DSALs. We distinguish between three implementation approaches:

1. Implementations on top of or in a GPAL like AspectJ or extensible GPALs such as LMP [BMV02].
2. Implementations using general language development approaches such as Stratego [FTD08], JastAdd [AET08], LTS [CD08].
3. Implementations using dedicated toolsuites. Compile-time approaches such as Josh [CN04], AspectBench [ACH+05], load-time approaches such as Reflex [FTD08], or run-time approaches based on interpreters such as the ones presented by Havinga et al. [HBA08] and Dinkelaker and Mezini [DM08].

What conscious design criteria based on engineering qualities and implementation mechanisms are being used when choosing an implementation approach?

A somewhat remarkable result that surfaced during the discussions is that AspectJ is frequently chosen, despite its well-known limitations. The main reasons for this choice are that research results can be more easily communicated, and it is a robust and efficient tool that is well supported. The most important downside to AspectJ for DSAL development is its closed nature. This makes it hard to impossible to, e.g. have a domain-specific joinpoint model that requires joinpoints that are not captured by AspectJ.

AspectJ also serves as an easy reference to compare features. The downside is that a lot of approaches that tackled some its limitations are not always referenced to compare

features. In order to boost the research results of this community we concluded that alternative approaches should be more accessible. As such their correctness and effectiveness can be more easily and reliably checked. They should also be more flexible for extensions.

Acknowledgments

The organizers wish to thank the following workshop attendants for participating in the workshop: Elisa Gonzalez, Stijn Timbermont, Antoine Marot, Miguel P. Monteiro, Tom Dinkelaker, Wilke Havinga, João L. Sobral, Edgar Souse, Jenny Munnely, and Oscar Gonzalez.

References

[ACH+05] Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Jennifer Lhoták, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam, and Julian Tibble. ABC: an extensible AspectJ compiler. In *AOSD '05: Proceedings of the 4th international conference on Aspect-oriented software development*, pages 87–98, New York, NY, USA, 2005. ACM.

[AET08] Pavel Avgustinov, Torbjörn Ekman, and Julian Tibble. Modularity first: a case for mixing AOP and attribute grammars. In *AOSD '08: Proceedings of the 7th international conference on Aspect-oriented software development*, pages 25–35, New York, NY, USA, 2008. ACM.

[BMV02] Johan Brichau, Kim Mens, and Kris De Volder. Building composable aspect-specific languages with logic metaprogramming. In *GPCE '02: Proceedings of the 1st ACM SIGPLAN/SIGSOFT conference on Generative Programming and Component Engineering*, pages 110–127, London, UK, 2002. Springer-Verlag.

[CD08] Thomas Cleenewerck and Theo D'Hondt. Modularizing Invasive Aspect Languages. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[CN04] Shigeru Chiba and Kiyoshi Nakagawa. Josh: an open AspectJ-like language. In Gail C. Murphy and Karl J. Lieberherr, editors, *AOSD*, pages 102–111. ACM, 2004.

[DM97] Wolfgang De Meuter. Monads as a theoretical foundation for AOP. International Workshop on Aspect-Oriented Programming at ECOOP'97, 1997. ftp://prog.vub.ac.be/tech_report/1997/vub-prog-tr-97-10.pdf.

[DM08] Tom Dinkelaker and Mira Mezini. Dynamically Linked Domain-Specific Extensions for Advice Languages. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[FTD08] Johan Fabry, Eric Tanter, and Theo D'Hondt. KALA: Kernel aspect language for advanced transactions. *Elsevier Science of Computer Programming*, 2008. http://dx.doi.org/10.1016/j.scico.2007.10.004.

[GBCDM08] Elisa Gonzalez Boix, Thomas Cleenewerk, Jessie Dedecker, and Wolfgang De Meuter. Towards a Domain-Specific Aspect Language for Leasing in Mobile Ad hoc Networks. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[HBA08] Wilke Havinga, Lodewijk Bergmans, and Mehmet Aksit. Prototyping and composing aspect languages – using an aspect interpreter framework. In *Proceedings of the 22nd European Conference on Object-Oriented Programming, ECOOP08*, 2008.

[Lop97] Cristina Videira Lopes. *D: A Language Framework For Distributed Programming*. PhD thesis, College of Computer Science of Northeastern University, 1997.

[MW08] Antoine Marot and Roel Wuyts. A DSL to declare aspect execution order. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[SM08] João L. Sobral and Miguel P. Monteiro. A Domain-Specific Language for Parallel and Grid Computing. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[TAH08] Stijn Timbermont, Bram Adams, and Michael Haupt. Towards a DSAL for Object Layout in Virtual Machines. In *DSAL '08: Proceedings of the 3rd Workshop on Domain-specific Aspect Languages*, New York, NY, USA, 2008. ACM.

[Wad92] Philip Wadler. Comprehending Monads. *Mathematical Structures in Computer Science*, 2(4), 1992. (Special issue of selected papers from 6'th Conference on Lisp and Functional Programming.).