A programming language is a notation for expressing computations (algorithms) in both machine and human readable form. Appropriate programming languages and tools may drastically reduce the cost of building new applications as well as maintaining existing ones. In this regard, a domain-specific language (DSL), which is a language that provides constructs and notations tailored toward a particular application domain, is no exception. Usually, domain-specific languages are small, more declarative than imperative, and more attractive than general-purpose languages for their particular application domain because of:

- easier program understanding, writing, and reasoning,
- enhanced productivity, reliability, reusability, maintainability,
- easier verification,
- reduced semantic distance between the problem and the program.

Despite the fact that domain-specific languages have been developed from the beginning (an early example is APT, a DSL for numerical control of machine tools developed in the mid fifties at MIT), many unanswered questions remain regarding when and how to develop a DSL.

The purpose of this minitrack is to bring together an international audience of researchers and practitioners actively involved in the design, development and application of domain-specific languages. The first two papers derive from the ancestry of domain-specific languages in the programming language specification field. In each case a technique that has been previously applied to defining large scale programming languages is given a DSL twist.

João Saraiva and Sergio Schneider in “Embedding domain-specific languages in the attribute grammar formalism” apply the domain-specific language idea to attribute grammars. Their approach uses attribute grammars in a traditional manner to define DSLs. Higher-order attribute grammars are then used to apply the DSLs in other attribution.

Adam Granicz and Jason Hickey in “Phobos: A front-end approach to extensible compilers” show how term rewriting can be used to extend language definitions with domain-specific notations.

The next paper is based on XML, a technology that has rapidly gained acceptance as a general notation that can be specialised to represent structured data in a wide variety of domains. Hayato Kawashima and Katsuhiko Gondow in “Experience with ANSI C Markup Language for a cross-referencer” show how XML can be used to represent C programs for data interchange between CASE tools.

Perhaps the most important unsatisfied need at the current stage of DSL development and adoption is proper studies of their effectiveness. The last two papers take on this important task.

Jeff Gray and Gabor Karsai in “An examination of DSLs for concisely representing model traversals and transformations” evaluate DSLs for domain modelling. Comparisons of the size of the DSL code with the code generated from the DSLs are used to analyse the benefits of the approach.

David Wile in “Lessons learned from real DSL experiments” describes how a research and development organisation has learned from attempts to design and install domain-specific languages in three real projects. This paper stands out by virtue of its frank and revealing analysis. In particular, organisational and social issues are considered and their impact on DSL adoption is clearly demonstrated.