

Domain Specific Languages for Domain Specific Problems

Position Paper

Don Stewart
Galois, Inc.

<http://galois.com>
dons@galois.com

ABSTRACT

As the complexity of large-scale computing architecture increases, the effort needed to program these machines efficiently has grown dramatically. The challenge is how to bridge this “programmability gap”, making the hardware more accessible to domain experts. We argue for an approach based on executable *embedded domain specific languages* (EDSLs)—small languages with *focused expressive power* hosted directly in existing high-level programming languages such as Haskell. We provide examples of EDSLs in use in industry today, and describe the advantages EDSLs have over general purpose languages in productivity, performance, correctness and cost.

1. INTRODUCTION

Emerging hardware architectures require specialized programming techniques in order to realise their performance potential. These new architectures (such as GPGPUs or FPGAS) place great stress on existing general purpose programming languages, making code less portable, performance harder to achieve, and correctness increasingly difficult to assert. The domain expert’s ability to write and maintain scientific code must keep pace with these changing architectures. There is clearly a need for mechanized approaches to portability that will make tractable the task of effectively programming future hardware.

A proven technique for managing complexity, and gaining abstraction is through *domain-specific languages* (DSLs)—languages tailor-made to describe the concepts of a particular problem domain. Deursen et al give the following definition:

A domain-specific language (DSL) is a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on, and usually restricted to, a particular problem domain.

A good DSL accurately models the problem domain. This enhances productivity, maintainability, portability and correctness by hiding irrelevant detail. In addition, such abstraction makes explorative coding easier.

Most DSLs are declarative, and serve as executable specification languages. Their restricted focus can make them more amenable to automated analysis, optimization and verification than a general purpose language.

No single DSL will cover all problems faced by domain scientists. Instead, families of DSLs will be developed, each appropriate to its problem area. The approach of *embedded domain specific languages* (EDSLs) makes this task much

easier. By embedding the domain language within an existing general purpose language, we avoid the need to write a compiler and support tools from scratch, making it far cheaper to build the DSL.

Many EDSLs—and all of the ones we describe—are hosted in the functional programming language Haskell. A high-level functional language such as Haskell makes it easier to construct domain-specific functions, libraries, and even syntax, and purely functional code is easier to verify (see for example, seL4). By using the EDSL the domain expert can work at the right level of abstraction, while gaining access to the existing compiler, libraries and validation tools of the host language.

2. EDSLs IN PRACTICE

DSLs and EDSLs have been successfully used in many problem domains. The following are some examples:

The Cell: Anand et al describe an EDSL that allows mathematicians to formulate novel high-performance SIMD-parallel algorithms for the Cell processor, using a Maple-like EDSL embedded in Haskell.

GPUs: Multiple groups have developed EDSLs for programming GPU hardware at a high level, including Chakravarty et al. and Claessen et al.¹.

Cryptography: Galois has developed a DSL—Cryptol—for the specification of cryptographic algorithms, enabling cryptographic experts to write high performance VHDL, and verify functional correctness automatically.

Financial modelling: EDSLs are perhaps most widely used in the financial services industry. Several groups use EDSLs to generate modelling software, along with supporting documentation and other artifacts.

Operating Systems: The designers of Microsoft’s recent “Barrelfish” manycore operating system used a number of Haskell-based DSLs to generate parts of the kernel.

EDSLs allow us to work at a high level, and also take advantage of domain-specific features to generate efficient code. We believe they are a cost-effective technique for effectively programming emerging hardware.

About the Author

Don Stewart is a Senior R&D Engineer at Galois, Inc. specializing in functional languages, compilers and optimization. He is the coauthor of the textbook *Real World Haskell*.

¹Haskell also supports a wide range of parallel programming models, including speculative evaluation, nested data parallelism, and software transactional memory.