Aspect-Oriented Product Line Development in Constrained Environments

Danilo Beuche
pure-systems GmbH
Agnetenstr. 14
D-39106 Magdeburg
danilo.beuche@pure-systems.com

Olaf Spinczyk
University of Erlangen-Nuremberg
Martensstr. 1
D-91058 Erlangen
olaf.spinczyk@informatik.uni-erlangen.de

1. INTRODUCTION

Software in constrained environments has to cope with hard limitations on RAM/ROM, processing power, and other resources. Adhoc implementations in this area often suffer from a very low degree of reusability, because versatile software tends to consume significantly more resources. The concept of software product lines promises to provide efficiency *and* reusability. However, adequate tools for variant management are still rare.

Here we will shortly describe Pure::Consul [1], a tool which supports the description of problem and solution domains of product lines, software families, or other variable artifacts in a highly flexible manner. We applied Pure::Consul for the development of a weather station product line running on a small 8 bit microcontroller with only a few KBytes of memory.

The implementation of this product line was based on AspectC++, an aspect-oriented extension to C++ [3]. We will show that by applying aspect-oriented software development, the number of configuration points in the code can be reduced. Both tools together form an ideal tool chain for product line development in constrained environments as one reduces the configuration complexity on the source code level, while the other helps to manage the variability on the abstract feature level and provides a mapping of features to aspects, classes, or other modularization units.

2. PURE::CONSUL

The Pure::Consul tool chain has been designed for development and deployment of software program families. The core of Pure::Consul are several models which are used to represent the problem domain of the family, the solution domain(s) and finally to specify the requirements for a specific representative (member) of the family. The tool allows for integration of many different variability realization techniques through its customizable transformation backend. Thus, it is able to incorporate frame processors, code generators, or arbitrary other tools.

The central role is played by *feature models* which are used to represent the problem domain in terms of commonalities and variabilities. Pure::Consul uses an enhanced version of feature models compared to the original feature models as proposed in the FODA method [2].

The solution domain(s) (i.e. the implementations) are described using the *Pure::Consul Component Family Model* (CCFM). It allows to describe the mapping of user requirements onto variable component implementations, i.e. the customization of a set of components for a particular context. As the name suggests, this model has been newly developed for Pure::Consul.

The *feature sets* are used at deployment time and describe a particular context in terms of features and associated feature values.

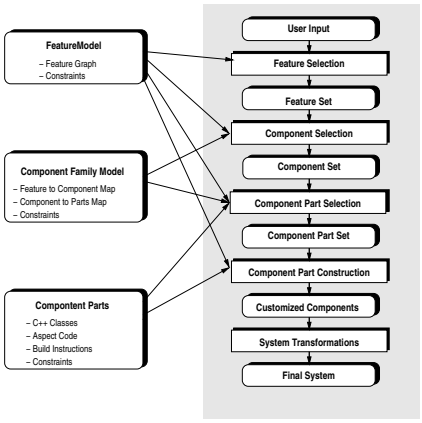

Figure 1: Overview of Pure::Consul process

Figure 1 illustrates the basic process of customization with Pure::Consul. Most steps can be performed automatically once the various models have been created. The developers of variable components have to provide the feature models, the component family models, and the implementations itself. A user[1] provides the required features, the tools analyze the various models and generate the customized component(s).

The key difference between Pure::Consul and other similar approaches is, that Pure::Consul models only describe *what* has to be done, not *how* it should be done. Pure::Consul provides only basic mechanisms for manipulation and generation, which can be extend according to the needs of the Pure::Consul user. This flexibility is achieved by combining two powerful languages inside Pure::Consul and allowing the user to extend this system.

The first language is Prolog, a widely known language for logic

[1]Here a user can be either human or also a tool which is able to derive the set of required features automatically from some input

To appear in the First Workshop on Reuse in Constrained Environments (RICE'03) at OOPSLA 2003

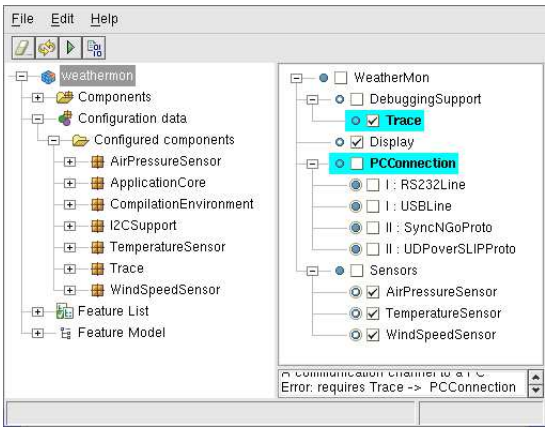

Figure 2: Pure::Consul graphical user interface

programming. Prolog is used for constraint checking, i.e. for expressing relations between different features. The same logic engine is used for component selection and customization control.

The second language is a XML-based language called XML-Trans which allows to describe the way customization (transformation) actions are to be executed. Due to its modular structure, it can be extended with user supplied transformation modules. This can be used to provide seamless access to special generators or other tools from within the tool chain.

The tool chain consists of several different tools for different purposes. A command line client allows integration into automated processes, while the graphiclal frontend (figure 2) can be used for both, modelling purposes and also deployment.

3. ASPECTC++

On the implementation level statically configurable software is typically difficult to maintain and reuse. Often a lot of configurable features affect the same module or a single feature affects large fractions of the system. That makes the modules hard to understand and the implementation of a feature can be widely scattered across many modules.

To overcome this modularization problem and to achieve clean separation of concerns aspect-oriented programming (AOP) can be applied. The aim of AOP is to provide means to modularize the implementation of crosscutting concerns. AspectC++ is a language extension for C++ that supports AOP with C++ component code. As there is no expensive runtime system needed, our language and compiler are well suited for deployment in ressource restriced embedded software projects.

To understand the advantage of using AspectC++ for the implementation of configurable features consider the example illustrated in figure 3. It shows three versions of bus transactions that might be executed by the CPU to communicate with peripheral ICs. Each transaction is a sequence of three single byte transfers. During a byte transfer an exact timing is required and, thus, hardware interrupts must be disabled. Depending on the implemented synchronizsation granularity the effects are different total execution times and delays during interrupt processing. There is no best granularity. The best choice depends on the requirements in a concrete application scenario. Thus, if the bus driver code should be reusable, the granularity must become a configurable feature.

With tradidtional implementation techniques configuration code has to be inserted at every point in the bus driver module, which

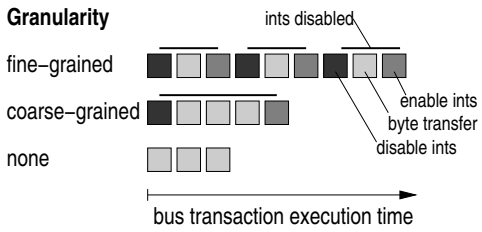

Figure 3: example: a configurable interrupt synchronization granularity

might be relevant for synchronization in *any* of the scenarios. By using AspectC++ the synchronization code can be completely separated and there can be one aspect for each granularity. For example, the following code fragment shows the fine-grained synchronization:

```
aspect FineGrainedIntSync {
  advice execution("void Bus::send(char)") ||
         execution("char Bus::receive()") :
           around () {
             disable_int ();
             tjp->proceed ();
             enable_int ();
           }
};
```

The code wraps calls to `disable_int()` and `enable_int()` around each execution of the `send()` and `receive()` functions of the bus driver from outside.

By separating this code from the bus driver itself there is now a direct mapping of the configurable feature in the Pure::Consul feature model to an implementation module. This simplifies the whole development process and a high degree of reusability is achieved.

4. CONCLUSIONS

Crucial factors in the devolopment of product lines in constrained environments are adequate models for the domain analysis and design, programming languages that support the modular implementation even of crosscutting concerns, and an integration of all this in a user-friendly development environment, which supports the whole process. The Pure::Consul tool chain together with AspectC++ addresses these factors successfully.

5. REFERENCES

[1] D. Beuche, H. Papajewski, and W. Schröder-Preikschat. Variability Management with Feature Models. In *Proceedings of the Software Variability Management Workshop*, pages 72–83, University of Groningen, The Netherlands, Feb. 2003. Technical Report IWI 2003-7-01, Research Institute of Mathematics and Computing Science.

[2] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. Feature Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, USA, Nov. 1990.

[3] O. Spinczyk, A. Gal, and W. Schröder-Preikschat. AspectC++: An aspect-oriented extension to C++. In *Proceedings of the 40th International Conference on Technology of Object-Oriented Languages and Systems (TOOLS Pacific 2002)*, pages 53–60, Sydney, Australia, Feb. 2002.