

OOPSLA 2000 Workshop: Methods and Tools for OO-Framework Development and Specialization

Submission by Markus Voelter, MATHEMA Software GmbH, Germany
markus.voelter@mathema.de

ABSTRACT

This paper is a submission to the OOPSLA 2000 workshop on "Methods and Tools for OO-Framework Development and Specialization". It presents an approach to framework configuration using the "explicit variability" metaphor. The approach presented in this paper will be verified in a research project within MATHEMA software during the next months.

The current situation on frameworks

A framework as it is today consists of fixed and variable aspects:

- The fixed aspects capture those aspects of a framework of which the domain analysis has determined that they are common to all possible framework instances.
- The variable aspects capture everything that is subject to change (or configuration) in different framework instances.

A programmer who intends to use a framework has to ensure that the framework suits his needs, i.e. that the future application can be implemented as an instance of the framework.

First of all, it is important to make sure that all of the fixed aspects of the framework can be used as they are provided, because they cannot be changed. Usually, this is fairly easy, because typical framework documentation provides a quite good definition of the target domain and its invariants.

Then comes the harder part. The programmer has to find out which aspects of the framework can be adapted, and how. The documentation is usually much worse in this respect because there are many more possibilities to change something than to leave something as it is. The literature proposes to use a formalized form of documentation, e.g. a definition for each hook, why, how and when it can be configured. However, this is still an imprecise way of specifying a framework's variability.

The problem with framework variability is that it is usually defined implicitly in the framework structure. This paper describes the first ideas for an approach how the variability of frameworks can be made more explicit. That means, that the configuration options are specified using program structures. The paper forms the basis for a project which I am about to start in my company that aims to build a prototype for this kind of framework design.

Explicit Variability in Frameworks

Preliminary Definition: *Explicit variability means, that the variable aspects of a framework are explicitly represented in the framework and externally visible. Therefore, the framework's variable aspects can be browsed or configured with the help of a tool, and after configuration, a framework's configuration can be validated.*

Goals

- Using a generic tool, a framework's variable aspects should be made visible.
- Each variable aspect (i.e. hot spot, hook) should contain a) its documentation b) the type of variability (i.e. switch, subclassing, etc) and c) the valid configuration options.
- The generic tool should provide a way to configure all variable aspects and should provide help in doing so, e.g. by letting the user select one of the options, or by generating code based on user input. However, using the tool must not be mandatory.
- The framework instance should then validate its own configuration. This includes e.g. checking pre- or postconditions on user-supplied classes or running test cases.

Ways to get there

FRAMEWORKS FEATURES AS PROGRAM STRUCTURES

The single most important aspect is an idea borrowed from generative programming and feature oriented domain analysis (FODA). There, the variability in a family of applications is modeled using features. A feature of an application can be mandatory or optional, a feature can be implemented by one of multiple alternatives, etc. There can also be interdependencies among features, or, more general, valid or invalid configurations. A feature diagram is a prominent way to visualize features and their dependencies.

A framework also has features that can be modeled using these techniques. Explicit framework design includes a runtime representation of a feature model. The model contains constraints which define a valid framework configuration and before a framework instance runs, these constraints are validated, ensuring that a framework runs only in a valid configuration; this prevents many kinds of runtime errors.

The model can be configured with the help of a tool. Each feature contains information about how it can be configured and, depending on the "how", helps the programmer in doing so, either by documentation or by providing alternatives which the user can select. Code generation will also be an alternative. The tool can request some parameters from the user and, based on this input, generate code that will be used as part of the framework.

By defining defaults for specific features or feature combinations, a framework could be configured on different "levels of detail".

Framework features could be of user-defined types; for example, this will allow explicit representation of patterns in framework design: A user registers a certain

class (with a defined interface) to play the role of an observer, while another class is registered to play the role of a subject. They are both attached to the same feature definition, which is of type *ObserverPattern*.

CONFIGURATION RUN

After the framework's configuration options have been selected by the user, either using the tool or by manual coding, the framework executes an explicit configuration run, which includes the following activities:

- **Code Generation:** The respective features generate the required code
- **Testing:** Code which was supplied by the user (in the form of classes) is tested, to make sure the necessary pre- and postconditions and invariants hold.
- **Validation:** The framework checks, whether the specified configuration is valid by checking the constraints on the feature model.

Then, if the configuration is valid, it is made persistent. Running the application first configures the framework using a valid configuration.

Discussion and Next Steps

The presented approach borrows concepts from framework development and combines them with techniques found in generative programming. The implementation and the possible types of features (i.e. they way how they can be configured) depend on the programming language in use.

The research project which will be started at MATHEMA aims to build a prototype implementation using the Java programming language. I hope that the workshop and the project can benefit from each other, by discussing and challenging the approach in the workshop and verifying the (perhaps modified) approach during the project.