DELTA
software technology

HyperSenses®

DSLs and Software Generators

"I would rather write programs to help me write programs than write programs

Dick Sites

The Challenge

Copy-Paste-Adapt seems to be the most commonly used method to implement similar solutions or variants of a system (to simplify matters, the term 'software family' is used when talking about similar solutions or a set of variants, a single member of a software family is referred to as variant or instance). But the disadvantage of this method is that every task, e.g. correction, change, enhancement, has to be repeated for every variant which is error-prone and time consuming as well as expensive.

Using generators and domain specific languages (DSLs) avoids the above mentioned problems.

However, most generator technologies require quite experienced developers. Often developers do not accept generators because the generated code is hard to read and maintain. Business-specific programming guidelines are rarely fulfilled by generators. Developing a DSL is often regarded as a simple finger exercise. If the DSL has to be extendable, the task gets a bit more tricky. But as soon as a DSL and a generator have to fit to each other, their development transmutes from a simple finger exercise to a time consuming challenge for an experienced developer. Everyone who is going to use the new language first has to spend some time to learn it. Depending on the language's complexity this familiarisation process can require quite a high effort.

Not until generator and DSL are completely developed, a testable system is available. This means (conceptual) errors will be found at a very late point of the developing process which in the worst case results in an expensive correction. In addition, the long development phase and the late availability of a testable system impede the usage of generators and DSLs in combination with agile methods.

Due to this reasons many companies avoid the usage of generators and DSLs and use Copy-Paste-Adapt instead.

The Requirements

When developing HyperSenses, we aimed to overcome the above mentioned weaknesses of the development of generators and DSLs. In concrete, the target was to develop a generator technology

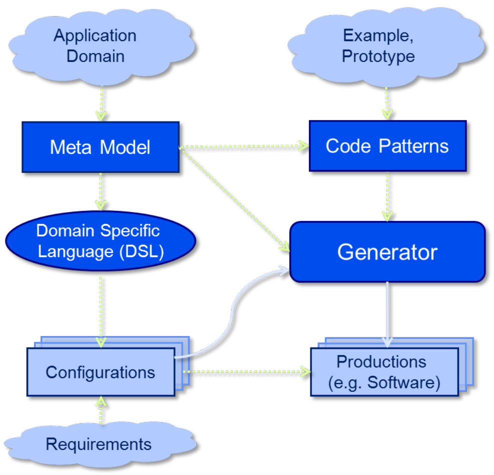
- Which allows the generator and DSL to be developed stepwise as well as deployed and to be used in combination with agile methods
- Which simplifies and mostly automates the tedious programming of a generator and a fitting DSL and which requires only very little knowledge on generator technologies and DSLs
- With an easy to read and maintain specification
- Which supports a DSL simple to use so that nearly no training is required
- Which encourages to reuse expert knowledge and share it with as many developers as possible
- Which generates code easy to read for humans, accepted by software developers and fulfils the companies programming guidelines if available.
- Which scales for all sizes of projects

To achieve these targets, HyperSenses introduces the concept of model-driven meta-development, (MD)² in short. While other approaches use generators and DSLs for model driven development and generative programming of applications we consistently apply model driven development and generative programming to develop generators and DSLs.

The Solution – (MD)²

Less is more

The concept's central element is a meta model. This model describes the variable parts of the application domain of the system family. It contains solely the variable parts, which differentiate the individual members of the family. In contrast to other approaches, invariant parts are not modelled in the HyperSenses meta model, the modelling effort is therefore reduced significantly. A meta model can be created in HyperSenses itself or created within an UML-case tool, imported via XMI interface and edited in HyperSenses if necessary.



"I'd never program it that way..."

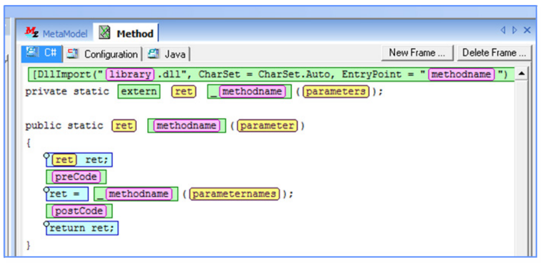
... might be the most commonly used argument from software programmers against the usage of a generator. As the (generator) developer himself defines the target language and code as well as its formatting, this argument doesn't hold in HyperSenses. The developer's definition ensures the readability and maintainability of the generated code and the fulfilment of company-specific programming guidelines. Also important, the look and readability of the code increases the acceptance of the generator by programmers.

Expert knowledge for everybody

This is achieved by using code patterns which specify the language dependant parts of the generator. Such a pattern is derived from existing code (or code fragment), the underlying concept is called 'pattern by example'. Existing implementations or prototypes, developed and optimised by an expert and possibly already usage-proven, are the basis for code patterns.

To create a code pattern, the code is searched for variable parts which differentiate single members of the system family. Those variable parts are exchanged by slots or optional code blocks which are linked to the meta model. Due to the linkage between meta model and slots respectively optional blocks it is only possible to generate code which is valid within the according application domain.

In this way, the expert knowledge is shared between all developers of the team, especially those with less experience, and can be easily reused and extended.



```
[DllImport("library.dll", CharSet = CharSet.Auto, EntryPoint = "methodname")
private static extern ret _methodname(parameters);

public static ret methodname(parameter)
{
    ret ret;
    preCode
    ret = _methodname(parameternames);
    postCode
    return ret;
}
```

Derived code pattern, frame for C#

Specifying generators instead of programming them

The meta model describes the variable parts of the application domain of the system family including their relationships. The code patterns describe the code to be generated and its formatting. Due to the linkage of the patterns to the meta model the necessary navigation is implicitly given.

Thus, the meta model and code patterns contain sufficient information to enable HyperSenses to automatically create the generator, i.e. instead of tediously programming the generator it is simply specified.

You're an expert in building generators and expected a long description about the programming of the generator as usually when a new technology is introduced? Sorry, in HyperSenses you just have to press the correct button and there is therefore no need for a long explanation.

DSL development – no secret science

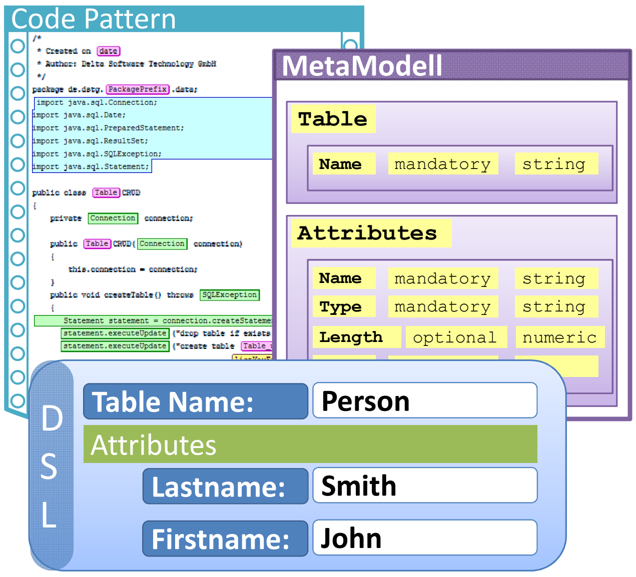
To create a certain instance of the system family, the generator needs to know how to calibrate the variable points. This calibration is task of a configuration which is described by a DSL. Generally, developing an extendable DSL and generator, where DSL and generator fit to each other, is nearly a secrete science. For this task, a developer needs significant knowledge about the creation of consistent languages and plenty of time. This is not required in HyperSenses. What can or has to be configured and how is specified by configuration patterns, i.e. they specify the DSL. Like code patterns they contain slots and block which are linked to the meta model. In addition to support the end user of the DSL it is possible to add explanatory text within the configuration patterns.

Similar to the generator which is specified by the meta model and code patterns instead of being programmed, the DSL is created automatically from the meta model and the configuration patterns whereby grammar and syntax are implied by the meta model.

Using a DSL as simple as ordering something on the internet

Depending on the DSL's complexity the familiarisation effort for the language can be quite high until a first configuration can be written.

HyperSenses bypasses this obstacle by introducing form-DSLs which are similar to electronic forms known from the internet for example. Such a form-DSL is easy to read and fill-in for the user and unambiguously evaluable by the generator, particularly only syntactically and grammatically correct input is possible. To further improve the readability of a form-DSL, its optic can easily be manipulated.



```
/*
 * Created on date
 * Author: Delta Software Technology GmbH
 */
package de.datg.PackagePrefix.data;
 import java.sql.Connection;
import java.sql.Date;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class TableCRUD
{
    private Connection connection;

    public TableCRUD(Connection connection)
    {
        this.connection = connection;
    }
    public void createTable() throws SQLException
    {
        Statement statement = connection.createStateme
        statement.executeUpdate("drop table if exists
        statement.executeUpdate("create table Table_
```

Schematic figure of the HyperSenses components

Stepwise and distributed development

As soon as a part of the meta model exists, the patterns (code as well as configuration) can be developed. This development can be done distributed. It is possible to change, extend or enhance the meta model and all patterns at any point in the development process with only little effort. Likewise, it is possible to create an executable generator and/or DSL which can then be tested at any time. Thus, HyperSenses supports a stepwise development as e.g. required when agile methods come into play.

From stand-alone to integrated

HyperSenses can be run as stand-alone tool as well as integrated into another development site like Eclipse or VisualStudio. It is also possible to exchange the meta model with other case tools via HyperSenses' XMI interface.

Highlights

Really agile

HyperSenses simplifies the development of generators and DSLs and supports their stepwise and distributed development. Testable and reproducible results are obtained within hours or a few days instead of weeks as usual. Therefore HyperSenses enables the development of generators and DSLs for agile methods.

Development with reuse for reuse

Deriving code patterns from existing code (fragments) has at least two advantages. At first, established and optimised solutions which were developed by experts can be reused and extended. At second, the expert knowledge is shared with anyone of the development team especially with less experienced developers.

Efficient already for small applications

HyperSenses scales for small (sub)tasks as well as for huge overall systems. Even for small sized or specialised tasks or projects, development and usage of generators and DSLs become profitable.

Acceptance

The usage of code patterns leads to the fact that the developer defines how the code has to be generated for a certain task, thereby also the readability of the generated code is determined. In this way, the acceptance of the generator is increased significantly. In addition, this method ensures that company-specific programming guidelines are met by the generated code.

Maintainability

In contrast to other generator technologies, HyperSenses strictly distinguishes between production instructions and target code which results in a higher readability and maintainability of the generator's specification.

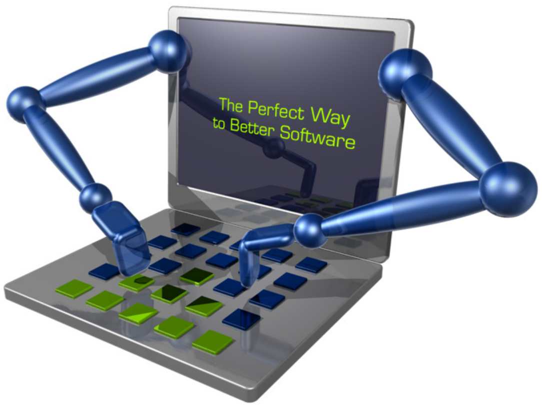
Specifying instead of programming

The simple derivation of patterns (code as well as configuration) allows the development of generators and fitting DSLs even without expert knowledge.

DSLs and generators do not have to be tediously programmed instead they are specified. Using this specification, DSL and generator are automatically created.

Simply configured

To create one's first configuration, no long lasting familiarization with the DSL is necessary, because HyperSenses offers the notion of an easy to use form-DSL which ensures syntactically and grammatically correct configurations.



Further information and a free Community Edition can be found under:
www.hypersenses.com

Delta Software Technology

Delta Software Technology is a specialist for generative development tools that automate the modernisation, integration, development and maintenance of individual IT applications. We understand the enterprise IT as a living organism that is continuously changing. Our automated solutions help you to quickly and safely adapt your applications to new business requirements, architectures, technologies and technical infrastructures.

AMELIO® Modernization Platform™
The tailor-made factory for the modernisation of large IT applications:
100% automatically and that's why it is safe, reliable and error-free.

HyperSenses®
Integrated system for model driven development of DSLs and software generators.

SCORE® Adaptive Bridges™
Intelligent service enablement for the reuse of proven
applications with modern technologies: flexible, profitable and non-invasive.

SCORE® Data Architecture Integration™
Data as real business services: fast, easy and independent of
data architectures and management systems.

SCOUT²™ Development Platform
Optimized and integrated development processes across all software components, tools and platforms: Stop the "fight against the infrastructure".

ADSplus™ Application Development
Platform-independent development for future-proof back-end applications.

Delta has a more than 35-year track record of successfully delivering advanced software technology to Europe's leading organisations, including AMB Generali, ArcelorMittal, Deutsche Telekom, Hüttenwerke Krupp Mannesmann, Gothaer Versicherungen, La Poste, RDW, Suva and UBS.

DELTA
software
technology

Delta Software Technology GmbH
Eichenweg 16
57392 Schmallenberg, Germany

phone +49 2972 9719-0
fax +49 2972 9719-60
e-mail info@d-s-t-g.com

www.d-s-t-g.com

Order number: MA 24015.02 – January 2012