An Agile and Extensible Code Generation Framework

Dimitrios S. Kolovos, Richard F. Paige, and Fiona A.C. Polack

Department of Computer Science, University of York, UK.
{dkolovos, paige, fiona}@cs.york.ac.uk

Abstract. Code generation automatically produces executable code by software. Model-driven code generation is currently the most flexible and scalable generative technique, but there are many complaints about the complexity it introduces into the development process, and the design decisions imposed on the code. Here, an agile code-oriented model-driven generative methodology is outlined that reduces complexity and allows the engineer to define the exact form of the produced code and embrace change in the requirements in an automated manner. A flexible tool, ECGF, supports this methodology, and a case study in rapid generation of large-scale HTML documents is outlined.

1. Introduction

Code generation is the automated production of code by software. In the context of this paper, a special type of code generation called *model-driven code generation* is examined. This concept is to design systems using abstract representations (models) and then, based on the models, to automatically derive working code.

There are two approaches to model-driven code generation. In the first model-driven approach, the engineer designs the system in terms of abstract models (e.g., in UML [10]) and then generates code based on them. In the *code-oriented* approach (COMMD), the engineer designs part of the code of the system and then creates models that can be used to generate the rest of the code needed.

Code generation tools available today, such as Rational Rose, support generation from abstract models, e.g., in UML, and for particular popular application domains, such as enterprise data-driven systems. Limited tool support exists for the code-oriented model-driven approach. A generic, unified tool support structure would be beneficial for this approach, and to support agile development techniques such as Extreme Programming [8]. COMMD is compatible with the tenets of agile development, as it is code-based and accommodates changing requirements as well as the need for flexibility in software.

We describe a COMDD methodology and a supporting tool. The tool, called the Extensible Code-Generation Framework (ECGF), is tailored for use in agile development approaches. We demonstrate the flexibility and power of ECGF via a case study in the agile development of large-scale systems.

2. Code generation and agility

There are many reasons why the structure or the functionality of code should be modified (even radically) during the development process. In these situations it is important to have an agile process that embraces change. The advantage of template based code generators is that the information on the functionality and structure of the code exists in the models and the templates respectively. To embrace a change the engineer has to modify the templates and/or the models and regenerate the code to fit the new requirements.

3. A Code-Oriented Model-Driven Methodology

Agile development shifts attention from models (e.g., in UML) to code. The rationale of this approach is that the quality of the code, and not that of the models, finally determines the success of the development process. With this principle in mind a COMDD methodology is outlined.

1. **Write a prototype by hand**: this is radically different to most model-driven development techniques, but similar to agile methods, since development starts with coding. The aim with this approach is to allow the generation of hand-written quality source code rather than non-human-modifiable and tool-specific code.
2. **Decide on applicability:** In order for a system to be suitable for code generation it should contain large amounts of repetitive code, which cannot or is not preferable to be eliminated by changes in the design.
3. **Identify common and variable parts of the code:** Models will represent the variable code parts, while templates will represent the common parts.
4. **Identify scope:** Thus, in this step the system is analyzed to determine which parts to assign to the code generator, keeping in mind that the more tasks that a code generator must do, the more complex it – and the resulting code – will become.
5. **Create models and templates:** Our methodology implies that the engineers compose the templates and the models themselves. Composing the templates and models manually ensures that they fit the desired structure of the system and that produced code follows the coding and quality standards that the engineer has set.
6. **Reproduce prototype using the code generator:** the next step is to reproduce the prototype system using the models and the templates. This reveals aspects that might have been mistreated, and builds confidence that the code generator can actually generate the desired code.
7. **Generate the final system:** The final step is to build the models and generate the rest of the modules of the system.

As the process is agile, it is important that the generated code is compiled and tested after each generation, to ensure that it does not contain syntactical and logical errors. Test skeletons, but not implementations, could be derived from the models in order to avoid error propagation from models to tests.

There are two additional points to note about the methodology. First, the development process needs careful management and coaching in terms of the use of templates and models using a configuration management system so that all the developers work with the same versions of the templates. The second point is that models and templates from a successful development process can be reused to generate similar software in the future. Of course, these artifacts need to be viewed critically and may need refactoring for different projects.

4. ECGF: The Extensible Code Generation Framework

ECGF is a tool to support the agile COMMD methodology outlined in Section 3. The tool has been designed to be usable by technical developers; this is an important quality attribute, since many developers are anecdotally averse to existing code generators, considering them inflexible to use. Usability is achieved by providing a simple integrated development environment (IDE). Since template and model construction, as well as code generation, are error-prone tasks, ECGF also provides detailed feedback about errors detected.

4.1 ECGF Framework

ECGF builds on an open-source template engine, the part of the system that interprets and executes the templates in order to generate the code. A simple scripting language called Velocity Template Language is used for writing templates. Scripting languages are most appropriate for rapid template development because they provide a brief and powerful syntax without the type-casting overhead of strongly typed languages.

General requirements for a suitable language for agile modeling include: extensibility, mechanisms for model validation, and a substantial existing user basis. The two alternatives that meet the requirements to some extents are XML and UML. UML is undesirable because of its semantic fragmentation, and the need to use heavyweight UML metamodel extension. XML has a simple syntax and provides mechanisms for validation, queries and transformations. Moreover, there are numerous high-quality open-source XML parsers that implement these features. Thus, ECGF supports XML as a modelling notation.

In order to integrate XML models with Velocity Templates we use a mechanism that combines the XML Document Object Model (DOM) [7] with pluggable Velocity introspection [5]. Pluggable introspection allows overriding the default method invocation behavior of the template engine. In this way we achieve an elegant API for iterating and managing the DOM that makes it look and feel like a custom object model. We briefly explain this mechanism below.

Consider a simple Velocity expression: **`$a.b`**. This expression is interpreted as following *"if `a` is an XML element, first see if it has an attribute named `b` and if so return the value of `b`"*. By this mechanism, we shield the user from metamodelling but still obtain the benefits from them, via introspection.

5. Generating Browsable Documentation for Modelware

To illustrate the usability of the methodology, we briefly outline a recent application of ECGF in an ongoing EU project, "Modelware" [11]. As part of Modelware, we need to spend many hours traversing the UML metamodel [9] as represented in XMI. Given the complexity of the physical structure of XMI, this is time-consuming and tedious, and it was decided to generate a browsable HTML-based navigation utility.

The first step was to manually create a sample class description page and a sample package description page as well as the frame page that would contain class and package pages (prototyping). When we stabilized the desired content and format of each page type, we composed the respective templates and produced the rest of the navigation utility.

The product consists of 147 web-pages which would have possibly taken weeks to compose by hand while the design, implementation and testing of the demonstrated solution required less than a day. We then used the same templates to generate identical mechanisms for the navigation of Common Warehouse Metamodel (CWM) and Meta Object Facility (MOF 1.4).

6. Conclusions

We have outlined an agile code generation technique, along with a supporting tool, that enables rapid development in an agile way. We have applied the approach in a number of practical case studies. The tool, ECGF, has proven to be usable, efficient, and practical in the large. We are currently using ECGF in the Modelware project and elsewhere for a variety of code generation tasks. We are also considering adding further features to ECGF, particularly pattern recognition for lightweight assistance in detecting common code, and also better visual support for model editing.

References

1. Sal Mangano. XSLT Cook Book. O' Reilly, 2003.
2. Jack Herrington. *Code generation in Action*. Manning, 2004.
3. OMG. Model driven architecture official web-site. http://www.omg.org/mda/.
4. XDoclet development team. http://xdoclet.sourceforge.net.
5. Velocity development team. http://jakarta.apache.org/velocity.
6. Code generation is a design smell. http://c2.com/cgi/wiki?CodeGenerationIsaDesignSmell.
7. Eric van der Vlist. XML Schema. O' Reilly, 2002.
8. Kent Beck. *Extreme Programming Explained*. AWL, 1999.
9. OMG. UML 1.4 Metamodel. http://www.omg.org/uml.
10. OMG. CWM Metamodel. http://www.omg.org/cwm.
11. Modelware Integrated Project. http://www.modelware-ist.org