Towards a Synthetic Approach to the Compilation of UML Models (Position Paper)

A Practitioner Viewpoint

Georges Reich
Reich Technologies
1 rue Amiral Nielly
FF-29200 BREST, FRANCE

georges.reich@acm.org

ABSTRACT

This paper summarizes 12 years of experience and recent research in meta-model driven code generation and large model compilation. From this extensive experience, lessons are drawn which result in a position globally in favor of a model-driven architecture. Reserves and pitfalls are expressed.

General Terms

Design, Experimentation, Human Factors, Standardization, Languages.

Keywords

UML, MDA, code generation, model compilation, case studies.

1. INTRODUCTION

In essence, what the Model-Driven Architecture (MDA) initiative of the OMG proposes is to build a generic domain model with UML; to explicitly represent the semantics of the target platform with UML semantics; to automate the generation of code through a mapping between the two; and then, to design applications with UML upon the generic model.

In this position paper, we mainly present and comment on design constraints associated with several industrial case studies. The projects were carried out by means of CASE-Tools, with automatic code generation, in domains as varied as telecom, banking or insurance, over a span of 12 years. OMT, UML and pre-versions of UML 2 were used to model applications, generic domain models and platform-specific targets.

Most of the case studies follow in intent or extent the scheme presented above. They will be reviewed in approximate historical order, because they are the result of evolution, and practical leverage and pitfalls of an MDA-like approach will be shown.

Conference '00, Month 1-2, 2000, City, State.

2. THE CONTEXT OF EXPERIENCE

Since 1991, we have built several CASE-Tools, from the same stock, providing facilities to model and generate applications. This object repository-based tool was named Intelligent Software Factory (ISF). From 1991 to 1995, the modeling tool supported/embedded a rigorous meta-model loosely based on OMT [13, 14, 15] (see below "Early experience with OMT" and "Financial Services"). Through a collaboration with IBM (see below "Insurance"), an alternate meta-model was developed.

Since our participation in the creation of the UML standard, the tool is based on the UML meta-model and meta-meta-model (see sections 5 to 9). Around the same times (1998), we stopped selling ISF as a product, in favor of building customer-specific co-generation environments based on the ISF frameworks and consulting on the target architectures.

Since its inception, ISF was designed not only to model, but to build applications. The target for code generation is a platform not only a language. The purpose is to generate whole applications, possibly incomplete, including behavior, not code skeletons. From 1994 onwards, the tools were mostly[1] generated with the previous versions.

Of importance is the fact that, as generated source code is hardly a final product, the generators must produce highly readable sources, where the intention of the designer appears clearly. Drastic optimizations and transforms are thus excluded when the target is not bytecode or machine-code. Users' conventions, style guides and programming rules may also bear heavily on generation stages.

The generators described in this paper offer support for oblique transformations [11], mostly forward refinements, but also some important horizontal transforms for aspects such as persistence, transactions and distribution, as well as some backward transforms.

Very small teams realized the various projects presented below, in practice a handful of people. Most represent several man*years of effort, spanning the design and implementation of modeling tools, generators, target frameworks and processes. Not all projects have been detailed, only key experiences that triggered major evolution of the generation technology, rather than simple refinement of the approach.

3. EARLY EXPERIENCE WIHT OMT

ISF started with hard-coded generators in Smalltalk, developed from scratch. Those procedural generators produced Smalltalk and C++.

In this version of ISF, model annotations, to express navigation and constraints, e.g. FSM transition guards, were added directly in the target language (Smalltalk, C++). The tool was meta-model-based, which was unusual at the time and thus inputs were restricted, e.g. if an operation is triggered by an event on a sequence diagram, the operation must be defined on the class of the target instance.

Huge monolithic generators then processed the semantic information and model graph annotation.

It is our experience that the presence or absence of an object repository, as important as it may be to the overall process, is mostly irrelevant to the code generation. In opposition, the forward only vs. round-trip nature of the process has a deep impact on the structure of the generators. We have never seen either extremes work in practice. The current best fit for IS seems to lay on a mostly-forward generator with some reverse engineering of the source code, and model graph annotations, along with a general "Model-Driven Attitude". The extraction after the fact of information from the application once debugged and linked to e.g. middleware libraries, and the "enrichment" of the model must be taken into account in a general generation strategy.

An important part of the work was de facto aimed at raising the target language to some concepts of the model language through extensions, libraries, and generative patterns to reduce the semantic gap, e.g. implement the semantics of associations in Smalltalk, instead of converting associations into mutual object reference. Metaclasses for Smalltalk and templates for C++ were allowing some form of simple language extensions, which enabled to reduce the semantic gap. "Languages that incorporate metaobject protocols blur the distinction between languages designer and language user". [16]. The Java-based targets suffered from the lack of feature. On the contrary, the python-based target was drawn further along the lines of extension (detailed below in "Mobile Services").

Among others, the US Department of Agriculture used this generator to generate a large farmer application.

4. FINANCIAL SERVICES AND AN EXTENDED OMT

X Financial Services, a leading leasing services company, projected a complete rewrite of their core business information system. The customer wanted a CASE-tool to handle requirements, modeling and Smalltalk generation for a client/server architecture. The initial generators were to be outsourced and later transferred under customer evolution. We were still believing in a closed components world and did not intent to provide what was the core know-how of our company. We have since been converted to open source and a service business model. The customization and evolution of the code generators was made available through a specific script language.

This version of the tool introduced scripted *Walkers* and *Producers*, while retaining the previous architecture for intermediary mechanisms (hard-coded in Smalltalk). A simple custom script language, ScriptTalk, was designed and implemented in order for customers to influence the code generation.

We call "*walkers*" a set of mechanisms (object schemes) which include model graph traversal strategies, local iterations and relational operators. *Walkers* are used during semantic analysis, both in a semantic analysis global phase and during local semantic analysis steps triggered by other generator elements, to enrich the model via optimization and inference, through the attachment of model graph nodes and annotations, and to delegate work to the *Producers*.

[1] For most organizations, we reduced the time-to-market by at least an order of magnitude, helping to save thousands of man*months in the process. In no case did we achieve 100 % code generation (as measured against the total amount of final code). Not even for our own purpose. Best cases measured range from 63 to 87 %.

We call *Producers*, small to medium size object schemes computing or outputting code, which can be part of the Smalltalk procedural subroutines. *Producers* can be scripted or can call upon a template-directed engine. The latter is a recent introduction, see below "Mobile Services" and "CTI".

Use case extensions contributed by Alistair Cockburn [7] and Ralph Hodgson led us to discover the value of collecting non-functional requirements (NFR) to generate more efficient code. Attaching e.g. expected volume and frequency of use to use cases allowed several tunable enhancements to be added to the generators. This configuration information could also be used to re-generate the application once the actual figures were refined. Those are enhancements of the model graph, or enrichment, rather than optimization in the sense that they allow a range of inferred deduction to be made during all phases of the generation. They may also be used for optimization.

During the Insurance experience (see next), the collection of NFR was systematized, and integrated into the development process.

5. INSURANCE BUSINESS ENGINEERING LANGUAGE

In 1996, the IBM Insurance Solutions Development Center in La Hulpe, well known for its generic insurance model, Insurance Application Architecture (IAA), was in the process of redesigning an object-oriented generic model of the insurance business. IBM wanted tool support for the meta-modeling, the modeling by insurance companies, the simulation and the projection to general platforms. The decision was to outsource the tool-suite.

The tools were built and applied successfully to a number of pilot projects. To the best of my knowledge, nothing of this experience has ever been released by IBM and only the most general aspects can be presented here to avoid infringement on intellectual property. Some of my personal views on the subject have also been published in [20].

For intellectual property reasons, a complete tool-suite was rebuilt (i.e. different from our baseline tool), using only the most generic components and frameworks available from ISF. A specific meta-model and its recursive meta-meta-model were designed with IBM with contributions from Steve Cook [9] and Alistair Cockburn [8] A business expression language was added to manipulate instances, models and *meta*model*. Navigation, invariants, pre and post-conditions could be expressed in IBEL, Insurance Business Engineering Language. This language was introduced by IBM into the UML and became OCL. For those who like small history, this is why the type system of OCL did not match the type system of MOF until the rewrite of UML 2.0, it was simply designed for a completely different, if congruent in intention, *meta*model*.

The expression "*meta*model compilation*" was coined, where * stands for its frequent meaning of [0:n] of the previous element. I now recognize that the fields of code compilation and model generation are essentially different, which doesn't preclude from mutual pollination.

A first attempt was made at Meta-driven *walkers*, which was abandoned because it introduced complexity and did not allow for the complex kind of procedural code required for the generators. No recent attempt at an XML+XSLT has been made because of the previous failure of this approach.

Target frameworks were not modeled, but represented as optimized *producers*, closely coupled to target architecture. No producer was yet template-based.

Strategies based on system-level analysis were added, building upon the work of Reich and Reenskaug for the UML [19], evolved from [18, 1]. Those proved exceptionally helpful from a generation standpoint. The system-centric approach has since then been advocated successfully by Trygve Reenskaug and was eventually incorporated into UML, though not in the early versions.

6. ONLINE SERVICES DOMAIN-SPECIFIC MODEL

In 1992, France Telecom, rich of online services on the Minitel platform and other proprietary networks (Image Bank, etc.), wanted to avoid the systematic rewrite occurring with each language/technology change. Initial experiments with Smalltalk convinced them that models could be designed independently of the target technology. A domain model was investigated and soon covered the web and interactive TV as well as the traditional services mentioned above. This domain model was first stable in late 1993, and has been maintained up to this day, with additions and modifications, but no major rewrite.

The CASE-Tool and generators were derived from the ISF Branch (no meta-driven *walkers*).

The target platforms for online services included server farms, transactional middleware, relational and non relational databases and an array of languages. As those requirements exceeded the capabilities of the script language, we were back to entirely hard-coded generators. In addition, the targets have also been meta-modeled and they are accessible through modeling tools.

OPeNS is a unique experience in that the ongoing Domain Analysis on problem domain and target platforms, the generic model for online services, and the co-design of the tools and generators started in 1993, and are still on the workbench. That gives us a long perspective on the evolution of target platforms [6, 21, and 22].

OPeNS also recognized early the need to explicitly build upon a different kind of software development organization, and to adapt the tools to the roles of the people in that organization. The need to "debug" the model before it was deployed onto a target arose, and OPeNS was distinguishing between Platform-Independent Models and Platform-Dependent Models before MDA was invented.

This experience demonstrated that, in several cases, using complex target components and frameworks could be counter-productive. As an example, gathering NFR and inferring transaction logic allowed transactional code to be generated "from scratch". In several occasions, this code was not only much simpler than elaborated schemes including transactional components, but also much more efficient. This relative failure at generating efficient programs based on high-overhead component technologies is a recurring theme of our experience, and should probably be construed instead as the success of an approach where the *what* is more important than the *how*: if a simple generated transactional scheme is more efficient, then it should be considered an optimization and used.

7. OTHER CASES WITH UML

7.1 Global Information System Experience

A global network equipment maker wanted to build a multi-country client/server application to connect call centers in several countries (in Europe and South Africa) and share information. The supporting network included an outsourced WAN (Wide-Area Network) and was to be implemented with EJBs (Enterprise Java Beans), because the technology was just out and it sounded hot (sic).

This first try at generating Java components was not conclusive [4]. The fat clients calling heavily upon EJBs through a high-latency WAN was the major setback. The generators can only be incriminated because they didn't help and we were used to have it so easy. Nevertheless, schemes to generate source code for component-based platforms were first elaborated during this experiment.

7.2 CTI Experience

A major telecom operator wanted to gain more control over their PBX in order to propose high-level Computer-Telephony Interaction (CTI) services. They have built a PBX/CTI platform along with composable services.

The target libraries for PBX control and state-machine mechanisms are implemented in Java. The generator uses template-driven producers and hooks. The target libraries have been designed with UML and are available as models upon which to built applications.

Template-based producers allow to express critical code generation segments directly in the target semantics while retaining the full expression power of the language used to build the generators. Model Graph manipulation and target language semantics can be better uncoupled. With simple tools, the users are empowered to change the generated code.

7.3 Telecom IS Experience

Alcatel defined its Domain Specific Modeling Language as a UML profile, as well as a process to develop IS with it. They wanted to extend a commercial CASE-tool for the support of the profile, the process, classical and in-house design patterns and code generation. Plugs-ins were developed to satisfy the requirements.

The external UML modeling tool was interfaced through plug-ins in Visual Basic and C++. The same mechanisms were used to implement a domain specific UML language and design patterns (classical and domain-related). The target platform, written in C++, includes among others, the association, aggregation, composition library COLOR++ by Franck Barbier [3]. For the trivia, Franck Barbier had already contributed to the initial core of the OMT modeling tool.

The fully procedural code generator had to be hard-coded.

This confirmed for us that the most efficient metamodel to handle UML inputs and transforms is not necessarily a direct representation of the metamodel in the standard. As in any domain, the analysis model has to be designed and the design model is not necessarily isomorphic to the analysis model. This truth applies to UML tools as well.

We also learned that the actual model graph, as implemented in the actual metamodel, and its control were crucial to the efficiency of the generators, i.e. the ability to modify the implementation of the metamodel to accommodate the generator is a key success factor. The choice of extracting information, e.g. through XMI, and parsing it makes for unpalatable edit/generate/run cycles.

8. MOBILE SERVICES, UML 2 AND MDA

France Telecom wants to propose games on mobile devices (mostly telephones, at the time of writing). The domain model for online services (see above Online Services Experience) has been slightly revised to take into account some specificities of this subdomain. The pilot game selected is a massively multiplayer "dungeon", where users can define rooms, objects and characters (both player and non-player characters), through simple scripting accessing a rich dedicated library for universe building. Then these users and others can navigate and explore the world through SMS messages and responses.

The target platform managing the SMS messages (as well as accounting, etc.) is the Soprano Java framework [24].

The adventure/exploration game is generated in Python, its uses Game Logic similar to those of the Python Universe Builder (PUB) [17], and Beyond [2]. AI co-routines use Python 2.2+ generators. They are represented with Domain-Specific Notations in models. Persistence is implemented through an aspect. Python made it easy to implement a simple Aspect-Oriented Protocol.

The game DSL can be implemented more efficiently thanks to the multi-paradigm nature of Python (used here in the sense promoted by [10]). The generator can select the more appropriate among procedural, functional, logical, aspectual, can resort to meta-programming (i.e. generating metaclasses) and to generate and call C++ code, even to inherit Python classes from C++ classes. The resulting Python code is still fully integrated and readable.

One of the standing issues in separating the DSL from the implementation frameworks is: Can we represent in models the self-modifying code used in some of the state-machines? Another is: How should we model prototype classes (**object** oriented) when the DSL is built upon a type-oriented meta-model (UML)?

This set of generator is the first to use Python as well as a script language for *walkers* and templates-based *producers*. A scheme of *postponed generation* has been introduced. In essence, it allows a first generation onto an integrated simulation platform, to debug the model, and much later a generation for the real target

This present experience should yield some interesting results. It is developed with the online services DSL presented above, but the Game Logic has its own DSL, tentatively developed upon UML 2.0. Weaving the interactions of the two is the task at hand.

We are also experimenting with event-driven *walkers*, similar to the control structures described in [5], not as a replacement, but as a complement to other types of walkers.

9. CURRENT EXPERIMENTATION

We have not been completely deterred from generating code for existing component platforms. We are currently investigating efficient component based targets [12]. Antonio Carrasco-Valero, a leading force of the EDOC (Enterprise Distributed-Object Computing) has been contributing very valuable input, and we believe than component "languages" will eventually allow us to

forecast the optimizations necessary for efficient component composition.

Inputs from Franck Barbier at Université de Pau help us shape a formal core engine for pre-simulation, based on UML 2.0. We aim to reduce generation for the simulation platform in the deferred/retarded generation scheme presented above and replace it with more direct execution. This will allow to enhance the completeness of the semantic specification and in turn permit a better code generation. Franck also shares his knowledge with us in issues dealing with inheritance/delegation of states in the complex state machines of the Python-based online game.

Cecilia Haskins at the University of Bergen is in the process of codifying best practices for systems engineering. Those could potentially influence not only the shape of role-based tools, but also the logic prevailing during early selection of which generators should cooperate to produce the best output (in terms of expectations).

France Telecom is integrating web services in the domain model and target platforms.

10. DISCUSSION

On the large projects that provided the context of our work, success and failure factors are numerous. The tool-builder is a small contributor to the overall effort. Abstracting key success factors for a contributing member is a difficult art. Still, applying smart code generation to large models in various domains, we learned a few lessons.

Code generation is helpless/useless without a deep and continuous domain analysis resulting in a generic domain model. It is near useless as well without at least a deep survey of variations in target platforms (hardware and software layers) and preferably explicit models;

Not one type of generator is ever enough, combination/composition/cooperation of several types of generators provide the results. A synthetic approach to model compilation needs to be explicitly researched, not just code generation techniques, and present the practitioners with better solutions for generator composition.

In practice, forward refinements need to be completed with some backward extraction from the debugged code. Horizontal transformations must be handled with care if the target is in source code: intention must be preserved and the weaving of aspects must not occlude it.

In opposition to some related fields, such as AST "massaging", MDA generation cannot be extracted from its context. Human and organizational factors influence deeply on the capacities and intentions of generators, and must be expressly modeled and taken into account. For most roles, people should deal only with the DSL and not be exposed to UML. The UML core engine must remain hidden. The tools must become multiple and provide interfaces for the job at hand, they will just "happen" to be implemented on a UML core engine.

Information Systems models frequently embed incomplete and ambiguous specifications. This is due to the volatile nature of the applications and to the lack of UML culture in the market. 95% of the models we deal with are still objectified entity-association models. Frequently the problem with generating MIS applications does not lay in the optimizations, but in the ability at all to generate smarter code than classes, accessors and optionally method names. And doing so with requesting as little as possible supplementary information to the designers. UML 2.0 should help marginally on this point, as it removes ambiguities from the meta-model, but it won't remove ambiguities in the intentions of the designers.

On the other extreme, some of the network telecom models we dealt with have so many detailed collaboration diagrams that no room for interpretation is left. For those, we believe that techniques borrowed from [5] and other researches, applied on UML 2.0, will provide valuable answers.

A large part of our effort (up to 40%) was always aimed at refining the metamodels to better suit our purpose. UML 2 was the result of a more rigorous approach, as opposed to the political bickering of UML 1 and 1+. UML 2 will most definitely be a major improvement for code generator makers.

In conclusion, our experience globally validates the MDA approach. Nevertheless, it is more ambiguous regarding component-based systems as target platforms. Not only have those proven difficult to handle, hard to fine-tune and evolve according to vendor changes, but they also proved not to be the most efficient implementation in a number of cases. We regard those facts as a proof of the relative immaturity of component standards, which could change quickly with EDOC and other initiatives, rather than a full invalidation.

11. ACKNOWLEDGMENTS

Our thanks to Joel Bayon, Gérard Bunel, Jean-Pol Castus (IBM ISDC), Alistair Cockburn, André Deleplanque (France Telecom), Cecilia Haskins (University of Bergen), Ralph Hodgson, Anna Karatza, Marc Skipper and many others that I cannot name, for their support and contributions over the years.

12. REFERENCES

[1] Andersen, E.P., Conceptual Modeling of Objects, a Role Modeling Approach, Dr Scient thesis 4 November 1997, University of Oslo, at ftp://ftp.nr.no/pub/egil/ConceptualModelingOO.ps.gz

[2] Asbahr, J., Developing Game Logic: Python on the Sony Playstation 2 and Nintendo GameCube, O'Reilly Open Source Convention, July 2002, at http://conferences.oreillynet.com/cs/os2002/view/e_sess/2423 see also http://www.asbahr.com/beyond.html

[3] Barbier, F., Henderson-Sellers, B., The Whole-Part Relationship in Object Modeling: A Definition in cOlOr , Information and Software Technology, 43(1), Elsevier, pp. 19-39, 2001 (ISSN: 0950-5849)

[4] Barbier, F., Reich, G., UML and Software Components, CITE-M 2001, November 2001

[5] Biggerstaff, T., A New Control Structure for Transformation-Based Generators, in Software Reuse: Advances in Software Reusability, Proc. of Sixth International Conference on Software Reuse, Springer-Verlag, June, 2000

[6] Bouron, T., Deleplanque, A., and Douget, J., TIMODE: a Testbed for the Interchange of Multimedia Objeccts in a

Distributed Environment, TOOLS EUROPE 1995 Proceedings, March 1995

[7] Cockburn, A., Surviving Object-Oriented Projects, Addison-Wesley Publishers, December 1997

[8] Cockburn, A., Writing Effective Use Cases, Addison-Wesley, January 2000

[9] Cook S and Daniels J, Designing object systems: Object-oriented modelling with Syntropy, PrenticeHall, 1994

[10] Coplien, J., Multi-Paradigm Design for C++, Addison Wesley Publishing Company, October 1998, ISBN : 0201824671

[11] Czarnecki, K., and Eisenecker, U.W., Generative Programming, Addison-Wesley, 2000

[12] Daniels, J.,Cheesman ,J., UML Components: A Simple Process for Specifying Component-Based Software, Addison-Wesley Professional, October 2000, ISBN : 0201708515

[13] Harmon, Paul (ed.), Object-Oriented Strategies, vol. VII, n°4, April 1997

[14] Haskins, C., Schuster, C., A Picture's Worth: Use of Models to Facilitate Knowledge Transfer during the System life Cycle, INCOSE 1995 Annual Symposium Proceedings, CD-ROM version

[15] Haskins, C., Model Driven Development: Practices and Tools, INCOSE 1999 Annual Symposium Proceedings, June 1999, CD-ROM version

[16] Kiczales, G, des Rivières, J., Bobrow, D.G., The Art of the Metaobject Protocol, MIT Press 1991

[17] Python Universe Builder, at http://py-universe.sourceforge.net/

[18] Reenskaug, T., Working with Objects: the OOram Software Engineering Method, Manning, 1996, out of print, revised edition as of February 2001 at http://www.ifi.uio.no/~trygver/documents/book11d.pdf

[19] Reenskaug, T. (ed.), Reich, G.P., and Cockburn, A., The OOram Meta-Model, OMG OA&D RFP-1, version 1.0, January 1997, available at http://www.omg.org/docs/ad/97-01-15.pdf

[20] Reich, G., Object-Oriented Business Engineering, the RROAD to Success, Object India 1997 Proceedings, January 1997

[21] Reich, G., Application Services, full day tutorial, CD-ROM Proceedings of ROOTS 2000, May 2000

[22] Reich, G., Online Services and UML, Proceedings of Forum Développement, January 2001

[23] Rumbaugh, J., & al., Object Modeling Technique, Prentice Hall, 1991

[24] Soprano, A Mobile Application Framework, at http://www.soprano.com.au

[25] Strysick, D., Pluggable Use Cases, CD-ROM Proceedings of ROOTS 2002, April 2002