MDSOFA: A MODEL-DRIVEN SOFTWARE FACTORY

Benoît Langlois & Daniel Exertier

THALES Research & Technology France
Domaine de Corbeville 91404, Orsay, France
{benoit.langlois, daniel.exertier}@thalesgroup.com

Abstract

A major issue in software engineering is software production improvement. This paper studies the objectives and the features of a model-driven software factory contributing to automate the production of software systems in evolving environments (specifications, standards, technology, and tools). Through these considerations, this paper introduces MDSoFa, a Model-Driven SOftware FActory tool developed at THALES meeting this need, and a set of technical and methodological lessons learned from this tool implementation and usage.

1. Introduction

Production automation is a recurring economic and organizational preoccupation. The leitmotiv is to rationalize production for improving productivity, quality and flexibility, in order to reduce costs and to increase profits, financial as well as technical. For software, the objective is the same. In this paper, we analyze the combination of two approaches, model-driven engineering and software factory, to rationalize software production. The main interest of model-driven engineering is that model is the primary type [3] for developing systems, *e.g.* from requirements down to code, and this in different perspectives, as domain, technical, or process. The main interest of software factory is software production automation, for going from a handcrafted to an industrialized software production and for allowing development time reduction and software quality improvement. In this vision, a model-driven software factory is a software factory where models are central with the main objective to ease and industrialize software production.

Section 2 clarifies the stakes of a software factory in the model engineering context. Section 3 presents the main characteristics of a model-driven software factory to understand its specificities. Section 4 presents MDSoFa, a Model-Driven Software Factory tool developed by the THALES MIRROR Pilot Programme[1]. Finally, section 5 lists a set of lessons learned on model-driven software factory techniques, while specifying, developing and using MDSoFa.

2. Stakes

Model engineering is a growing trend for building systems that will be used and maintained for many years and sometimes for many decades, as in the defense and aerospace domains covered by THALES, while standards, technologies, and platforms evolve inevitably. Answering to these combined needs, the Object Management Group (OMG) is standardizing the MDA® [14]. In the meantime, what do development teams really expect? They need mature modeling tools and modeling processes to produce the expected assets with efficiency in order to deliver the system to be developed in due time, with the required functions and the required qualities. In model engineering approach, UML® is the most widespread modeling language and most of the industrial modeling CASE-Tools are today UML®-based. The first

[1] MIRROR has the mission to put the MDA® (Model Driven Architecture) vision at work.

paradigm shift in modeling is to develop productive models rather than contemplative models only drawing informal sets of concepts about the system to be developed. The paradigm shift with model-driven software factories is to automate as far as possible the software production from models. This approach implies tools built in this purpose but also adapted methods of work and software factory awareness from the development teams.

Industrialization stake. Software factory is a generative technique with the purpose to reduce software complexity and ease software production 1) in abstracting raw or complicated software aspects, 2) in producing software in series. Software factory is a means for going from a handcrafted to an automated software production, ensuring asset quality and improving the reactivity facing the specification, methodology, standard and technology evolutions.

Capitalization stake. Besides, software factory becomes a vector for capitalization. Large-scale developments imply multiple pieces of sharp expertise[2] (domain, technical or a process). The stake is to capture generic expertise to reuse it in different contexts during the software mass-production. (Fundamentally, capitalization means that a production can be reintroduced later in the development process, creating new software value.)

Maturity level improvement. In this perspective, further than a technical viewpoint, a major issue is to improve progressively the software production maturity level. For illustration, Table 1 describes a three-level maturity model using a model-driven software factory approach, from "Repeatable" to "Managed" to "Optimized" developments.

Maturity level	**Indicators of the software production**
Repeatable	• Software production complexity: the software factory applies a systematic and repeatable asset production, *e.g.* code generation, model generation from patterns. • Process awareness: individual and common practices are applied, meaning that the asset architecture of the software factory is not completely clarified.
Managed	• Software production complexity: reflective and adaptive approach of the software factory, the asset architecture of the software factory is managed, a product-line approach is supported, *e.g.* a model-driven engineering chain is automatically produced resolving different variation points on functions, target technical platforms... • Process awareness: an architect assumes the long-term vision of the core metamodels and the tools involved in the software factory technique; the software production process is predictive, because under control.
Optimized	A continuous feedback contributes to improve the software production and the ROI (return on investment) of the software development.

Table 1. A three-level maturity model with a model-driven software factory approach

Software production improvement requires strategic decisions consistent with a software factory approach. For illustration, and to better understand how to build and use a software factory tool, mention two best practices.

Core technology capitalization. Experience, refactoring, and architecture vision all contribute to elicit common and recurrent assets. For large developments with long life cycles, it is essential to build a core technology that capitalizes the transversal experience from different developments. As shown in Figure 1.a, such a core technology can be reused for building system, software and other engineering chains. This core technology requires a clear architecture, as a clear relationship between used and produced assets (Figure 1.b).

[2] An expertise is any software description consumed or produced by a model-driven software factory. It can be a recurrent solution, as a pattern (model transformation pattern, process pattern, etc.), or a specific description, modeled or not, as a best practice or some code.

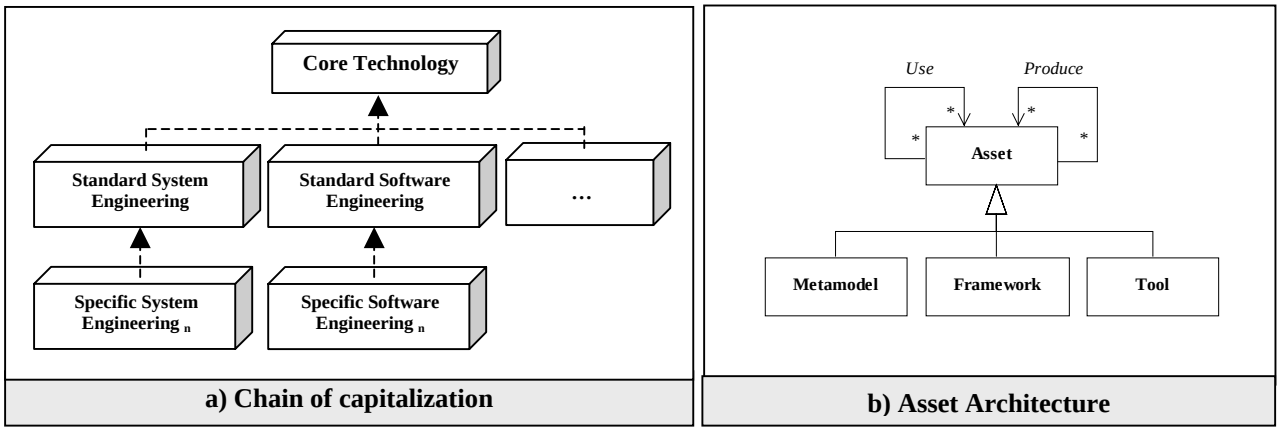

Figure 1. Technology capitalization

Develop synergies between standards, processes, and tools. A clear and predictive method of work needs standard domain, technical or process assets. Standardization can be international, enterprise, or project wide. Besides, a rationalized process implies not only rationalized activities but also engineering tools adapted to the defined engineering processes. The common denominator is to develop a systematic method of work, the way for automation. The interest is to create a synergy between standards, processes and tools, raising together the software production maturity level.

3. Features

After this first overview, it is now time to address the main features and usages of a software factory in the model-driven development perspective.

Typology of the model-driven software factory assets. A software factory automates output assets production from input assets. As shown in Figure 2, a software factory for model-driven development handles four main kinds of assets:

- *Metamodel*. A metamodel can describe 1) a domain, *e.g.* air traffic control, 2) a technology, *e.g.* QoS [20], platform metamodel, but also a model transformation metamodel, or 3) a methodology, *e.g.* system engineering [19].
- *Expertise*. Here, an expertise is a capitalized piece of a software description. An expertise can be large, in the sense that it can be built upon composition (merging) of finer grain pieces of expertise. In process modeling, expertise corresponds to the SPEM guidance [18] but it may be also purely technical, without process consideration. An expertise is considered generic when reusable in different contexts while an expertise is considered specific when context-invariant. For instance, patterns are generic expertise whereas a set of textual or modeled requirements is a specific expertise. A generic expertise can be transformed into a specific expertise. For instance, a pattern becomes a specific expertise when applied to a specific design or programming context. An expertise description can adopt different forms: some text (code, documentation), a DSL description, etc.
- *Tool*. A tool is any device realizing actions and producing outputs from inputs. For instance, Perl or model transformation engine are tools. By extension, Perl scripts and model transformation scripts can also be viewed as tools.
- *Framework*. A framework is a structure for recurrent solutions. For instance, a software factory can provide modeling chains (system, software engineering modeling chain, etc.) built upon a modeling framework providing a set of common modeling services.

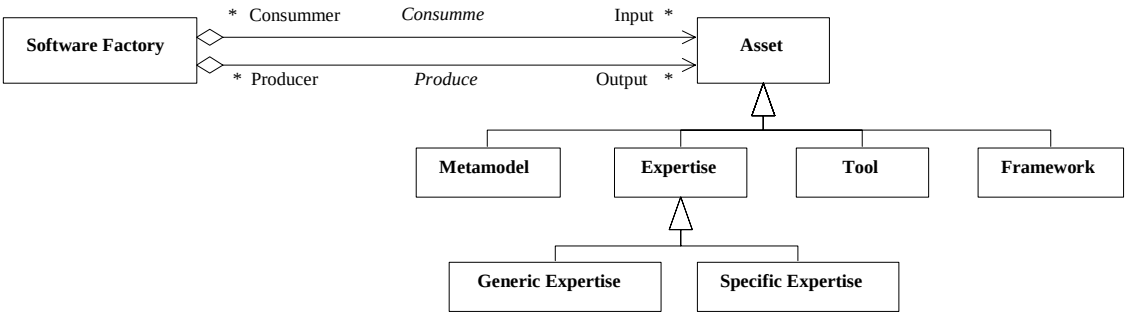

Figure 2. Input and output assets of a software factory

Cartography of the model-driven software factories. A model-driven software factory is a combination of metamodels, expertise, tools and frameworks for producing output assets in an industrial way, that can be also metamodels, expertise, tools and frameworks, *i.e.* recursively, a model driven software factory can produce a model-driven factory. Depending on the focus of the produced assets, a software factory has the following functions:

- *Model factory*. In this case, models are produced automatically from models. For instance, a model transformation can be deduced from the application of a model transformation pattern on a domain metamodel.
- *Expertise factory*. In this case, the software factory produces specific or generic expertise from specific or generic expertise. For instance, model checks and wizards can be produced from a methodological metamodel and a generic expertise for model checking and assistance.
- *Tool factory*. In this case, the software factory produces tools or executable environments in a tool, as a tool-specific modeling chain.
- *Framework factory*. In this case, the produced asset is a framework.
- *Software factory factory*. As mentioned above, this specific case covers the reflective approach when all types of asset are involved in input and output of the software factory for producing a software factory.

Modeling chain factories. We focus now our interest on software factories producing modeling software factories (*Figure 3*), involving cooperation of heterogeneous domains and pieces of expertise, with development qualities to be respected. In the model-driven perspective, a software factory production can be seen as a mapping execution: the modeling environment is modeled at the metamodel level and the result of the software factory is the modeling environment that will be used by modeling users. *Table 2* exemplifies a set of assets that can be produced by such a software factory.

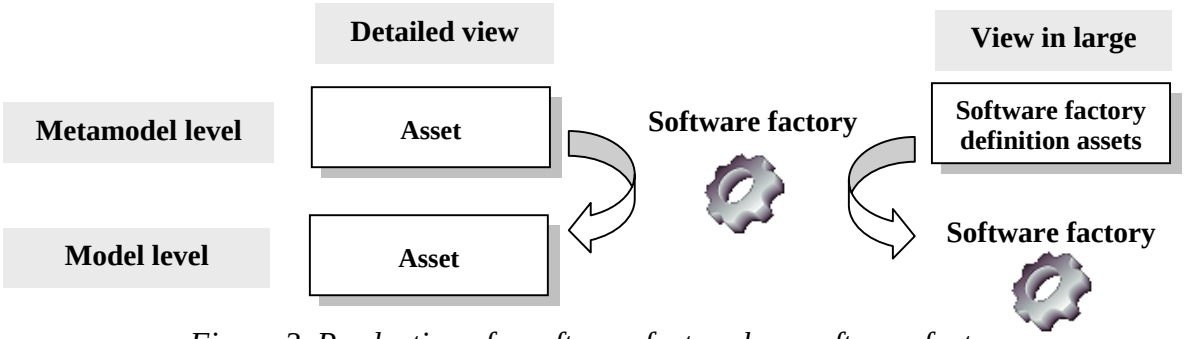

Figure 3. Production of a software factory by a software factory

Description at the metamodel level	Description at the model level
Metamodel	A UML profile
Engineering process, *e.g.* expressed in SPEM	Process control for modeling, for instance with modeling commands or wizards
Reusable model transformation rules (patterns, architectural patterns, QoS…)	Model transformations adapted to a domain
Documentation rules at the metamodel level	Context-specific documentation

Table 2. Examples of information mapping between metamodel and model levels

This software factory mapping is comparable to an abstract to concrete syntax mapping, or, in MDA terms, to a PIM (Platform Independent Model) to PSM (Platform Specific Model) mapping. A further step consists in instantiating the same software factory for different modeling tool platforms, each having its own specificities (tool-vendor UML metamodel, language, packaging, deployment protocol…) where target platform becomes a parameter to be considered during the software production [4]. This is key for large companies where methodologies and practices are similar with different modeling tools.

4. MDSoFa

The previous sections have presented the main objectives and features of a software factory. This section presents MDSoFa, a **M**odel-**D**riven **So**ftware **Fa**ctory tool.

MDSoFa, a core technology tool. The MIRROR pilot programme is developing a MD suite, a set of model-driven methodologies and tools, to the service of the THALES business units. MDSysE is a tooled methodology of this suite dedicated to Model-Driven System Engineering, in line with SysML [19]. MDSysE is being deployed into some THALES business units. MDSoFa is one of the core technology tools that have contributed to the realization of MDSysE in supplying basic model manipulation APIs, as well as higher level functions for modeling assistance, such as wizards.

Technical features of MDSoFa. MDSoFa is been developing with the Softeam's Objecteering CASE-Tool but with the objective to produce assets targeting different modeling platforms. MDSoFa is an environment for building software factories with the following core functions:

- *Metamodeling*. To define metamodels, MDSoFa simply uses an EMOF level of description (Class, Attribute, Operation, Generalization…) [15].
- *Metamodel mapping*. For relationships between languages, a MOF to MOF mapping allows correspondence between metamodels. The MOF to UML mapping is the most used mapping.
- *Queries / Views / Transformations (QVT) rule definition.* To express model transformation metamodels, MDSoFa separates QVT specification from QVT implementation. The specification language is a graphical declarative language identifying input and output metaclasses of a QVT rule, and the relationships between QVT rules (rule composition, rule inheritance). For the implementation part, MDSoFa provides an internal language to substitute, during the software production, templated expressions with the result of metamodel queries (including queries on metamodel mappings and on model transformation metamodels). This template approach is a key point of the software factory for producing QVT rules in series from metamodels, in others words for transforming a generic expertise (templated rules) into a specific expertise.

- *Aspect separation.* To avoid monolithic developments, the notion of aspect has been introduced in MDSoFa, offering different viewpoints on the same metamodels, as model structure manipulation, model quality, or presentation. The aspect weaving may be solved during the software production with the query mechanism.
- *Product-line.* To customize a standard product, a simple product-line approach has been introduced for targeting different modeling platforms, for managing, for instance, different UML versions or variations on domain metamodels – see Figure 1.
- *Deployment.* For modularity and reusability, the software assets are grouped into autonomous and deployable units, called MDA Components [9][5][17]. As a precondition, metamodels and their aspects shall be organized on that purpose; moreover, a flexible logic of deployment has been initialized in MDSoFa for targeting different modeling platforms, each requiring specific assets with specific formats.

The MDSoFa process. For producing software and modeling tools in series, the MDSoFa process is divided into four sub-processes (Figure 4).

Process 1: Metamodel development process. During this process, a METAMODEL DESIGNER is in charge to define the MOF metamodels, the MOF to MOF mappings and the aspect definition. The METAMODEL DESIGNER and the SOFTWARE ARCHITECT must share the same vision in large, especially on the core technology and domains metamodel organization, and in detail, as the UML mapping.

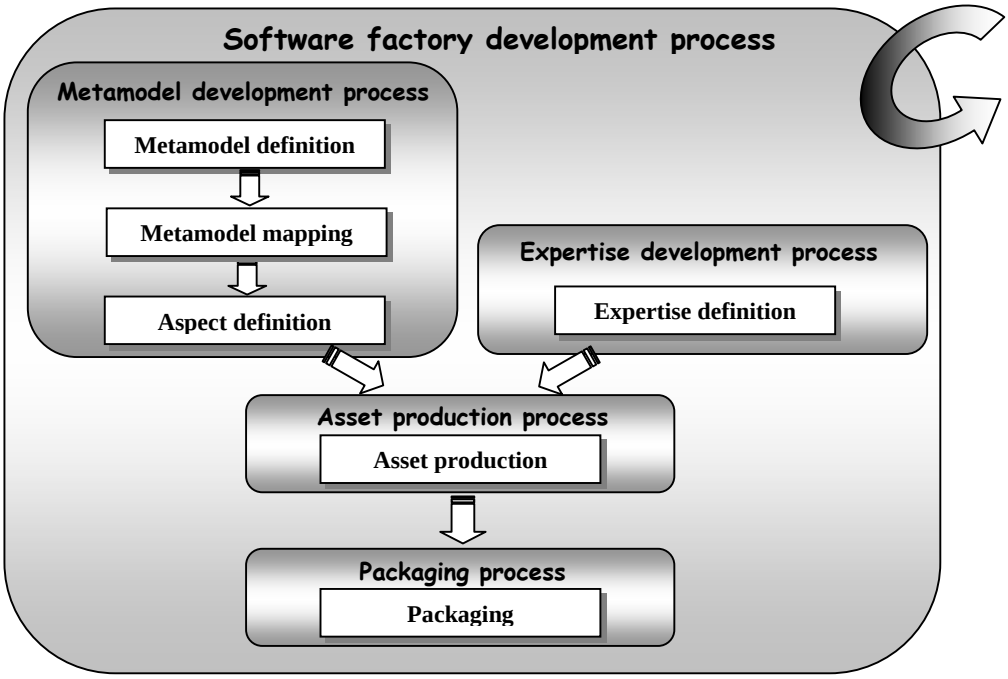

Figure 4. Software factory development process with MDSoFa

Process 2: Expertise development process. During this process, an EXPERTISE DESIGNER is in charge to define the specification of the QVT rules and an EXPERTISE DEVELOPER their implementation. To be generic, an expertise has to be domain-independent in order to be applied in different domain contexts. Graphical specification view is a good way for communication, especially with the SOFTWARE ARCHITECT. For the implementation, different languages may be used in function of the target platform, this implying different EXPERTISE DEVELOPERS. EXPERTISE DESIGNERS and EXPERTISE DEVELOPERS have to share the

SOFTWARE ARCHITECT vision, particularly on the software organization (core technology architecture, pattern elicitation, framework infrastructure…) and on the non-functional aspects (maintainability, usability, performance…).

Process 3: Asset production process. This process is the heart of the software factory where target assets are produced in series. An asset production in MDSoFa combines metamodels and generic expertise to produce output assets. Asset production uses the pattern matching technique for selecting QVT rules (expertise) involved in the concerned aspect(s), and the template technique for transforming a generic expertise into a specific expertise.

Process 4: Packaging process. With MDSoFa, the packaged assets are UML profiles, code and configuration files compliant with the modeling tool format.

Properties of the MDSoFa process. The MDSoFa process tries to defend a set of values:

- *Process customization*. The described overall process is a backbone can be enriched with process plug-ins. For instance, the introduction of some product-line technique involves extra activities such as feature specification.
- *Cooperation*. The process organization authorizes multiple and cooperative development processes. For instance, the expertise developed during a system engineering software factory development can contribute to the expertise of the software engineering software factory development.
- *Iteration.* Software mass-production with the generative technique and the aspect separation favors multiple iterations. The boundary of the MDA Components defines the building areas.
- *Agility*. Software factory process and tool have to share Agile Modeling values [1] for easing and improving the software production in the modeling context. This means to propose efficient principles as embracing change (as metamodel and expertise evolutions), allowing rapid feedback from the users using the assets produced by the software factory, assuming simplicity (languages, facilities… in the software factory tool), easing communication (as with the graphical QVT specification view), productive assets (metamodels, expertise) over contemplative assets (documentation).

Validation. The following table sums up a set of results on MDSoFa.

Modeling Chain	Modeling chain for system engineering (MDSysE): • 14 metamodels, 130 metaclasses, 250 associations • 15.000 generated methods
Target modeling tools	• Softeam's Objecteering (J language) • I-Logix' Rhapsody (Visual Basic with the COM API)
Framework	The generated MDSysE framework is extensible by the developers.

Table 3. MDSoFa facts and figures

5. Lessons learned

We list now a set of lessons learned from MDSoFa classified into three categories: 1) lessons for a core software factory tool, 2) lessons for producing modeling chains with a model-driven software factory, 3) lessons on methodology.

Lessons for a core software factory tool:

- Mandatory: for being insensitive to ever-evolving standards, methodologies, technologies and tools, the internal language of the software factory tool shall be independent of the used MDD (Model-Driven Development) environments (UML, domain languages, platform languages, tool language…) and very reduced. This is the unified language of the software factory.
- Mandatory: the minimal functions required for a software factory shall be: 1) a model editor for metamodeling, extended with a constraint language, 2) a metamodel-to-metamodel mapping facility, 3) a language for manipulating generic and specific expertise, 4) a tool for producing assets in series, 5) a set of facilities for building easily a deployable product for a defined target modeling tool.
- Mandatory: the expertise capture and the customization process shall be easy and efficient.
- Mandatory: the software factory performance shall support asset production for large-scale developments.
- Mandatory: the software factory shall be extendable (module, plug-in, etc. approaches).
- Optional: for managing and producing in parallel multiple products, the software factory shall integrate a product-line approach.
- Optional: the software factory shall offer architecture and tools for managing multiple pieces of expertise built in parallel by different members of a project team.
- Optional: the software factory shall be able to interoperate with external tools.

Lessons for producing modeling chains with a software factory:

- Mandatory: methodological impacts (efficiency, organizational, etc.) of the produced modeling chain shall be measurable. A modeling chain is not technology driven or tool driven, but methodology driven. At the metamodel level, a methodology describes a set of coordinated activities realized by actors to produce defined work products. The stake is to define at the metamodel level, for multiple software factory instances meeting different types of project teams needs, a methodology improving the modeling process maturity.
- Mandatory: modeling chains shall be agile. A modeling chain shall contain facilities for guiding the end-users and easing their modeling tasks. This includes guidance and assistance for abstracting information.
- Mandatory: for multiple standards, methodologies, technologies and tools, the reusable expertise shall be refactorable and easily instantiable to rebuild existing framework and tools as well as to build new ones.
- Optional: a modeling chain shall be extendable, as with the Eclipse/EMF plug-in technique, or connectable to others assets.

Methodological lessons:

- Mandatory: a pyramidal building shall be adopted. For controlling the maturity level of a software production with a software factory, automate first the basic functions, *e.g.* model management API's; then automate the production of more and more complex functions until reaching the production automation of the high-level functions.
- Mandatory: generic expertise development shall differentiate the specification view from the implementation view. The specification view represents the black-box view of a reusable solution. The implementation view, the white-box view, may have different concrete representations (textual / graphical view; DSL; programming-language…). The stake is to differentiate design from platform aspects.

- Building units shall be small with clear objectives. Prefer generation units with small metamodels and separate the different aspects of generation. The stake is to partition the problems and the solutions for mastering the software production complexity.
- Mandatory: a software factory shall be a tactical tool, not a strategic tool. A software factory is a powerful tool to achieve a production in series with a high level of quality. However, at a given level, except non-functional considerations such as performance or storage, the mass-production effect disappears. For instance, 1,000 or 10,000 or 100,000 (inflation is easy) produced methods have, at a high level, the same value: the production is measurable by its effects, not by its content. A software factory will not be able to replace strategic decisions. It is just a means to increase the software production. Software activities rationalization and systematic solutions development are the real input of a software factory to improve the maturity of the software production.
- Mandatory: flexibility for producing in series shall be kept until decreasing profits. Flexibility reflects the ability to evolve. Generation from models increases this flexibility. However, flexibility has a cost, growing with its complexity. Specific expertise, first with lower costs, increases the system rigidity (impossibility to evolve). Which is the flexibility level to adopt? In fact, it is better to introduce flexibility in a system only when needed to avoid useless costs and proceed to the required refactoring.
- Mandatory: a unified formalism and a unified methodology shall be adopted for reflexivity and integration. Uniformity is required to avoid multiple and heterogeneous developments and to manage automated software production.

Finally, here is listed a set of general recommendations for building a software factory:

- Hypothesis 1: Respect of the target
- Hypothesis 2: Respect of the budget
- Principle: Be efficient, the least effort for the best effect
- Corollary: Go to the essential
- Rule 1: Be systematic until decreasing profit
- Rule 2: Be optimal until decreasing profit (no useless element, factorization at the same level or at the metamodel level)
- Rule 3: Your system is in balance else reconsider it in detail or in large with the rules 1 and 2 or proceed to a refactoring.
- Rule 4: Practice to validate

6. Perspectives

MDSoFa is expected to be actively used for offering modeling chains to THALES Business Units. On this account, MDSoFa product-line facilities have to be improved for easing the creation of new modeling chains with their own specificities. Performance is also a recurrent concern for producing various and large modeling chains.

Acknowledgment

We would like to thank Serge Salicki, manager of the THALES MIRROR pilot programme, Pascal Bizien, Madeleine Faugère and Eric Jouenne, members of MIRROR, Nicolas Farcet manager of the CARROLL research program, and the members of CARROLL.

References

[1] Ambler, S., The Object Primer, Agile Model-Driven Development with UML 2.0, Cambridge University Press, 3rd edition, 2004.

[2] Bass, L., Clements, P. Kazman, R., *Software architecture in practice*, SEI Series in Software Engineering, 1998.

[3] Bézivin, J., *From Object Composition to Model Transformation with the MDA*, proceedings of TOOLS'USA, Volume IEEE TOOLS-39, Santa Barbara, August 2001.

[4] Bézivin, J., Farcet, N., Jézéquel, J.M., Langlois, B., Pollet, D., *Reflective Model Driven Engineering, UML 2003 Conference*, October 2003, San Francisco. Springer, LNCS 2863, 2003.

[5] Bézivin, J., Gérard, S., Muller, P.-A., Rioux, L., *MDA components : Challenges and Opportunities*. Metamodelling for MDA, York, England, November, 2003.

[6] Budinsky, F., Steinberg, D., Merks, E., Ellersick, F., Grose, T.J., *eclipse, Modeling Framework*, the eclipse series, Addison Wesley, 2003.

[7] Cook, S., Kent, S., *The Tool Factory*, OOPSLA 2003 "Generative Techniques in the context of Model Driven Architecture" workshop. October 27, 2003. http://www.oopsla.org/oopsla2003/files/ws-3.html.

[8] Czarnecki, K., Eisenecker, U.W., *Generative Programming*, Addison Wesley, 2000.

[9] Desfray, P., *When a Major Software Trend Meets our Toolset, Implemented since 1994*, OMG document, http://www.omg.org/mda/presentation.htm, November, 2001.

[10] Frankel, D., *Model Driven Architecture, Applying MDA to Enterprise Computing*, OMG Press, Wiley Publishing, Inc., 2003.

[11] Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1995.

[12] Garnder, T., Griffin, C., Koehler, J., Hauser, R., *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final Standard*. Metamodelling for MDA, York, England, November, 2003.

[13] Greenfield, J., Short, K., *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*, OOPSLA 2003 "Generative Techniques in the context of Model Driven Architecture" workshop. October 27, 2003. http://www.oopsla.org/oopsla2003/files/ws-3.html.

[14] OMG. *Model Driven Architecture (MDA)*, Document number ormsc/2001-07-01, July 9, 2001.

[15] OMG/RFP. *Meta Object Facility (MOF) 2.0 Core Proposal*, Revised Submission, ad/2002-12-10, January 6th, 2003.

[16] OMG/RFP. *MOF 2.0, Query / Views / Transformation*, Revised Submission, ad/2002-04-10, Version 1.0, 2004/04, QVT-Merge Group.

[17] OMG/RFC. *RFC Submitted to OMG, Reusable Asset Specification (RAS)*, ad/2003-10-10. October 2003.

[18] OMG/RFP. *Software Process Engineering Metamodel Specification*. formal/02-11-14, version 1.0, November 2002,

[19] OMG. *Systems Modelling Language: SysML*. Version 0.3 (first draft). January 12, 2004.

[20] OMG/RFP. *UML Profile for Modeling Quality of Service and Fault Tolerance Characteristics and Mechanisms*. Revised submission. August 18th, 2003.

[21] OMG/RFP. *Unified Modeling Language: Superstructure*. 3rd Revised submission, version 2.0, ad/00-09-12, April 10th, 2003.