16 Evaluation Framework for Model-Driven Product Line Engineering Tools

J. Oldevik, A. Solberg , Ø. Haugen, and B. Møller-Pedersen

Abstract

Both the model-driven development (MDD) approach and the product line engineering (PLE) approach envisage more efficient system development capable of delivering high-quality products by means of reuse, abstraction, configuration, and transformation. In order to succeed with model-driven product line engineering we need tools that support architects and engineers in tasks such as system modeling, variability modeling, model analysis, model transformation, system derivation, code generation, and model traceability.

Managing and automating these processes and tasks can be complex processes themselves. How to solve these complexities is a current topic of research. Unsurprisingly, no existing tool provides full support for an envisioned model-driven product line engineering approach. However, MDD and PLE are being paid a great deal of attention by the software development community, leading to an increasing number of tools emerging within this area. This is particularly the case for tools supporting Object Management Groups (OMG) envisioned model-driven engineering approach, Model Driven Architecture (MDA).

When exploring tool support for the evolving MDD and PLE disciplines, it can be difficult to know what features to look for and what to expect. This chapter relates traditional model-driven engineering to product line engineering and establishes a general framework for evaluation of tools in this area. The framework is defined in terms of desired characteristics, based on elicited requirements for model-driven product line engineering. It adheres to the general tool selection process described in the ISO 14102 standard. Some example MDD/PLE tools are evaluated using the framework to show its applicability and results.

16.1 Introduction

In product line engineering (PLE), the philosophy is to specify a general product line from which specific products can be derived or configured. The product line is specified at a higher abstraction level than the specific product, and it encompasses commonalities and variability [35]. Chapter 6 defines an approach toward a standard way of representing commonality and variability of product lines. Based on the product line, specific systems are derived by resolution of variability and abstractions. This task is often called *product derivation*. There exists a set of various techniques for performing product derivation, such as model transformation, code generation, and variability resolution. Examples are the approach described in Chap. 15, which looks at using UML for describing static and dynamic PL aspects and deriving products from these, and the approaches described in [1,2,17].

In model-driven system engineering, system development is performed in an integrated environment where models are the main instrument for development and integration. In model-driven development (MDD) processes, an extensive set of different interrelated models at different abstraction levels is developed. These models may range from business models, requirements models, and design models to deployment models and code. MDD envisions efficiency through modeling at different abstraction levels and automatic transformations between abstractions, including the generation of executable code. Thus, an advanced framework for MDD should provide well-structured support for modeling at different abstraction levels, traceability between model elements at different abstraction levels, model transformations, code generation and model synchronization.

MDD and PLE are currently being paid a great deal of attention by both academia and industry. A growing number of tools supporting MDD and PLE tasks are becoming available. In [4], Gartner predicts that model-driven service frameworks with architecture-based code generators will become as prevalent as traditional fourth-generation languages were in the 1990s. Furthermore, the Gartner Group recognizes portfolio management of product lines becoming a peak technology by 2004 [9].

MDD and PLE have similarities and differences, which in combination can provide mutual benefits. For instance, [14] suggests using PLE principles and techniques to define appropriate modeling concepts and thus obtaining proper scoping in an MDD environment, and using MDD principles to model the product line and derive systems. A combined approach has also been investigated in the FAMILIES [11] project [17,34]. Within testing, PLE and MDD share many of the needs. Chapters 11 and 12 show this in their applications of testing product line requirements.

Performing MDD and PLE tasks can be very complex, and tool support is essential to success. Since MDD and PLE are evolving and are relatively recent software system engineering disciplines, there are no well-established guidelines on how to evaluate and select proper MDD and PLE tools. In this chapter, we present an evaluation framework to support evaluation and selection of MDD and PLE tools.

The following sections justify, define, and exemplify the evaluation framework. Section 16.2 describes the relationships that exist between model-driven development and product line engineering. Section 16.3 elicits characteristics for tools and defines the evaluation framework. Section 16.4 shows an example of an evaluation of a selection of tools. Section 16.5 evaluates the tool evaluation framework and draws conclusions.

16.2 Combining Model-Driven Development and Product Line Engineering

Combining model-driven development (MDD) and product line engineering implies that the set of artifacts developed is based on models. In MDD, models are actively used in the development process, both as first-class artifacts and for producing documentation, code, etc.

The product line engineering approach brings concepts such as scoping, product line architecture, definition of domain concepts and components, variation, and product derivation into play [1,2,3,5,6,14]. A well-defined product line inherently specifies the scope of ones domain and defines the common architecture for the set of products in the product line. The variation spans the set of systems that may be derived. The product line approach aims to gain extensive reusability by generalizing a set of related products in a product line.

To combine MDD and product line engineering, it is necessary to specify the product line by models. Models can be specified using a standard modeling language such as Unified Modeling Language (UML). Another trend in MDD is to specify the models using Domain Specific Modeling (DSM) languages, for example using the MetaCase approach [24], Microsoft's software factory approach [14,24], or Xactium [38]. In UML, the profile mechanism [33] provides a means of defining DSM languages, for instance by defining stereotypes of domain specific concepts.

In addition to product specifications the model specifications typically describe the product line reference architecture, domain concepts, patterns, variability specifications, etc. By viewing product line derivation as a special case of model transformation [17], tools supporting MDD should in principle be able to support essential PLE tasks.

Many MDD and PLE approaches are based on component frameworks [8], in which abstractions, concepts, transformations, etc. are defined as part of the framework. The MDD/PLE combination can be implemented as a component framework, in which the product line defines the scope and MDD technologies, such as for instance UML and Meta Object Facility (MOF) [27], are used for specification of the framework. Model transformation technology may be used to perform model transformation and product derivation.

An example of a generic MDD framework that can be customized to support PLE is described in [36]. It provides tailoring to specific domains by means of UML profiles, reusable models, and patterns. UML profiles are used for defining domain concepts and reference architectures. Existing models are prepared for reuse if applicable. Patterns describe standard solutions of recurring problems within the domain. Using a product line to scope the domain, the framework will provide an environment of (a) domain concepts relevant for the actual product line, (b) the product line architecture, (c) common components and artifacts represented as reusable models at the product line level, and (d) variability mechanisms and variability that can be specified by patterns. Table 16.1 shows some parallels between activities of PLE and MDD.

Table 16.1. Parallels between the product line and MDD approaches

product line approach	**model-driven development approach**
scoping	elicitation of requirements
model of product line	high-level model of system
variability resolution and product derivation	model refinement and transformation
model of product	model of system
transformation of product model	transformation of system model
testing of product	testing of system
executable product	executable system

There are many overlaps between activities in PLE and traditional MDD approaches. The major difference is the reuse aspect of a single product line model, the scoping of this model, and the management of variability and commonality within it. The product line model is used for each production of new products. However, this is similar to the reuse of domain libraries (and models) in traditional development. Reuse is the main motivation for product lines. The main differentiating technical factor is the explicit usage of variability and variability resolution in the development process in PLE.

Variability resolution can be viewed as a kind of transformation process, or part of a transformation process, whereby decisions regarding variability in a Product Line Model are taken. The result is a new model, with less (or no) variability. The main difference between variability resolution and traditional MDD transformations is that the latter traditionally has no human interactions during the process.

Looking at the forthcoming standard transformation specification language in OMG, the Query/View/Transformation language (QVT) [30,32], human interactions during the model transformation are not allowed. However, provision of such interactions has been suggested in an evaluation report on QVT [15]. QVT is in the final stages of standardization at the time of writing. It defines a metamodel for transformations and concrete notations for expressing transformations. Two main parts are defined: a relational part that provides a declarative way of specifying and enforcing relationships between metamodels, and an operational part that offers imperative constructs for writing transformations in a procedural style. Another related process in OMG is the standardization of MOF Model to Text Transformations [29]. This process addresses the generation of text from MOF-based models, for example generating code or documentation from UML models. Standards such as these are likely to become key technologies in MDD and play important roles in model-driven product line engineering processes.

An example of a process in which a product line approach is combined with model-driven techniques is illustrated in Fig. 16.1. Here, it is assumed that the product line model is defined by a formal model, e.g., in UML. This model describes different aspects of the product line, such as business aspects, requirements, architecture, design, platform details, and the variability of the product line.

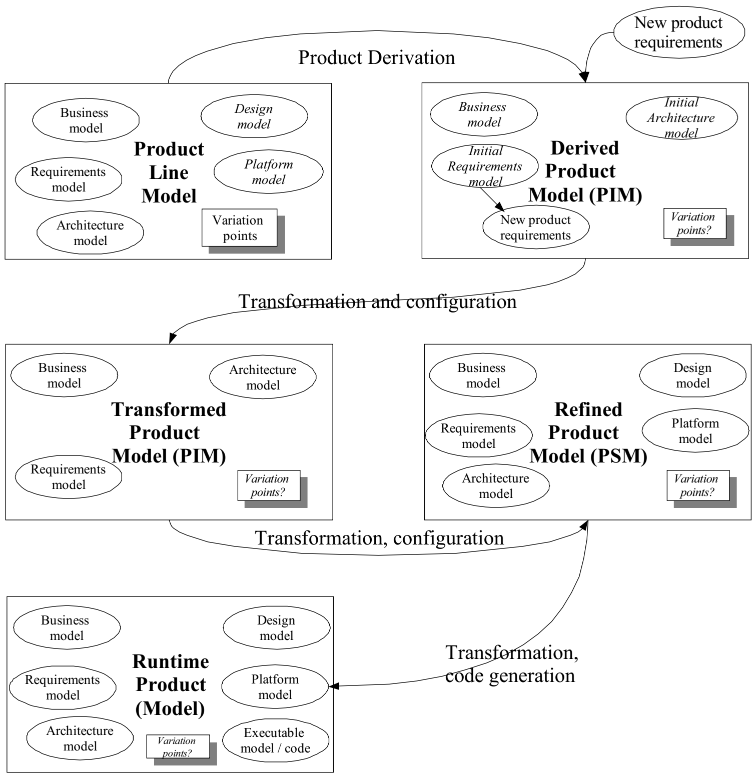


Fig. 16.1. Model-driven product line engineering – example process

When the process of developing a new system is initiated, it is based on a *product derivation* from the Product Line Model. This derivation and the model of the variability in the product line are the main factors that differentiate PLE and MDD. The variability defines a space of possible systems that can be derived. Once this process is completed and the Product Model has been defined, PLE can use the same techniques as traditional MDD.

During the development process, there may be *unresolved variabilities* from the original Product Line Model at different levels, which can be resolved at some point in the process. Consequently, a product line can be resolved, or configured, through a set of steps toward a more specific system.

Following the product derivation come phases that allow for system extension as well as refinement and configuration toward the final runtime system, starting with the Derived Product Model. Here, MDD techniques such as transformation and configuration may be used. New model elements, driven by new requirements, may be introduced on the way. In this kind of process, there may be any number of refinement steps toward different levels of model abstraction. In the example, the terms *platform-independent model* (PIM) and *platform-specific model* (PSM) are used to describe the abstractions.

The terms PIM and PSM are relative to some definition of the platform. For example, defining middleware as the platform (e.g., J2EE, CORBA and .Net), separation of platform-independent and platform-specific concerns occurs when a middleware-independent model (a PIM) and a corresponding middleware-specific model (a PSM) are defined for a particular application. Since the PIM and PSM are relative to the chosen platform, these concepts form a recursive structure, in which a PSM in one context may be a PIM in another. (This terminology is compliant with the MDA [31] definitions of these concepts.) MDD and PLE tools need to provide support for specifying systems at different levels of abstraction. Techniques for model transformation, product derivation, and configuration are keys to the provision of model-driven product line engineering.

16.3 Tool Evaluation Framework

This section defines the evaluation framework by discussing elicited characteristics for model-driven product line engineering (Sect. 16.3.1). The elicited characteristics are analyzed in order to derive the evaluation framework table shown in Sect. 16.3.2. The usage of the evaluation framework is exemplified by evaluating a set of tools (Sect. 16.4). The characteristics have been elicited via a survey of relevant literature, such as [1,2,5,8, 12,15,30,38], through case studies in projects like FAMILIES, COMBINE [7], and MODELWARE [26], and through our own experience gained in the course of development and provision of the UML Model Transformation open source tool [16,37].

16.3.1 Characteristics Elicitation

The following subsections offer motivation for the evaluation framework characteristics.

Support for MDD and PLE Mechanisms

Combining model-driven development (MDD) and product line engineering implies some prerequisites. First, it is required that the set of artifacts developed is in the form of models. Furthermore, model specifications of both the product line and the specific products need to be available. In MDD, the engineering process is driven by the set of prescribed models that need to be developed. Thus, tool support for modeling should be provided, and modeling languages such as UML should be supported.

Providing tailoring and configuration of the tool to better support a specific domain such as support for defining DSM languages (e.g., UML profiling) is important. In [10], several advantages of DSM languages over general purpose modeling languages are discussed. For instance, a DSM language raises the level of abstraction using constructs directly related to the application domain and provides notation close to practitioners' natural way of thinking.

In a combined MDD and PLE approach, the domain should be scoped by the product line. Variability specification and support for transformations and product derivation are other key mechanisms that ought to be in place.

Support for Standards

In many cases, it is important that a tool should support standards, as this caters for open architectures, easy integration, tool interoperability, and tool migration. For a business that is investing in model-driven tool technologies, this is important in order to avoid vendor locking.

The Object Management Group (OMG) is a major standardization organization in the MDD area. It operates through the promotion of MDA, which is based on standard modeling technologies such as the Unified Modeling Language (UML) [33], Meta Object Facility (MOF) [27], and XML Metadata Interchange (XMI) [28]. Ongoing standardization efforts like QVT and MOF Model to Text Transformation are also expected to be key technologies for realizing the MDA vision. These standards target languages for specifying model transformations and code generation, respectively.

MDD and PLE tools should provide mechanisms that support the separation of concerns, such as abstraction levels and views. Most graphical modeling languages provide a set of views through its set of diagram types (e.g., UML, which provides class diagram, interaction diagram, deployment diagram, etc.). Furthermore, the modeling language should support modeling of standardized viewpoints such as ISO RM-ODP [18], as well as any number of user-defined views. Also, features for modeling of PLE variability should be provided. General modeling languages like UML enable modeling of standardized and user-defined views. UML also support modeling of PLE variability to some extent, and UML profiles can be defined to extend the support for variability modeling [17,39].

Product Line Support

Currently, MDD does not address all aspects needed for product line engineering, such as specification and resolution of variability, which are key tasks for PLE.

In PLE, the timing for resolving variabilities may vary. For example, some variation elements may be resolved when deriving architecture models from business and requirements models, others when deriving detailed design models. When deriving implementations as executable code, some variabilities may still remain unresolved. These can be resolved at run time (*runtime variability*), for instance in order to gain context adaptation of the running system.

A tool should provide a flexible way of handling variability resolution. Variability should be permitted to be resolved at different stages in the development lifecycle, and also during run time.

Variation specifications may be inter-related. This may imply that a specific resolution of a variation may conflict with a set of possible resolutions of other specified variations. A resolution of a variation can depend on resolutions of a set of other variation specifications. Management of these kinds of dependences needs to be handled.

The consolidated meta-model for variability described in Chap. 6 provides valuable input for model-driven product line engineering, as it brings forward standard concepts for representing variability. It aims to provide a common basis for implementation by PLE tools.

Process Support

Process support is important in software engineering. Many general-purpose system development process frameworks are available and can be chosen in a combined MDD and PLE approach, for example the Rational Unified Process [22]. In addition to support MDD and PLE tasks, a model-driven product line engineering tool should enable integration and interoperability with standard tool portfolios used in software engineering processes.

In order to support a consistent development process, iterative and incremental development should be supported. In comparison with a waterfall-oriented process, iterative and incremental development caters better for change and for the fact that knowledge of the system and its purpose is typically evolving as it is developed. Iterative and incremental processes have become mainstream in the software engineering discipline, and tool chains used in software development should provide support for this paradigm. For MDD and PLE tools, this includes features such as:

- Support for roundtrip engineering
- Management of traces and relationships between models
- Management of change propagation between model abstraction levels without distorting model consistency

Model Transformation

Providing general refinements of abstract system specifications to more concrete specifications, and eventually to executable artifacts that meet expectations in terms of provided functionality and quality is a complex process.

Tools supporting a combined MDD and PLE approach should offer the capability to *specify* and *execute* transformations between models at different abstraction levels, as well as between models and implementation code. The standardization of model transformation technologies within OMG (QVT and the MOF model to text transformation) will coerce a new level of maturity in this field. Related aspects, such as *traceability support* in transformations and *bidirectionality*, will be of importance in many model transformation scenarios.

When performing model transformation and code generation it is essential to produce the desired results in terms of derived models and code. An important consideration in this respect is production of *expected functionality*; another key aspect is to deliver models and code that specify systems that will adhere to the required *quality* of the provided services. Thus, the specification and consideration of quality of service (QoS) when deriving product models are significant. Quality aspects such as usability, availability, performance, and security need to be managed throughout the system development process. For this reason, the support provided by tools in this respect needs to be evaluated.

Nonfunctional Properties

Nonfunctional tool properties will also be of importance for selecting the appropriate tool. Aspects such as tool pricing, availability, licensing, and maturity of the tool are important properties that affect decisions and the selection of tools. In [20], a more extensive set of nonfunctional properties is defined; subsets of these may be considered relevant dependent on the particular needs of the user.

16.3.2 Evaluation Characteristics

This section presents the evaluation characteristics for MDD tools in general and MDD tools that support PLE in particular. The previous section suggested a number of characteristics that were analyzed with the aim of identifying appropriate criteria within the evaluation framework.

The evaluation characteristics define a set of desired properties. The justification for each of them is indicated by a question, which needs to be answered during an evaluation. The output domain of permitted answers is defined for each question. Some questions have *Yes* or *No* as the output domain while others have a range of possible answers. An evaluation framework can hardly be complete, as is also argued in [23]. This framework includes common characteristics derived from a survey of relevant literature, case studies and own experience. However, the user can extend or modify the framework. For instance, more details of a characteristic can be explored by adding subcharacteristics with associated questions. Answers can be extended to include more options, and the weighting and criticality may be altered. Finally, characteristics can be added or removed by users. Each answer may also be accompanied by a more elaborate description of the specific issues concerning that feature of a tool. Table 16.2 shows the characteristics of the evaluation framework.

Table 16.2. Evaluation characteristics

CID *x.y*	characteristic	description/question	weight 1–5	critical Y/N
1	model specification	does the tool support specification of systems as graphical models? *{Yes/No}*	4	N
2	graphical notation for model transformation	does the tool support graphical specification of transformation? *{Yes/No}*	1	N
3	lexical notation for model transformation	does the tool support lexical specification of transformation? *{Yes/No}*	5	N
4	model-to-model transformation support	does the tool support model-to-model transformation? (e.g., from one UML model to another?) *{Yes/No}*	4	N
5	model-to-text transformation support	does the tool support model-to-text transformation, such as generation of source code? *{Yes/No}*	5	Y
6	support for model analysis	is there any support for model analysis? *{Yes/No}*	1	N
7	support for QoS management	is there any support for managing QoS during model specification and transformation? *{Yes/No}*	1	N

8	metamodel-based	is the tool based on explicit descriptions of the metamodels of source and target transformation? *{Yes/No}*	3	N
9	MOF integration	is the tool integrated with a MOF (or other metamodel-based repository)? *{Yes/No}*	4	N
10	XMI integration	is the tool integrated with XMI? *{Yes/No}* which version(s) of XMI is supported? *{list of versions}*	4	Y
11	based on UML	is the tool based on UML models as source and/or target models for transformation*?* *{Yes/No}*	2	N
12	UML specification	does the tool provide support for UML modeling *{Yes/No}*	4	N
13	UML tool integration	can the tool be integrated with existing UML tools? either directly, as active plug-ins in UML tools, or indirectly through model exchange via, e.g., XMI? *{Yes/No}or{names of the set of techniques}*	4	N
14	iterative and incremental transformation support	does the tool handle reapplication of transformation after model updates? *{Yes/No}*	3	N
15	bidirectional transformations	does the tool support bidirectional transformations? *{Yes/No}*	1	N
16	traceability	does the tool handle traceability of transformations, i.e., can it maintain traces of the source and targets of a transformation? *{Yes/No}*	4	N
17	product line variability modeling	is there support for modeling product line variability? *{Yes/No}*	4	N
18	product line variability Resolution	is there support for variability resolution? *{Yes/No}*	5	Y
19	DSM language support	is there support for defining domain-specific modeling languages (e.g., UML profiling) and DSM transformations? *{Yes (1)/DSM Transformations (0,5)/No.(0)}*	4	N
20	QoS variability	is there support for modeling and resolving QoS variability? *{Yes/No}*	3	N
21	decision process support	is there support for a decision process? *{Yes/No}*	5	N

22	maturity	what is the maturity of the tool? {Mature (0.7–1), medium(0.4–0.6), under development (0–0.3)}	2	N
23	usability	what is the usability level of the tool? is it *{Easy and intuitive (0.7–1), medium learning curve (0.4–0.6), steep learning curve (0–0.3)}*	1	N
24	availability and license	what is the license for the tool? *{Open source (1), freeware (0.4–0.9), commercial(0–0.3)}*	2	N
25	pricing	what is the pricing of the tool? *{the approximate pricing (0–0.9), N/A (1)}*	4	N

Characteristics 1–6 evaluate general support for MDD and to what extent a tool supports model specification and transformation. The *support for model analysis* characteristic will evaluate support for analysis and checking of model consistency, correctness, etc. Management of QoS during system specification and transformation is evaluated through characteristic 7. Flexibility and the extent to which the tool supports standards and enables easy integration and interoperability are the focus of characteristics 8–13. Supporting an iterative and incremental process model is evaluated through characteristics 14–16. Characteristics 17–21 are specifically tuned to supporting the specific requirements of product line engineering. General nonfunctional properties of the evaluated tool are the focus of characteristics 22–25. Many additional nonfunctional properties such as the extensive set presented in [20] may be relevant in particular cases. This framework only includes some of the important ones that will typically be considered. The user can add more nonfunctional properties if needed.

The Characteristic Identification (CID) field is used to number the characteristics for later reference. The numbering can be flat as shown in Table 16.2. The CID field can also be used to define a hierarchy of categories and characteristics. For instance, defining a category five named *Support for Product Line Techniques* would appear as shown in the table below.

5	support for product line specific techniques	
5.1	product line variability modeling	is there support for modeling product line variability? *{Yes/No}*
5.2	DSM language support	is there support for defining domain specific modeling languages (e.g., UML profiling) and DSM transformations? *{Yes (1)/DSM Transformations (0.5)/No(0)}*
5.3	product line variability resolution	is there support for variability resolution? *{Yes/No}*
5.4	decision process support	is there support for a decision process? *{Yes/No}*

This allows categories of characteristics to be summed separately. The CID field can also be used to add subcharacteristics using a similar technique. The weights and critical fields of the table are optional and are used to perform more advanced evaluations. The values assigned are used for the purpose of exemplification. The weight field is used to indicate how important a particular feature is for a particular user/domain. The weight function is used to cater for different users with various preferences and different problem categories requiring different types of support. The answers to the set of questions are normalized to a figure ranging from zero to one. For yes/no answers, *yes* can be normalized to 1 and *no* to 0. The weight may be a number from 1 to 5, and the final value of the characteristic is the product of weight and normalized value. If all features have the same importance, the weighting function is superfluous.

The *critical field* is used to indicate if a feature is critical. If the normalized answer appears to be 0 for a critical characteristic, the tool is not usable for the particular case. The evaluation framework characteristics in Table 16.2 define example instances of weights for each characteristic and set some of them to be critical [5,10,18].

In the following section, the evaluation framework is applied on a set of MDA-oriented tools.

16.4 Examples of Tool Evaluations

This section presents a selection of existing tools in the MDD/PLE area, examining their characteristics and seeing how they support the characteristics described in Sect. 16.3.2. The evaluations apply the weights for each characteristic and calculate the weighted score, which are summed up for each tool.

16.4.1 The Evaluated Tools

Since variability, domain concepts, and reference architectures can be specified in modeling languages like UML and product derivation can be viewed as a special case of model transformation [17], tools supporting MDD should in principle be able to support essential PLE tasks. Most of the relevant tools currently on the market are promoted as MDD tools. However, the evaluation framework explores the extent to which tools are able to support essential PLE tasks and to which they can be used in a model-based PLE approach.

The focus has been on evaluating a selection of tools, some of them commercial and some open-source based, which are positioned within the MDD arena and that focus on model transformation and code generation. In consequence, they should in principle support product derivation to some extent. Pure modeling tools such as traditional UML tools have not been evaluated, since we are interested in evaluating tools that provide support for the distinctive software engineering tasks that have appeared with the introduction of the MDD and PLE approaches, such as model transformation and system derivation.

The list below gives a brief overview of the tools evaluated:

- *Atlas Transformation Language* (ATL). An open-source MOF-based model transformation tool, which is part of the Eclipse GMT project (Sect. 16.4.3).
- *UML Model Transformation Tool* (UMT). An open-source UML/XMI-based tool for model transformation and code generation (Sect. 16.4.4).
- *ArcStyler*. A commercial MDA tool from Interactive Objects, which is bundled with the UML tool Magic Draw (Sect. 16.4.5).
- *XMF-Mosaic*. A commercial tool from Xactium, which provides a meta-programming environment (Sect. 16.4.6).

16.4.2 A Common Example

This chapter introduces a common example used in the evaluation of the tools – the watch example – a simple application representing a software wrist watch, described in terms of a UML-based feature model as shown in Fig. 16.2.

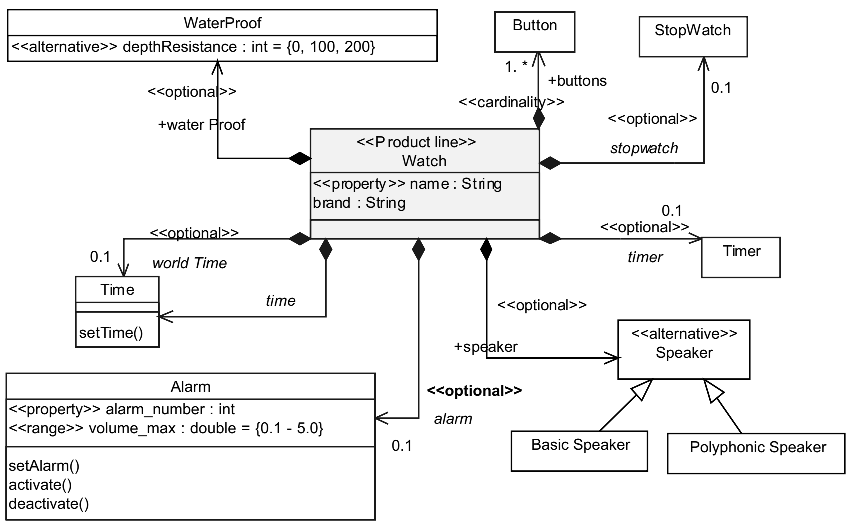


Fig. 16.2. The Watch example UML model

The Watch model represents a Watch product line (a general watch application), with a set of commonalities (such as the Time feature) and a set of variabilities (such as the Alarm and StopWatch feature). We recommend specifying a concrete domain example relevant for the particular product line, and using this actively when performing tool evaluation and selection. The watch example used here is defined in full detail in Chap. 6.

In the evaluation process, the Watch example has been used as a common basis for investigating tool characteristics. It has typically been used as an input model for testing transformation and product derivation capabilities, which has been valuable input for performing evaluation of the set of characteristics specified by the framework.

16.4.3 Atlas Transformation Language (ATL)

The Atlas Transformation Language (ATL) was developed by INRIA/University of Nantes as open source under the Eclipse (Generative Model Transformer GMT – http://www.eclipse.org/gmt) project. It is a hybrid language (a mix of declarative and imperative constructions) designed to express model-to-model transformations. ATL is similar to the QVT submission in terms of semantics, but differs in syntax. It is based on declarative rule definitions, which define mapping between source models and target models. The example below illustrates the ATL syntax in a transformation from a product line model to a product model, which could take as input, the Watch model.

```
module ProductLineDerivation;
create OUT:ProductMdl from IN:ProductLineMdl, IN2:VariabiliyMdl;

--
-- Product Line Model to Product Model rule
--
rule ProductLineMdl2ProductModel {
     from lineMdl : ProductLineMdl!Model
     to prodMdl : ProductMdl!Model
     (
            name <- lineMdl.name,
            classes <- lineMdl.modelElements
     )
}
--
-- Optional classes
--
rule ClassToClass {
     from                           lineClass :
ProductLineMdl!Class[lineClass.getVariability('Optional')
            and lineClass.variabilityIsSelected()]
     to productClass : ProductMdl!Class
     (
            name <- lineClass.name,
            description <- lineClass.description,
            attributes <- lineClass.attributes
     )
}
```

ATL provides no direct support for product line derivation. One possible way of supporting this would be to use a variability resolution metamodel as input for transformations together with the Product Line Model. The transformations could then use this combination of models to derive product models. The ATL code shown above illustrates this process. Two separate models are defined as input models; one defining the product line; the other the variability resolutions. Table 16.3 describes the characteristics of ATL.

Table 16.3. ATL characteristics

CID	characteristic	score/evaluation	weighted score
1	model specification	no. ATL cannot be used to specify models. It uses models as input for transformations and can generate new models	0
2	graphical notation for model transformation	no. ATL only provides lexical syntax for transformation	0
3	lexical notation for model transformation	yes. ATL lexical language, a declarative (hybrid) language	5
4	model-to-model transfor-mation support	yes. ATL's main functional purpose is model-to-model transformation.	4
5	model-to-text transforma-tion support	yes. Model-to-text transformation can be supported by streaming mechanisms of models to textual format.	5
6	support for model analysis	no. There is no direct support for model analysis. However, queries on models may be used to perform different analytical tasks	0
7	support for QoS manage-ment	no. There is no support for quality of service in ATL	0
8	metamodel-based	yes. ATL is based on MOF metamodels. It provides integration with several metamodel repository implementations	3
9	MOF integration	yes. ATL integrates with Netbeans Metadata Repository (MDR) and Eclipse Modeling Framework (EMF)	4
10	XMI integration	yes. ATL imports XMI files for metamodels and models, using support in underlying MOF/XMI frameworks, such as EMF	4
11	based on UML	yes. ATL supports transformation on UML models through MOF and XMI support	2
12	UML specification	no. There is no support for UML specification in ATL	0
13	UML tool integration	no. There is no direct integration with UML tools. There is indirect integration through MOF/XMI	0
14	iterative and incremental transformation support	no. There is no specific support for handling aspects such management of retransformations, reverse transformations, etc.	0
15	bidirectional transformations	no. There is no support for bidirection transformations	0
16	traceability	no. Traceability is not handled explicitly	0

17	product line variability modeling	no. There is no support for variability modeling in ATL	0
18	product line variability resolution	no. There is no support for variability resolution in ATL, but it may be supported through transformations based on input models that represent resolutions	1
19	DSM language support	the tool does not provide support for defining DSM languages. It provides support for transformations of DSM languages. E.g., transforming one DSM-based model to another DSM-based model	2
20	QoS variability	no. There is no support for variability of QoS aspects	0
21	decision process support	no. There is no support for handling a decision process. This would require human interaction during the transformation process	0
22	maturity	medium/underdevelopment	0.8
23	usability	steep learning curve	0.2
24	availability and license	open source (Eclipse Public License)	2
25	pricing	N/A	4

Summary. ATL provides a transformation language and tool that supports very general and flexible means of transforming between model abstractions defined by metamodels. It is open source, with an increasing user community, and currently under continuous development. However, it provides poor support for product line characteristics, such as the critical characteristic 18. The total weighted score using the defined weighting system is 37.

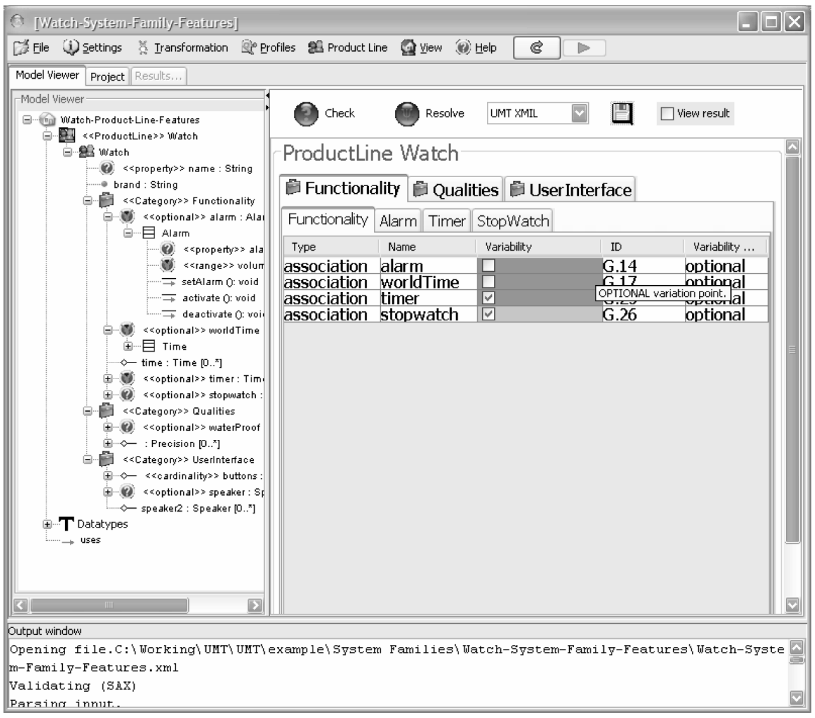
16.4.4 UML Model Transformation Tool (UMT)

UMT is an open-source tool for code generation from UML models [34,37]. It is based on reading UML models via XMI from different UML tools, such as Rational Rose, Together, ArgoUML, Poseidon, and Objecteering. Currently, it supports structural models (class) and activity models. It uses Java and XSLT as code generation/model transformation language and provides several example transformations toward EJB, WSDL, XML Schema, IDL, SQL, and more. The process of installing new transformations is quite simple.

UMT provides a graphical environment to install generators and run transformations on UML models. It uses a simplified XMI-like representation as the internal format, which is the structure used as input by transformations. There is no explicit basis in metamodels of target and source models. Transformations are thus based on ad hoc assumptions regarding input and output. It has support for a crude representation of profiles, which to some extent can be used to check model compliance. Figure 16.3 shows a snapshot of the UMT GUI after the product line model (the Watch model) has been loaded. The left field shows the model tree, with different model features and properties. The right field shows the variations and provides the user with resolution options.

In addition to code generation support, UMT supports variability resolution of UML product line models based on profiles and constraints on the source models. It provides a GUI that allows the user to resolve variabilities and generate configurations or products based on the decisions taken. Variability can be expressed within a UML model according to a simple UML profile. It supports selection of values (resolution of variability) and generation of new model configurations or concrete product models. Table 16.4 describes the characteristics of UMT.

Summary. UMT is an open-source, XMI-based tool tuned to code generation through XSLT or Java. It provides support for UML-based models, but not general MOF models. It provides support for product line variability based on a UML profile. Product line functionality is currently limited to using UML models that are according to a predefined UML profile. All the critical characteristics are supported. The total weighted score using the defined weighting system is 35.5.



Type	Name	Variability	ID	Variability ...
association	alarm	☐	G.14	optional
association	worldTime	☐	G.17	optional
association	timer	✓	[illegible]	optional
association	stopwatch	✓	G.26	optional

```
Opening file.C:\Working\UMT\UMT\example\System Families\Watch-System-Family-Features\Watch-Syste
m-Family-Features.xml
Validating (SAX)
Parsing input.
```

Fig. 16.3. UMT with variability resolution support

Table 16.4. UMT characteristics

CID	characteristic	score/evaluation	weighted score
1	model specification	no. There is no support for specifying models in UMT. It relies entirely on exported models from UML tools	0
2	graphical notation for model transformation	no. There is no graphical notation for model transformation	0
3	lexical notation for model transformation	yes. UMT uses XSLT and Java as transformation languages, with possibilities of extending to support other languages	5
4	model-to-model transformation support	no. There is no real support for model-to-model transformations. There is, however, possibility to generate "new" XMI models based on existing ones	0
5	model-to-text transformation support	yes. Model-to-text transformation is the main functional domain for UMT	5
6	support for model analysis	no. There is no support for model analysis, except for very simple support for checking of a model's conformance to simple profiles	0
7	support for QoS management	no. There is no support for management of QoS	0
8	metamodel-based	no. UMT only targets the UML metamodel and is not flexible with respect to changing this	0
9	MOF integration	no. There is no integration with MOF	0
10	XMI integration	yes. UMT imports UML/XMI files from different UML tools	4
11	based on UML	yes. UMT supports UML through XMI integration.	2
12	UML specification	no. There is no support for specifying UML models. UMT relies wholly on model input from external UML tools	0
13	UML tool integration	no. There is no direct UML tool integration. Integration is indirect through XMI	0
14	iterative and incremental transformation support	yes/no. There is lightweight support for regenerating code without overwriting previously generated and modified code	1

15	bidirectional transformations	no. There is no direct support for bidirectional transformation. However, there is some support for reverse engineering of code to XMI models	0
16	traceability	no. There is no support for traceability in UMT	0
17	product line variability modeling	no. There is no modeling support, but active support for loading UML models in which variability is specified	0
18	product line variability resolution	yes. There is support for resolution of variability specified in a UML model. This is supported for models that adhere to a product line profile, provided by a specialized tool for variability resolution.	5
19	DSM language support	the tool does not provide support for defining DSM languages. It provides support for transformations of DSM languages. E.g., transforming one DSM-based model to another DSM-based model	2
20	QoS variability	no. There is no support for QoS variability	0
21	decision process support	yes. A decision process is partly guided by the variability resolution part of the tool	4
22	maturity	medium	1
23	usability	medium learning curve	0.5
24	availability and license	open source (LGPL)	2
25	pricing	N/A	4

16.4.5 ArcStyler

ArcStyler is a commercial MDA tool bundled with the MagicDraw UML tool. ArcStyler is tuned to code generation, based on what are called MDA Cartridges, which have been developed in the MDA Cartridge Architecture – CARAT. A cartridge is essentially a specification and implementation of a transformation.

In ArcStyler, a set of predefined cartridges for common platforms is provided (e.g., J2EE, .NET). A user can also develop his own cartridges or adapt existing ones. A special model and code-based editing environment is provided for cartridge development.

Cartridges are designed partly on the basis of cartridge models, which specify the high-level structure of a cartridge in terms of artifacts and sets of artifacts. These specify which metamodel elements to work on. The details of cartridge transformations are implemented in Jython (previously JPython). Table 16.5 describes the characteristics for ArcStyler.

Table 16.5. ArcStyler characteristics

CID	characteristic	score/evaluation	weighted score
1	model specification	yes. Model specification is provided in a bundled UML environment (MagicDraw)	4
2	graphical notation for model transformation	yes. The overall structure of a cartridge is specified as a graphical model structure. The details of a transformation, however, are specified textually	1
3	lexical notation for model transformation	yes. The Jython language is used for lexical transformations	5
4	model-to-model transformation support	yes. There is some support for specifying and executing model-to-model transformations	4
5	model-to-text transformation support	yes. Generation of code is supported via the MDA Cartridges and the Jython language. This is the main functional area of ArcStyler	5
6	support for model analysis	no. There is no specific support for model analysis	0
7	support for QoS management	no. There is no specific support for QoS management	0
8	metamodel-based	yes. In some sense, ArcStyler is based on metamodels. The elements of a Cartridge use metamodel elements as input	3
9	MOF integration	no. There is no MOF integration	0
10	XMI integration	yes. The XMI capabilities provided by MagicDraw are supported	4
11	based on UML	yes. UML models from the bundled MagicDraw tool are the basis of generation	2
12	UML specification	yes, through the bundled UML tool	4
13	UML tool integration	yes. ArcStyler is bundled with MagicDraw. Integration with other UML tools is also possible through plug-ins	4
14	iterative and incremental transformation support	yes/no. Does not protect code areas in the built-in editor. Regeneration operates on the basis of commented tags. There is support for re-engineering through a Harvesting component	2

15	bidirectional transformations	no. There is no support for bidirectional transformation. However, there is support for harvesting code and regeneration	0
16	traceability	yes. Traces model elements to code using ID's in code comments	3
17	product line variability modeling	no. There is no support for variability modeling. However, this can be supported by applying a product line profile	0
18	product line variability resolution	no. There is no support for variability resolving	0
19	DSM language support	yes. Since it is bundled with MagicDraw, DSM language definitions can be specified using UML profiles	4
20	QoS variability	no. There is no support for QoS variability	0
21	decision process support	no. There is no support for a decision process in transformations	0
22	maturity	mature	1.6
23	usability	steep learning curve. Medium usability when just applying built-in cartridges. Cartridge development requires more time/has a quite steep learning curve	0.2
24	availability and license	commercial. Free "Community Architect Edition"	0.6
25	pricing	from €0 for the Community Edition to €9,800 for the full Architect Edition	0.4

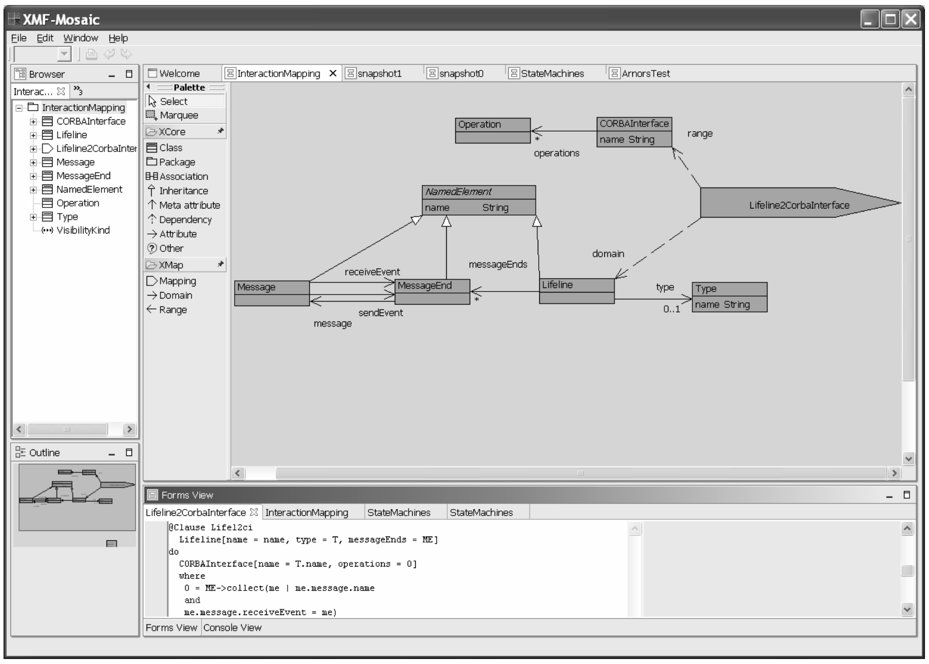
Summary. The transformation capabilities of ArcStyler are powerful with respect to structuring, definition, and reuse of transformations. However, it is not possible to define points in a transformation where user decisions can control a transformation during progress. It thus seems difficult to support product line derivation using variation elements. The evaluation reveals a lack of support of critical characteristics [18]. The total weighted score using the defined weighting system is 47.8.

16.4.6 XMF-Mosaic

XMF-Mosaic has been developed by Xactium. It is a new tool, currently available in version 1.0. XMF-Mosaic provides a metaprogramming environment, which aims to offer freedom to program and model in *any* language with full support from graphical and textual editors.

The languages and tools that come with XMF-Mosaic provide general capabilities for language modeling. The tool is currently based on MDA standards such as MOF, OCL, and QVT.

XMF-Mosaic provides a modeling interface that is typically used to define the domain language (metamodel). It may also be used to model mappings. An example is shown in Fig. 16.4, which shows the definition of a simple interaction metamodel and a mapping to Corba Interfaces (the arrow symbol in the model). The source and target are specified using domain and range associations to the anchor concepts of the source and target for the specific transformation (*Lifeline* and *CORBAInterface* in Fig. 16.4).



```
@Clause Lifel2ci
  Lifeline[name = name, type = T, messageEnds = ME]
do
  CORBAInterface[name = T.name, operations = O]
  where
   O = ME->collect(me | me.message.name
   and
   me.message.receiveEvent = me)
```

Fig. 16.4. Modeling interface

XMF-Mosaic provides support for the specification of model transformation through a language called XMap. XMap is defined using the XOCL language, a metaprogramming language for constructing languages and environments. It provides facilities for inspecting and controlling its own behavior and is the key technical feature that allows XMF-Mosaic to support tool development. The language is an imperative extension of OCL.

XMap is aligned with OMG's QVT language. An example of XOCL XMap syntax is as follows:

```
@Clause Lifel2ci
 Lifeline[name = name, type = T, messageEnds = ME]
do
 CORBAInterface[name = T.name, operations = O]
 where
  O = ME->collect(me | me.message.name
  and
  me.message.receiveEvent = me)
end
```

Since XMF-Mosaic is a framework with support for defining languages and environments and for building tools, and almost every technical criteria of our evaluation framework may be supported. *It just has to be built first.* However, the current version provides basic tools that support modeling and model transformations. The following evaluation is partly based on the provided tools and partly on the fact that characteristics may be developed as extensions. Table 16.6 describes the characteristics of XMF-Mosaic.

Table 16.6. XMF-Mosaic characteristics

CID	characteristic	score/evaluation	weighted score
1	model specification	yes. The tool supports specification of systems as graphical models by providing a subset of UML diagrams and notation	4
2	graphical notation for model transformation	yes. The downloadable version comes with limited graphical notation, which is combined with lexical notation (XMap) to make the specification complete	1
3	lexical notation for model transformation	yes. Lexical notation for model transformation is provided through XMap	5
4	model-to-model transformation support	yes	4
5	model-to-text transformation support	yes	5
6	support for model analysis	yes. Validity of models can be checked (i.e., whether they are according to their metamodel), both through an editor console and by building snapshots using the modeling interface. XWalk is an extension to XOCL, which provides facilities for efficiently running over large XCore object structures and evaluating their properties, for example running constraints or modifying data	1
7	support for QoS management	no. There is no explicit support for QoS management. However a QoS profile may be defined and used to specify QoS. These QoS profile concepts may also be used to derive QoS-aware transformation specifications	0
8	metamodel based	yes. It is based on XMF XCore, which is a MOF-like metakernel	3
9	MOF integration	yes	4

10	XMI integration	yes. XMF provides facilities for parsing and generating XML documents. High-level grammatical rules can be written, which state how a specific XML element pattern can be mapped to an XCore element or trigger the invocation an XOCL action. These rules can be used to generate a parser for a specific XML syntax	4
11	based on UML	yes. There is support for UML. It may support arbitrary modeling languages defined using XOCL. The downloadable version provides UML syntax	2
12	UML specification	yes. A subset of UML diagrams and notation is provided	2
13	UML tool integration	no. May use XMI. XMF-Mosaic supports sophisticated input and output facilities, which enable data to be streamed to and from files or other tools in a variety of different data formats	0
14	iterative and incremental transformation support	no. Process support, configuration management, etc. are not part of the XMF-Mosaic framework. XMF-Mosaic comes with the XSync language, which provides a high-level way of synchronizing data, where changes in one element cause changes to be automatically propagated to other elements	0
15	bidirectional transformations	yes. Languages for specifying bidirectional transformations may be defined using XOCL	1
16	traceability	no	0
17	product line variability modeling	yes. A product line variability modeling language may be defined	4
18	product line variability resolution	yes. Product line variability resolution mappings may be defined	5
19	DSM language support	yes. The tool provides support for defining DSM language through its metaprogramming environment and performs transformations based on these language definitions	4
20	QoS variability	no. There is no explicit support for QoS variability, but resolving functional types of QoS such as security and transaction control will be similar to defining and resolving functional variability	0
21	decision process support	no. There is no explicit support for a decision process	0

22	maturity	mature. XMF-Mosaic v1 was released in 2005	1.4
23	usability	medium learning curve	0.5
24	availability and license	commercial, free evaluation version	0.6
25	pricing	according to the web page XMF-Mosaic is competitively priced and includes 12 months' support and maintenance as standard. Discounts are available for bulk purchases and with consultancy-related packages. A significantly discounted noncommercial license (for students and academic departments) is also available	1.2

Summary. XMF-Mosaic is a very flexible tool, due to its meta-architecture providing functionality for defining relevant metamodels of the actual product line. This flexibility can appear as a problem as it lays the burden of defining metamodels on the user. However, some common metamodels and features come with the tool. Due to its flexibility, the tool can be configured to support many of the MDD and PLE tasks. The tool is model oriented, and both metamodels and transformations may be specified using models. The total weighted score using the defined weight system is 52.7.

16.5 Evaluation of the Framework

This section evaluates the work done, by analyzing the evaluation framework, the evaluated tools, and the applicability of the results. Then it compares the results with related works.

16.5.1 The Tool Evaluation Framework

The evaluation framework was derived from characteristics discussed in Sect. 16.3. The evaluation criteria are tuned to model-driven development tools in general with a specific focus on model transformation. The tool also includes important requirements for product line engineering, which are essential for supporting PLE in a model-driven context.

The resulting criteria are a mix of technical and practical aspects, which can act as a guide for selecting appropriate tools. The criteria alone do not allow for an easy comparison. In order to achieve this, the weight and critical properties must be defined and used in the evaluation. It is not fruitful to predefine these properties, since they will always be relative to specific domain needs. A set of domain experts should therefore define these prior to an evaluation.

16.5.2 The Tools Evaluated

The example evaluation is included to illustrate how the evaluation framework can be used. A set of state-of-the-art and advanced tools for model-driven development, both open source and commercial, are evaluated. The particular tools were included on the basis of their positioning as MDD tools, with a tuning to model transformation and code generation aspects. However, other tools could as well have been chosen. As part of the work, several additional tools were evaluated. These were mostly dedicated MDD tools, most of them lacking support for PLE, but providing different aspects of MDD functionality. The ones evaluated here were selected on the basis of their maturity and relevance as open source or commercial tools. Among the tools evaluated but not included in this chapter were the open source tools *MTL Engine*, *ModFact,* and *AndroMDA*, and the commercial tools *OptimalJ, Codagen Architect,* and *IQGen*.

This study has not included evaluations of dedicated UML tools. To a large extent, these also provide many aspects of MDD functionality, such as modeling and code generation. Traditionally, there has been little support for model transformation in this category of tools, and no direct support for PLE characteristics. At this time, however, we observe a growing degree of support for model transformation frameworks and even QVT in commercial UML tools. Examples are the latest Borland Together product, which implements the QVT specification, and the IBM Rational Software Architect (RSA), which implements a proprietary model transformation framework. Using built-in extension mechanisms in these tools, some support for PLE characteristics may be provided.

The evaluated Xactium tool is a representative of a DSM tool. This category of tools is characterized by their ability to support specification of domain specific languages. The language definition is then used to specify appropriate transformation specifications. In a PLE setting this is appealing, since specifying domain specific languages is an efficient mechanism for scoping product lines. Examples of other tools in this category are [24, 25].

The *V-Manage* tool suite from European Software Institute (ESI) has been described in Chap. 6. It provides an environment for defining and resolving variation models, and relating this to implementation of specific components. This tool has been excluded primarily because it is an in-house product not available to external purchasers.

16.5.3 Applicability of Results

The evaluation framework provides a baseline that can be used to evaluate and compare tools in order to make decisions when acquiring tools for model-driven product line engineering.

As shown above, the framework can be applied using selection guidelines and weights based on user requirements, which would leverage it for practical applications. It can also be integrated with existing case tool evaluation frameworks [20,23] for more holistic purposes. The evaluation examples show how different tools can be evaluated using the assigned weights. The resulting evaluation sum for a tool can be used to guide the final tool selection. A clear specification of the characteristics and the weighting is the key to a good evaluation.

This framework can be used in tool selection processes for model-driven product line tools, and will give the users a baseline, which can be modified based on their specific selection of characteristics. Such a selection would be more easily achieved if the framework characteristics have assigned weights and criticality.

16.5.4 Related Work

The ISO 14102 standard, guideline for the evaluation and selection of CASE tools [20], proposes a general standard for evaluation and selection. It defines a broad hierarchy of characteristics used to evaluate and select case tools in general. As pointed out in [23], there is a coverage problem with this standard; in any given case, it is not likely that the standard will cover all relevant characteristics; at the same time, it will probably include irrelevant characteristics.

This framework has a smaller scope and focuses only on evaluations of MDD- and PLE-type case tools. In line with experiences presented in [23], this framework is less extensive than that of ISO 14102, but it includes characteristics not listed there. Reference [23] also argues that the hierarchy presented in ISO 14102 can be a problem, since there is an agreed characteristic hierarchy, while most cases will need to deviate from this hierarchy. This framework provides a flat structure that can be defined as a hierarchy by the user. This is done by means of the identification number for categorization. (For example, the identification numbers of characteristics in category 1 is numbered 1[.*x*]*, where *x* is a subnumber and [.*x*]* implies zero or more subnumbers in order to build a multilevel hierarchy.) Other standards in the area such as [19,21] have similar problems to those of the ISO 14102 standard.

This framework can be seen as a specialization of ISO 14102, in which the domain of tools has been narrowed. Moreover, when using this framework, the evaluation and selection process as described in ISO 14102 can be used. ISO 14102 defines four major processes: Initiation Process, Structuring Process, Evaluation Process, and Selection Process.

In [13], the Gartner group suggests a list of recommendations when evaluating and selecting tools, including (1) do not worship one “hot” technology, (2) do not select tools before institutionalizing an application architecture and infrastructure, (3) do not acquire tools without an analysis/design tools acquisition strategy, (4) do not acquire too many or too few tools, (5) do not make deliberate trade-offs between application portability and optimization per platform, (6) always consider return on investment (ROI) and time-to-payback of analysis and design technologies, but extend the ROI model through end-user costs/benefits, (7) always try to select stable vendors with durable technology, (8) institute a modern, iterative methodology for analysis and design.

These characteristics are generally valid when evaluating and selecting many kinds of tools and are somewhat orthogonal and supplementary to guidelines like ISO 14102 and the framework presented here. One of the criteria (7), however, is in conflict with selecting open source technology, which is not always good advice. As this evaluation shows, open-source providers may provide software that supports pieces of model-driven product line processes, which may not even be supported by commercial tools.

16.6. Conclusions and Future Research

This chapter has offered an overview of model-driven development and product lines and has looked at how they can be integrated. We have described a framework, based on tool characteristics that can be used to evaluate and compare the suitability of MDD and PLE tools. We have also described a set of tools, which we have used as examples for evaluation, and applied the framework to these in specific evaluations.

When considering MDD and product lines in light of existing tools, it is clear that few tools available today provide specific functionality capable of supporting product line and MDD concepts out of the box. This is primarily due to lack of acknowledgment of the need for product line support from traditional MDD tool providers. Looking at the assessment of the range of tools used as input for this chapter, some tendencies can be seen: A growing number of tools support model-driven development in both modeling and transformation. Generally speaking, few of these specifically address PLE at present. However, the inherent flexibility of many tools permits extensions that may address this to be built. Looking ahead, we can expect more stability and more possibilities of providing such extensions. The increasing attention to domain-specific modeling (DSM) languages in the MDD area, e.g., [14,24,38] is promising seen from the PLE perspective. Defining domain-specific modeling languages can for instance be used to scope product lines and provide more efficient support for modeling domain specific concepts.

Product line engineering is currently the subject of much attention, as documented for instance in [5,9,35]. In [14], which describes the Microsoft *Software Factory* concept, PLE is predicted to be an important part of modern software engineering. This is confirmed by recent provisions in Microsoft's Visual Studio tool suite, such as the domain-specific language tools and the spec# language [24].

Our experience from projects such as COMBINE [7] and MODELWARE [24] is that well-defined scoping is essential for success with MDD. Using product line engineering techniques to provide proper scoping seems appropriate. For this reason, we believe that PLE techniques and mechanisms will be incorporated in future MDD tools. Initially, this will happen through suitable configuration and scoping mechanisms, then through the provision of product line-reusable assets and variability management. Support for more interactive transformation processes is also needed both for pure MDD [15], and in model-driven product line engineering approaches.

The market and focus for tools supporting different aspects of MDD are steadily growing, and the quality and functionality of such tools are improving. Influencing or initiating standards, e.g., for variability modeling, will improve the chances of achieving more tool support for PLE, through both open source and commercial tools.

The evaluation framework presented here provides a baseline for evaluating MDD and PLE tools. It can be extended or supplemented, for example with characteristics defined in ISO 14102 and tailored to the need of the specific domain, and as such would be applied to future tools.

Acknowledgments

We gratefully acknowledge the extensive reviews of Juan Carlos Dueñas, Alessandro Fantechi, Timo Käkölä, Janne Luoma, Juha-Pekka Tolvanen, and Tewfik Ziadi, which significantly improved the quality of this chapter.

References

1. Atkinson, C., Bayer, J., Bunse, C., Kamsties, E., Laitenberger, O., Lagua, R., Muthig, D., Peach, B., Wust, J., Zettel, J.: *Component-based Product Line Engineering with UML* (Kobra) (Addison-Wesley, Reading, MA 2001) ISBN 0-201-73791-4. http://www.iese.fhg.de/Kobra_Method/
2. Atlas Transformation Language (ATL) homepage. http://www.sciences.univ-nantes.fr/lina/atl. Cited 24 Nov 2005
3. Becker, M.: Towards a general model of variability in product families. In: Software Variability Management Workshop (SVM 2003). 25th International Conference on Software Engineering (ICSE, 2003)
4. Blechar, M.J., Driver, M.: Predicts 2004: MDSFs Offset J2EE Complexity, Gartner report, ID Number: SPA-21-5432
5. Bosch, J.: *Design & Use of Software Architectures – Adopting and Evolving a Product-Line Approach* (Addison-Wesley, Reading, MA 2000) ISBN 0-201-67494-7
6. Clauß, M.: Generic modeling using UML extensions for variability. In: Workshop Domain Specific Visual Languages, OOPSLA, USA, October 2001
7. COMponent-Based INteroperable Enterprise system development (COMBINE), ESPRIT V IST project no. 20893. http://www.opengroup.org/combine/. Cited 16 Nov 2005
8. Czarnecki, K., Helsen, S.: Classification of model transformation approaches. In: 2nd Workshop on Generative Techniques in the Context of Model-Driven Architecture, Conference on Object-Oriented Programming, Systems, Languages, and Applications 2003 (OOPSLA'03)
9. Duggan, J., Vecchio, D., Plummer, D.C., Driver, M., Natis Y.V., Hotle, M., Feiman, J., James, G.A., Sinur, J., Pezzini, M., Light, M., Blechar, M.J., Valdes, R., Lanowitz, T.: Hype Cycle for Application Development, 25 June 2004, Gartner report, ID Number: G00120914
10. Estublier, J., Vega, G., Ionita, A.D.: Composing domain-specific languages for wide-scope software engineering. In: MoDELS 2005 Conference, ed by Briand, L., Williams, C., ISBN3-540-29010-9, pp 69–83
11. FAct-based Maturity through Institutionalisation Lessons-learned and Involved Exploration of System-family engineering (FAMILIES), ITEA project ip02009, Eureka ∑!2023. http://www.esi.es/en/Projects/Families/. Cited 16 Nov 2005
12. Gardner, T., Griffin, C., Koehler, J., Hauser, R.: A review of OMG MOF 2.0 Query/Views/Transformations Submissions and Recommendations towards the final Standard, (MetaModeling for MDA Workshop Nov 2003. York, UK)
13. Gartner Group: Application Development Management – Enterprise Applications Development Tools – Evaluation and Selection, Strategic analysis report, Gartner Group, Sept 1996
14. Greenfield, J., Short, K., Cook, S., Kent, S., Crupi, J.: *Software Factories, Assembling Applications with Patterns, Models, Frameworks and Tools* (Wiley, New York 2004) ISBN 0-471-20284-3
15. Grønmo, R., Aagedal, J., Solberg, A., Belaunde, M., Rosenthal, P., Faugere, M., Ritter, T., Born, M.: Evaluation of the QVT Merge Language Proposal, MODELWARE project report, SINTEF report number STF90 A05046, ISBN 82-14-03659-3, OMG document ad/2005-03-05. http://www.omg.org/cgi-bin/doc?ad/05-03-05
16. Grønmo, R., Oldevik, J.: An empirical study of the UML Model Transformation Tool (UMT). In: The 1st International Conference on Interoperability of Enterprise Software and Applications (INTEROP-ESA), Geneva, Switzerland, Feb 2005
17. Haugen, Ø., Møller-Pedersen, B., Oldevik, J., Solberg, A.: An MDA-based framework for model-driven product derivation. In: The 8th IASTED International Conference on Software Engineering and Applications, ed by Hamza, M.H. (ACTA, Nov 2004) pp 709–714
18. International Standards Organization (ISO): ISO/IEC 10746-1:1998, Information technology – open distributed processing – reference model: overview (ISO RM-ODP), ISO/IEC 10746-1:1998 (ISO standard, 1998)
19. International Standards Organization (ISO): ISO/IEC 12119:1994, Information technology – software packages – quality requirements and testing (ISO Standard 1994)

20. International Standards Organisation (ISO): ISO 14102:1995, Information technology, guideline for the evaluation and selection of CASE tools, JTC 1/SC 7 (ISO Standard 1995)
21. International Standards Organization (ISO): ISO/IEC 25000:2005, Software engineering – software product quality requirements and evaluation (SQuaRE) (ISO Standard, 2005)
22. Jacobson, I., Booch, G., Rumbaugh, J.: *The Unified Software Development Process* (Addison-Wesley, Reading, MA 1999) ISBN 0-201-57169-2
23. Lundella, B., Lings, B.: Comments on ISO 14102: the standard for CASE-tool evaluation. Comput. Standards Interf. **24**(5), 381–382 (November 2002)
24. MetaCase Whitepaper: ABC to MetaCase Technoology. http://www.metacase.com/, © 2004 by MetaCase. Cited 26 Nov 2005
25. Microsoft Corporation: Visual Studio 2005 Team System Modeling Strategy and Faq, In: MSDN Library. http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnvs05/html/vstsmodel.asp. Cited 16 Nov 2005
26. MODELing Solutions for softWARE systems (MODELWARE), ESPRIT VI IST project no. 511731. http://www.modelware-ist.org. Cited 16 Nov 2005
27. Object Management Group (OMG): Meta Object Facility 2.0 (MOF), Meta Object Facility (MOF) 2.0 Core Specification, OMG document ptc/03-10-04. http://www.omg.org/cgi-bin/apps/doc?ptc/03-10-04.pdf. Cited 16 Nov 2005
28. Object Management Group (OMG): Meta Object Facility 2.0 XMI Mapping Specification, OMG document ptc/04-06-11. http://www.omg.org/cgi-bin/apps/doc?ptc/04-06-11.pdf . Cited 16 Nov 2005
29. Object Management Group (OMG): MOF model to text transformation language request for proposal, OMG document: ad/2004-04-07. http://www.omg.org/cgi-bin/doc?ad/04-04-07. Cited 16 Nov 2005
30. Object Management Group (OMG): MOF Query/Views/Transformations RFP, OMG document: ad/2002-04-10. http://www.omg.org/cgi-bin/doc?ad/02-04-10. Cited 16 Nov 2005
31. Object Management Group (OMG): OMG MDA Guide v1.0.1, OMG document omg/2003-06-01. http://www.omg.org/docs/omg/03-06-01.pdf. Cited 16 Nov 2005
32. Object Management Group (OMG): MOF QVT Final Adopted Specification, OMG Adopted Specification, OMG document number ptc/05-11-01 http://www.omg.org/cgi-bin/doc?ptc/05-11-01. Cited 9 April 2006
33. Object Management Group (OMG): Unified Modeling Language 2.0 (UML 2.0), UML 2.0 infrastructure final adopted specification. http://www.omg.org/cgi-bin/apps/doc?ptc/03-09-15.pdf. Cited 16 Nov 2005
34. Oldevik, J., Model transformation for system families prototype, FAMILIES consortium-wide deliverable, CWD4.3:2.3 version 1.0. http://www.esi.es/Families/. Cited 16 Nov 2005
35. Pohl, K., Böckle, G., van der Linden, F.: Software product line engineering – foundations, principles, and techniques (Springer, Berlin Heidelberg New York 2005) ISBN 3-540-24372-0
36. Solberg, A., Oldevik, J., Jensvoll, A.: A generic framework for defining domain-specific models. In: *UML and the Unified Process*, ed by Favre, L. (IRM, Hershey, 2003) pp 23–38
37. UML Model Transformation Tool (UMT). http://umt-qvt.sourceforge.net/. Cited 16 Nov 2005
38. Xactium Limited: Language Driven Development and XMF-Mosaic, Whitepaper. http://www.xactium.com (2005). Cited 24 Nov 2005
39. Ziadi, T., Hélouët, L., Jézéquel, J.M.: Towards a UML profile for software product lines. In: *Software Product-Family Engineering*, ed by van der Linden, F., 5th International Workshop, PFE 2003, Italy, Nov 2003. Lecture Notes in Computer Science, vol 3014 (Springer, Berlin Heidelberg, New York 2003) pp 129–139