DSL Classification

Benoît Langlois[1], Consuela-Elena Jitia[2], Eric Jouenne[2]

[1]Thales D3S/EPM, Domaine de Corbeville, 91404 Orsay Cedex, France
benoit.langlois@thalesgroup.com
[2]Thales Research & Technology, RD 128, 91767 Palaiseau Cedex
{consuela.jitia | eric.jouenne}@thalesgroup.com

Abstract
In model-driven engineering, a Domain-Specific Language (DSL) is a specialized language, which, combined to a transformation function, serves to raise the abstraction level of software and ease software development. However, in practice, beyond this general definition, DSLs adopt multiple forms of representation and implementation. Actually, the issue is projects, and mainly large-scale projects, have to deal with DSL and DSL tool variants. The purpose of this paper is to propose a DSL feature model in order to identify DSL and DSL tool variability.

Keywords
DSSD, MDA, MDD, DSL, DSL TOOL, FEATURE MODEL, PRODUCT LINE.

1. Introduction

A DSL is a specialized and problem-oriented language [4]. Contrarily to a General Purpose Language (GPL) (e.g., UML, Java or C#), a DSL serves to accurately describe a domain of knowledge. The interest to combine a DSL and a transformation function is to raise the abstraction level of software. A DSL user concentrates her/his efforts on domain description while complexity, design and implementation decisions and details are hidden. The stake is to improve productivity and software quality.

However, what is the next consensus beyond this general definition? An experience consists in starting the development of a DSL editor coupled to a generator. Quickly, the issue of the variants of DSL editors and generators emerges. Regarding the language, is it a tree-based DSL or a set of data without real structure? Is it a graphical or textual notation? Is it a declarative or imperative style? Regarding the transformation, what are the used transformation techniques? What is the update strategy of the produced artifacts: destructive or incremental? Regarding the process, what is the level of assistance of the DSL editor? Actually, those differences are normal because it depends on the type of software development projects have to deal with and available tools. But all these differences make difficult to share a general agreement to answer the question. The purpose of this paper is to show the links between DSL variants. For this, this paper proposes a DSL feature model for expressing variations on DSLs and DSL tools.

This paper is organized as follows. Section 2 proposes a DSL feature model. Section 3 crosses this feature model with three DSL tools, GMF (Graphical Modeling Framework), Microsoft DSL Tools, and Xactium's XMF-Mosaic. Section 4 presents further work, and Section 5 concludes.

2. DSL Feature Model

This section presents a DSL feature model, which covers languages, transformation, tooling, and process aspects of DSLs. The feature model is expressed with the FODA notation presented in [4].

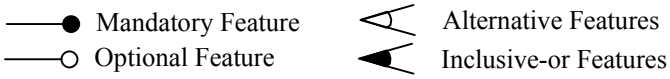

Figure 1. Model Feature Notation

At the root level, language and transformation are mandatory features because they are parts of the DSL definition. Tool is also mandatory because it serves to automate transformation from a domain, the problem space, down to lower abstraction levels, the solution space. Process is optional because it can be undefined or implicit.

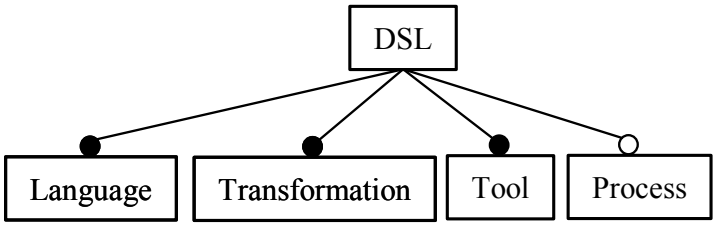

Figure 2. Root of the DSL Feature Model

2.1 Language Features

The language of a DSL is formalized by an Abstract Syntax (AS) and one or several Concrete Syntaxes (CS).

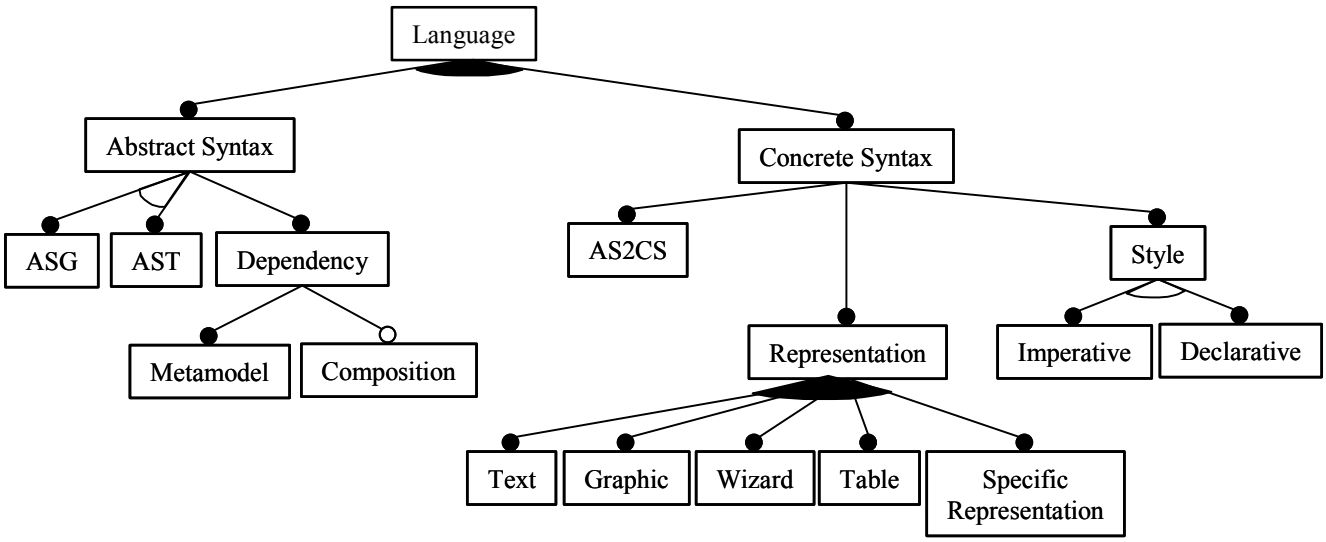

Figure 3. Language Features

ABSTRACT SYNTAX. An AS characterizes elements of a domain and their relationships without implementation consideration. An AS can be expressed either by a context-free grammar (a set of productions of the form $A_0 \rightarrow A_1 \ldots A_n$, where $A_1 \ldots A_n$ is terminal or not) or

with a meta-model. A meta-model conforms to a meta-meta-model (e.g., OMG's MOF [20] meta-meta-model). The meta-model, also named DSL meta-model in the sequel, formalizes the domain language. For reusability, a DSL meta-model can be the composition of several DSL meta-models. For instance, a set of DSLs reuse a common constraint language. Transformation, from the problem to solution space, is defined at the DSL meta-model level and applied at the model level. Transformation is covered by the feature model in the transformation part. An AS is either represented by an Abstract Syntax Graph (**ASG**) or an Abstract Syntax Tree (**AST**). An AST is a hierarchical representation of non-terminal and terminal elements. For instance, for a simple calculator AST, non-terminal elements are operators and leaves are values. ASTs, powerful for languages with a strict composition relationship, cannot support all relationships, such as a relationship between an attribute and its type (this relationship does not respect a tree-structure). In this case, language elements and their relationships create a graph.

CONCRETE SYNTAX. A Concrete Syntax is a specific representation of a DSL in a human-usable form. An **AS-to-CS mapping** defines the correspondence between an AS and a CS. A CS has a type of **representation** (e.g., text, graphic, wizard, table, matrix), and a **style** (i.e., declarative or imperative). The AS-to-CS mapping identifies mapped elements and associated element types of representation, such as a text field for wizards. Bidirectional, this mapping is used during parsing operations to realize the CS-to-AS translation. TCS [23] and *Xtext* [27] are examples of tools parsing textual DSLs.

By convention, in the sequel, DSL description means information expressed at the CS level, and domain data, information at the AS level.

2.2 Transformation Features

AS captures the problem space properties. The transformation, which ensures the correspondence from the problem to the solution, takes into account the problem-to-solution element mapping, and all design, implementation, platform and architecture decisions. Transformation has to answer to three questions: How to specify transformation? What are the assets expected from the transformation? How to realize the transformation to produce the expected assets?

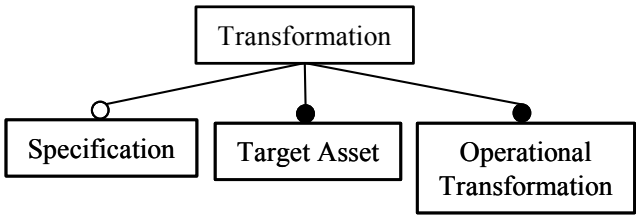

Figure 4. Root of the Transformation Features

SPECIFICATION. A specification transcribes emerging transformation criteria. They can be described in two different ways. A **problem-to-solution mapping** describes the correspondence between problem and solution elements. This mapping can be formalized by a transformation model. A set of **viewpoints** [7][8] on transformation describe design, implementation, platform descriptions, architecture decisions, and non-functional constraints, and allow having an exhaustive and complete mapping. Examples of viewpoints: persistence mapping, deployment, architectural patterns. All these specification features are optional

because mapping and viewpoints can be implicit and immersed inside transformation generators / interpreters, frameworks, or inside COTS (Component Off the Shelf).

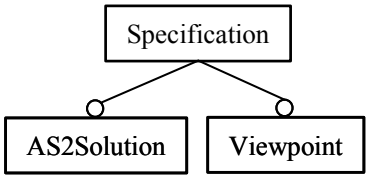

Figure 5. Specification Features

TARGET ASSET. A target asset (TA) is a software artifact, result of the transformation. It has a **representation** form (e.g., model, text (documentation, configuration files…), graphic, executable code). This type list is not limited due to the variety of possible software artifacts. In a DSL-to-DSL transformation process, a TA can be a DSL. In this case, the TA respects a CS, and representations of a CS and of the TA match.

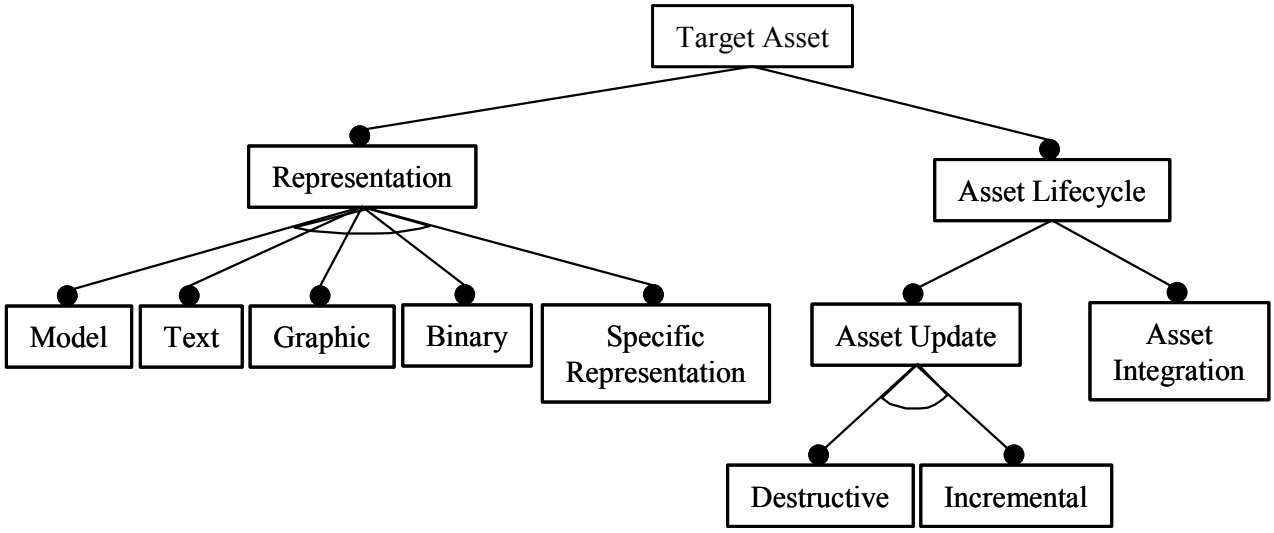

Figure 6. Target Asset Features

From the user viewpoint, successive transformations can be applied on a DSL description. This implies a TA has a **lifecycle**. The first time, it is created, next updated, and possibly deleted. An update is **destructive**, when a TA is destroyed and created at each generation, or **incremental**, when there is a synchronized update between source and target elements. Incremental update implies traceability and/or merging management. Asset lifecycle covers also asset **integration**, which consists in assembling and packaging assets for delivery. Assets to be integrated are produced by the DSL transformation or not (e.g., when assets are manually created).

OPERATIONAL TRANSFORMATION. This part covers features related to the application of a DSL transformation. Are distinguished technique, execution, scheduling, and variability aspects.

The **transformation technique** addressed all techniques realizing an AS to TA transformation. Transformation techniques around model and text (which are only a set of techniques among all transformation techniques) are model-to-model [M2M] (e.g., ATL [2]),

model-to-text [M2T] (e.g., JET [11], TCS [23], Xpand [13]), text-to-text [T2T] (e.g., PERL), and text-to-model [T2M] (e.g., TCS [23], Xtext [27]) transformations. Code production from M2T/T2T transformations can be **compiled** to produce byte/machine code. An alternative is to produce directly TA from a DSL description with an **interpreter** (e.g., MetaEdit+ [17]). Transformation can be reduced to one transformation or can be a transformation chain. On the other hand, a DSL TA can be injected in a new DSL transformation process. In final, this transformation sequence meets the need of **scheduling**. Scheduling can be implicit, when it is undefined or not automated, or it can be explicit. When explicit and automated, scheduling is either described in the development environment of the transformation or externally, for instance with scripts (e.g., Ant [1]).

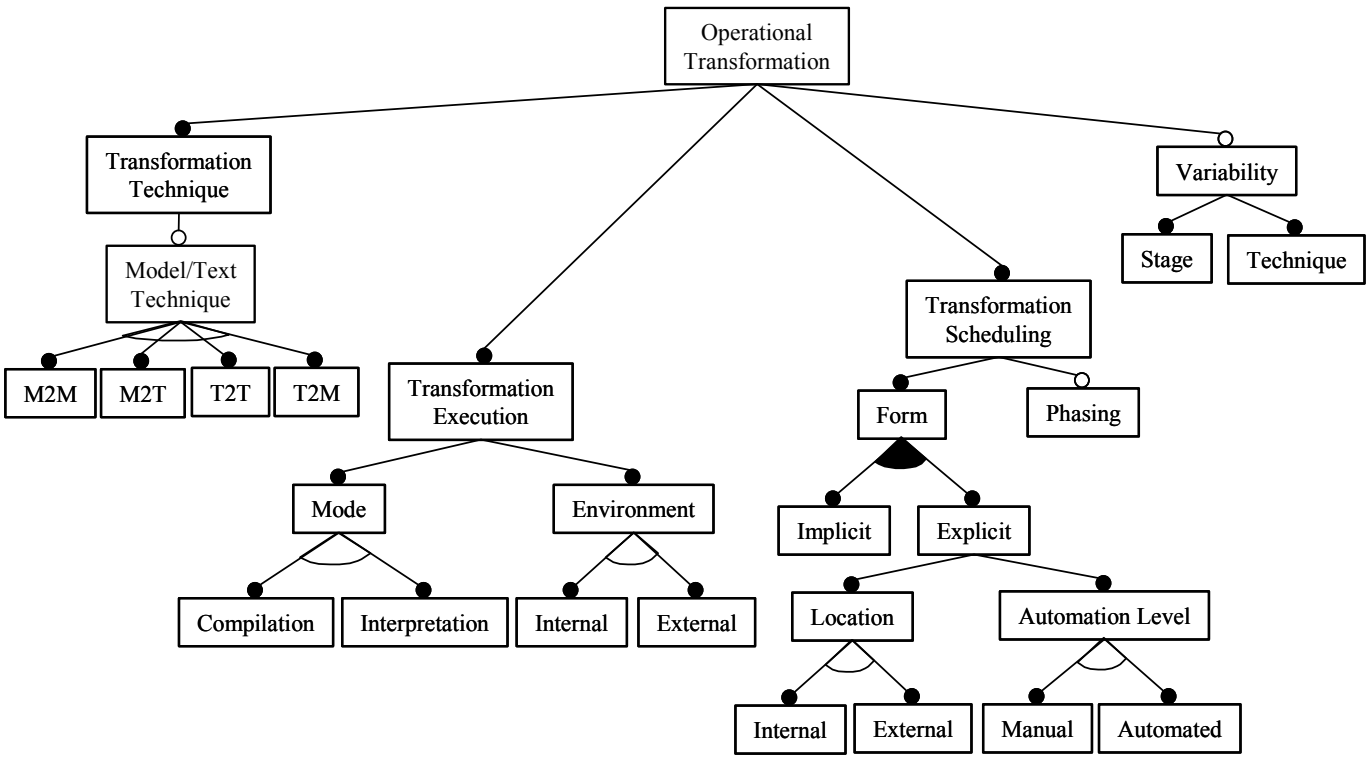

Figure 7. Operational Transformation Features

Regarding the **transformation execution**, all the transformations can be applied in the same development environment or externally in different environments. Languages such as Lisp-like languages, Smalltalk, or script languages (e.g., Ruby) have the ability to execute transformations in the same environment and hide internal DSLs, parsing and transformation chains. External execution is necessary when nature of transformations and execution platforms are different (e.g., for generation of database scripts from a modeler, or when the development process implies a separation of generation and execution stages), or when the platform does not support generation and execution in the same execution platform instance (e.g., Eclipse).

Variability on transformation, in order to obtain problem-to-solution variants, can be realized at different stages: during generation / interpretation, compilation, integration with other TAs, frameworks and tools, during deployment or execution. Several techniques can be used, such as AOSD (Aspect-Oriented Software Development) [22], with template evaluation, by extensions (e.g., Eclipse plug-ins), or instance creation with factories at runtime.

2.3 DSL Tool Features

This part stresses on DSL tools (e.g., MetaEdit+ [17], Microsoft DSL Tools [18], or XMF-Mosaic [26]).

RESPECT OF ABSTRACTION. The purpose of abstraction is to reduce software description. There are two levels of maturity depending on the ability to encapsulate transformation. The first level is **intrusive** transformation when the user has to be aware of internal transformation details. For instance, the user has to add pieces of code for specific behaviors or she/he has to decorate domain data with technical markers [19]. This intrusion is the sign of non-mature tools or of not properly designed DSLs – the expression level of the problem is too low. The second level is **seamless** transformation when transformation and solution details are completely encapsulated.

ASSISTANCE. Tool assistance aims at guiding the DSL tool user during definition and transformation of domain data. Assistance is **adaptive** when assistance changes in function of the context of usage. For instance, help documentation is **static** because it never changes; on the contrary, a DSL tool with DSL meta-model knowledge is able, by introspection, to complete or prevent actions contextually. The second kind of assistance is the **process guidance**. The DSL tool is able to guide the user at a process step or at the process workflow level. The third assistance is DSL **checking**. Checking is mandatory in the feature model because a DSL tool must ensure consistency and completeness of domain data. DSL checking can be realized on the fly or on user action.

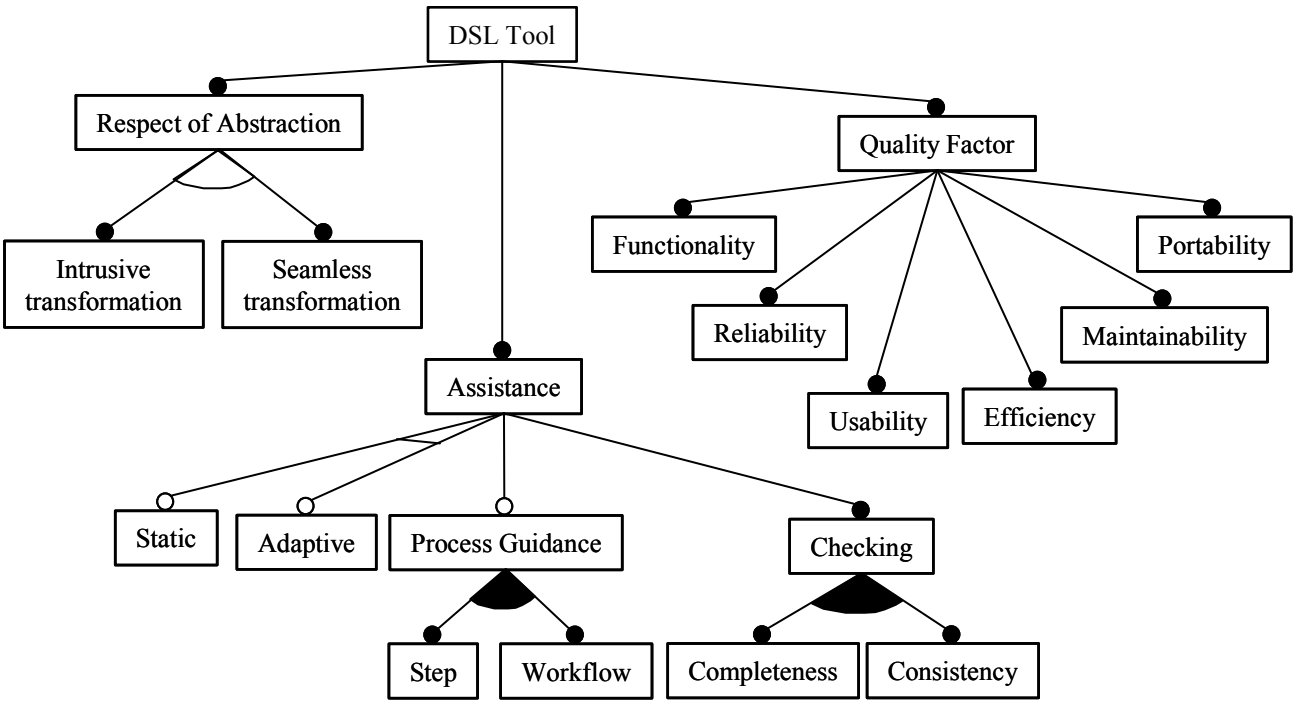

Figure 8. DSL Tool Features

QUALITY FACTOR. Quality factors covers non-functional aspects of DSL Tool. [9] organizes these factors into six categories: functionality, reliability, usability, efficiency, maintainability, and portability. **Functionality**, which includes suitability, accuracy, interoperability, security and functionality compliance factors, is the capability of a DSL tool to provide expected functions when it is solicited. For instance, edition or transformation functions are able to manage incomplete or not well-formed DSL descriptions. **Usability** is

the capability of the DSL tool to be understood, learned, used and attractive to the user. This quality is crucial for a user to accept a new DSL tool and to change is a way to work. **Efficiency** is the capability of the DSL tool to provide appropriate performance, relative of the amount of resources used. For instance, complex transformation and large models decrease performance. **Maintainability** is the capability of the DSL tool to be modified. This includes corrections, improvements, adaptation to the environment. A difficulty is for instance to keep up the same transformation results when generators or generator algorithms change. **Portability** is the capability of the DSL tool to be transferred from one environment to another. Tooling and DSL descriptions are a long-time investment for companies. The concern is to have platform-independent, and then durable, DSL tool development and DSL descriptions.

2.4 DSL Process Features

A DSL process defines how development projects with DSL must be executed. This part of the feature model addresses the Domain-Specific Software Development (DSSD) [15], i.e., the software engineering which focuses on DSL activities and practices. DSSD reuses a lot of practices from Model-Driven Software (MDSD) [22]. For instance, transformation is a MDSD topic and connections with other software disciplines, such as software product-line [3][4] or aspect-oriented programming, are common.

This feature model covers work definition, role, and guidance aspects (refer to SPEM terms [21]). For comprehension, an illustration could be: i) A DSL process is broken down into three kinds of activities: DSL definition, transformation, design with DSL; ii) the two first activities are allocated to a DSL designer and the last one to a final DSL user; iii) best practices help the designer to develop transformations and a tool mentor explains the user how to use a DSL tool.

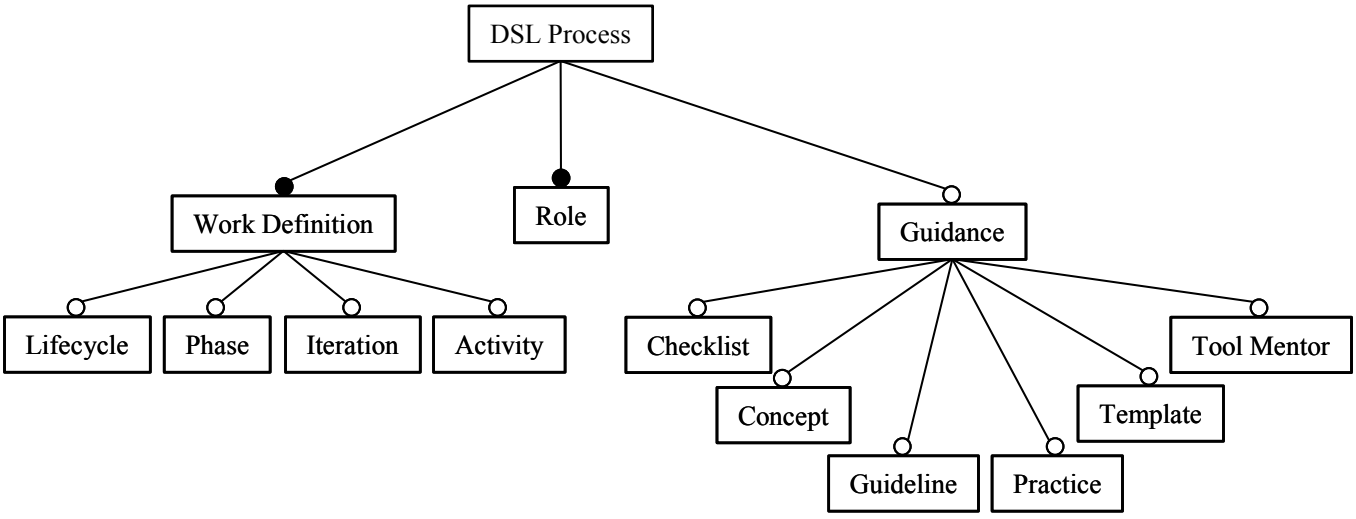

Figure 9. DSL Process Features

3. Comparison of DSL Tools

This section crosses the DSL and DSL tool feature model with a set of tools. Three tools have been selected: a) two main challenging tools, Eclipse GMF (Graphical Modeling Framework) [6] in the context of the Eclipse Modeling Project and Microsoft DSL Tools [18] in the context of the Microsoft Software Factories approach, and b) and a technologically interesting tool, Xactium's XMF-Mosaic [26]. In the following table, GMF means GMF is the

framework for graphical DSL editors at runtime and not the tool for developing graphical DSL editors. For the transformation, each generator is developed apart and uses the results of a graphical DSL editor.

	GMF	Microsoft DSL Tools	Xactium XMF-Mosaic
DSL features			
Language / Abstract Syntax			
DSL meta-model	Ecore (similar to EMOF [20])	Proprietary meta-meta-model. Not explicit but can be discovered by observation.	XCore (similar to MOF [20])
ASG/AST representation	ASG	ASG	ASG
Language / Concrete Syntax			
CS Representation	Graphical	Graphical	Graphical as well as textual notations for the same AS
CS Style	Both, depending on the editor	Declarative	Declarative and imperative
Language / AS 2 CS Mapping			
AS2CS mapping (explicit / implicit)	Explicit, based on a complex but powerful mapping meta-model.	Explicit, but the mapping meta-model is not explicitly defined.	Explicit mapping with XTools. Based on an internal meta-model.
Transformation / Specification			
Problem2Solution mapping Expression	Expressed by a Generator Model built through a Java coded model transformation from the graphical, tooling and mapping definitions.	The mapping to a C# implementation is hard-coded in the template files. They take as input the domain model, graphical, mapping and editor definitions.	The mapping can target text or models. For code, off-the-shelf transformations (to Java, XML, XMI, XOCL) are available. Other M2T transformations can be written in XOCL using XMatch. XMap is used for M2M transformations.
Existence of Viewpoints (perspectives to specify design, implementation, architecture and/or deployment decisions)	Any by default.	Some decisions can be expressed in the editor definition (DSL Explorer window)	No
Transformation / Target Assets			
Destructive / incremental asset update	Jmerge for instance, a JET component, allows incremental changes in generated java sources.	Destructive. DSLFactory.Utilities library claims to integrate incremental code generation.	Destructive
Asset integration support	Generated and additional plug-ins can be packaged for delivery using Eclipse Feature facilities (packaging, license association, link to an UpdateSite).	DSL deployment mechanisms are available.	Deployment is supported. Asset execution requires the XMF-Mosaic platform.
Transformation / Techniques			
Model / Text Techniques	EMF comes with JET. Other tools can be used for M2M, M2T transformations, such as ATL for M2M transformations.	T4 template language used for M2T.	M2T through XOCL and XMatch, M2M through XMap, XML parsers and generators for input/output of XML model instances (T2M, M2T).
Transformation / Execution Environment			
Internal/external execution environment	External	External	Both external and internal with XOCL interpreter that can run XOCL code without compiling.
DSL Tool Features			
Abstraction			
Abstraction: intrusive /seamless	Abstraction encapsulated in the GMF definition models. Their internal meta-models though need to be known, which is intrusive.	Almost seamless for the editor generation. Intrusion for model coherence and validation.	Good because of working mostly with models.
Assistance			
Providing adaptive tool assistance	Yes, e.g., context creation tools, context sensitive environment. The same in generated editors.	Yes, e.g., with context creation tools, context sensitive environment. The same in generated editors.	Relatively good platform adaptive assistance. Bad in generated editors.

	GMF	MICROSOFT DSL TOOLS	XACTIUM XMF-MOSAIC
Providing step / workflow process guidance	GMF generation workflow guidance through a Dashboard. Wizards available for each workflow step. Cheatsheets for learning guidance.	Quickstart solution templates for usual types of diagrams: Task Flow, Class, Component, Minimal Language. They contain ready to use domain, graphics, mapping and editor definitions.	No process guidance, or wizards. Menus are overloaded with multiple purpose actions, which makes them difficult to use.
DSL checking	Possibility to declare audit rules in the mapping model for checking, consistence and validation based on the EMF.Validation framework.	Coherence Rules framework for intercepting model modifications and firing actions accordingly. Validation Framework. C# coding to express rules.	Interactive model checking mechanisms. Dynamic instances of domain models (Snapshots) can be created visually, and complex constraints and queries on the domain model can be tested on the fly. Constraints expressed in XOCL.
QUALITY FACTOR			
Usability	Ability to create very usable DSLs (ergonomics features, adaptive solutions and process guidance).	The definition environment and the generated editor provide good usability. Rapid and easy editor definition. The generated editor is sufficiently mature and has good ergonomics.	Usability is correct, but in terms of ergonomics, there are many drawbacks (lack of an Undo/Redo mechanism, unorganized menus, look and unresponsiveness of the Properties view).
Portability	DSL tool format. Very good interest in being based on Java and Eclipse.	DSL tool format. Windows platform.	Tool dependency, but tool based on Java and tries to respect OMG standards.
Interoperability (part of Functionality)	Model serialization with XMI. Models can also be imported from Rose, UML2 and XSD representations.	Model serialization in a proprietary XML format. No model import/export possible.	Model serialization with XMI.
Maintainability	GMF takes profit from the Eclipse plug-in architecture.	Good, being an integrated part of the Visual Studio environment.	The platform is designed for extensibility. All features over XCore are modeled and therefore available for treatment and use, which should result in a good maintainability.

The versions evaluated here are: Eclipse GMF 1.0.3 [6], Microsoft DSL Tools V1 [18] and Xactium XMF-Mosaic V1.9 [26].

Beyond individual tool capabilities, this table is valuable for other reasons.

- It represents a set of criteria useful for DSL tool comparison and selection.
- Common DSL tool capabilities reflect mandatory technical and user requirements, such as isolation of domain, technical and mapping meta-models, offering very professional end-user tools, or supporting guidance and DSL checking.
- We can notice lacks for DSLs and possible investigations. Limitation of DSL portability is the sign that there is no DSL standard, even if XMI can be a support for DSL exchange.

4. Further Work

The DSL feature model has been experimented in the framework of a DSL tool factory of MDSoFa [14], a Thales software factory tool. Its interest will be to make explicit DSL tool variations which are implicit today.

Several DSLs are currently being developed as part of the Thales work in IST Project Modelplex [10], addressing the representation of different types of architectural knowledge- in particular, security analysis representation, or system capability modeling. This ongoing project involves the combined usage of different tools including a UML modeller, GMF, Xactium XMF-MOSAIC and Microsoft DSL Tool. This work will provide additional insight

with regards to the classification provided in this paper, in a non-striclty software and code generation perspective.

5. Conclusion

A DSL is a problem-oriented language, which combined to transformation tools, such as generators, serves to raise the abstraction level of software and ease software development. But beyond this general definition, DSL and DSL tool variants are numerous. The reason of a DSL feature model is to formalize DSL and DSL tool variants. A first application of this feature model is a DSL tool factory, which applies variations during production of DSL tools. A second application is the selection of pertinent DSL families among all possible families from the feature model. A third application is the definition of DSL tool foundations. A fourth usage is the selection of DSL tools. The feature model, combined with classification criteria, contains needed information to evaluate DSL tools. DSL feature model is in the scope of domain analysis of DSLs. Its clarification becomes a prerequisite for long-term and large-scale DSL developments.

6. Acknowledgments

We thank Philippe Millot, Daniel Exertier, and Catherine Lucas from Thales-EPM, and Véronique Normand from Thales Research & Technology.

Part of this work was realised under IST Project Modelplex. Modelplex is a project co-funded by the European Commission under the IST Sixth Framework Programme (2002-2006).

7. References

[1] ANT, Apache Project, http://ant.apache.org/

[2] ATL, Atlas Transformation Language, official web-site. http://www.sciences.univ-nantes.fr/lina/atl.

[3] Bass, L., Clements, P., Kazman, R., *Software Architecture in Practice*, Addison Wesley, 2003.

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

[5] Czarnecki, K., Helsen, S., *Classification of Model Transformation Approaches*, OOPSLA 2003, Workshop on Generative Techniques in the Context of Model-Driven Architecture.

[6] Eclipse, GMF web site, http://www.eclipse.org/gmf/

[7] Greenfield, J., Short, K., Cook, S., and Kent, S., *Software Factories, Assembling applications with Patterns, Models, Framework, and Tools*, Wiley, 2004.

[8] IEEE, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000, 21 September 2000.

[9] ISO/IEC TR 9126 (1991). International Organization for Standardization, Geneva. An international standard for quality factors.

[10] IST Project Modelplex, http://www.modelplex.org/

[11] JET, Model To Text Eclipse Tool, http://www.eclipse.org/modeling/m2t/?project=jet#jet

[12] Jouault, F., Bézivin, J., Kurtev, I., *TCS: a DSL for the Specification of Textual Concrete Syntaxes in Model Engineering*, GPCE 2006.

[13] Klatt, B., *Xpand : A Closer Look at the Model2Text Transformation Language*, http://bar54.de/benjamin.klatt-Xpand.pdf , 06 July 2007.

[14] Langlois, B., Exertier, D., *MDSoFa: a Model-Driven Software Factory*, OOPSLA 2004, MDSD Workshop.

[15] Langlois, B., Paiano, R., Saeki, M., Sånchez-Ruỉz, A., *Domain-Specific Software Development Terminology : Do we All Speak the Same Language?*, Proceeding of the 6th OOPSLA Workshop on Domain-Specific Modeling.

[16] Lauesen, S., *Software Requirements, Styles and Techniques*, Addison Wesley, 2002.

[17] MetaEdit+, MetaCase, http://www.metacase.com/mep/

[18] Microsoft, DSL Tools web site, http://msdn.microsoft.com/vstudio/DSLTools/

[19] OMG, *MDA Guide Version 1.0*, omg/2003-05-01, 1st May 2003.

[20] OMG, Meta Object Facility (MOF) 2.0 Core Proposal, ad/2003-04-07, 7 Apr 2003.

[21] OMG, *Software & Systems Process Engineering Meta-Model (SPEM 2.0)*, ad/2006-08-01.

[22] Stahl, T., Völter, *Model-Driven Software Development*, Wiley, 2006.

[23] TCS (Textual Concrete Syntax), Eclipse/GMT component, http://www.eclipse.org/gmt/tcs/

[24] Voelter, M., *Domain Specific Languages, Implementation Technologies*, http://www.voelter.de.

[25] Voelter, M., *Textual DSLs*, http://www.voelter.de

[26] Xactium web site, http://www.xactium.com/

[27] Xtext Reference Documentation, http://www.eclipse.org/gmt/oaw/doc/4.1/r80_xtextReference.pdf