

The Knowledge Industry Survival Strategy Initiative  
(KISS)

Tony Clark<sup>1</sup> and Jorn Bettin<sup>2</sup>

<sup>1</sup>*School of Computing, Thames Valley University, St. Mary's Road, Ealing, London, UK, W5 5RF*  
[tony.clark@tvu.ac.uk](mailto:tony.clark@tvu.ac.uk)

<sup>2</sup>*Sofismo, Sägestrasse 50, 5600 Lenzburg, Switzerland.*  
[jbe@sofismo.ch](mailto:jbe@sofismo.ch)

## Abstract

*The commercial benefits claimed for software based on Domain Specific Languages are well documented. Many DSL tools exist and are being used as point solutions. Tailoring of notations to the specific application domain and combined use of several languages define the nature of the approach, and constitute the source of the achievable benefits. Unfortunately data representations and the mechanisms used to integrate languages tend to be highly tool specific. This compromises the use of DSLs in building tool chains that may contain components from several suppliers. KISS is an industrial initiative that aims to define a set of fundamental principles for the design and integration of DSLs and to provide a framework for the development of DSL tools that achieves interoperability.*

## 1 DSL: Benefits and Problems

The main motivation for the use of a DSL [1] is the desire to express problems in a compact form that reflects the natural terminology of human domain experts, and that is easily accessible to software tools. In short, DSLs are raising the level of abstraction of software specifications and of knowledge representation in general [2]. When DSLs are used to formalize the results of domain analysis, the result is a clean separation of concerns in the problem space [3]. This is a major advance over aspect oriented programming, where separation of concerns is only achieved in the solution space. The value of a DSL increases with the intuitiveness of the concrete syntax. Visual and graphical elements may be needed to increase usability, and often such languages are referred to as domain specific modelling languages (DSML).

The level of interoperability between current DSL tools is comparable to the level of interoperability between CASE tools in the 90s. To increase the popularity of DSL based approaches, this needs to change. With the extensive use of outsourcing and with the increasing investment in open-source software, software development has become highly decentralized, and an assumption that all parties in a global software supply chain will use identical tooling is simply not realistic. As a result today's software supply chains are much less automated than supply chains in other, more mature industries.

For example, the typical outsourcing relationship between two software manufacturers is based on informal specifications. Either the tooling used by the outsourcer is unknown to the customer, or if the customer is more discerning, the outsourcer sets up an environment that matches the one of the customer, usually at a substantial cost (licenses, staff training, etc.). The inefficiencies are such that it is not uncommon to set up mixed teams with staff from both organisations, and to rely heavily on extensive travel and face-to-face collaboration to achieve the desired outcome.

The challenge lies in identifying the ingredients needed to create highly automated software supply chains that minimise the effort to integrate new suppliers of specialized software artefacts. Meeting this challenge requires a significant increase in the use of formal (yet highly compact) specifications, and advanced tooling for creating, managing, and exploiting formal specifications to the fullest.

Ideally, supply chain participants would be able to exchange formal language definitions and models, and would rely on a shared implementation of basic services for managing such model based artefacts. A

customer may then provide an outsourcer with definitions of specification and implementation languages, and with appropriate instances of specification models. The outsourcer should be free to choose which tools are most appropriate for producing a solution that meets the specifications.

The KISS initiative proposes a pragmatic bottom up approach to interoperability that is driven by an open community. Existing standards for modelling tool interoperability lack public reference implementations of test beds that can be used to establish compliance, and they are focused on the Unified Modelling Language rather than the requirements of heavily DSL based approaches.

This paper is structured as follows: section 2 introduces a consistent terminology for describing DSL tool-chain components; section 3 reviews the current approaches and standards that influence the construction and interoperability of DSL applications; section 4 states the KISS values and aims; section 5 describes the issues that influence general tool interoperability and section 6 refines these to DSL tools; section 7 describes how meta-models influence tool interoperability; section 8 identifies a collection of KISS compliance levels that are proposed in order to achieve interoperability; section 9 describes how KISS is organized and presents the KISS roadmap; section 10 places KISS in context and reviews related work.

## **2 Definitions**

This section defines the key terms used throughout this paper to describe DSL tool-chain concepts. To motivate the definitions, we use the following simple collection of independent modelling tools as an example:

- A. A tool to support business motivation modelling in terms of organizational goals. Each goal places constraints in terms of pre- and post-conditions on high-level information structures describing the state of a business. Goals can be decomposed into conjuncts and disjunctions of sub-goals.
- B. A tool to support design of business components. Each component has an interface and an internal data model. The interface is implemented using an action language supported by the tool. The tool can execute the action language against information snapshots.

- C. A tool to support the definition of business processes. The processes are event driven and produce actions handled by component interfaces. The tool can execute the processes (either by simulation or deployment).

Each tool can be used to model different features of a commercial enterprise. The tools have been developed independently, however they could be combined to form a tool-chain that supports the development and deployment of business processes that are generated from the analysis of high-level business motivation models.

### **2.1 Tool-Chain Scenario**

The following is a scenario that described how tools A, B and C can be combined to form a tool-chain. In order to construct the tool-chain, new tools will be required and these are noted:

A business architect uses A to produce a model describing how a business should change to achieve a collection of business goals. The motivation model is developed by decomposing a single root goal to the required level of detail. At each level, the goals specify pre- and post-conditions on high-level information structures describing aspects of the business and its commercial environment.

An information analyst uses tool B to construct a detailed description of the business in terms of its information structures. The models saved from A are imported into B as a starting point using a translation process (tool D) that acts on models and diagrams. Models constructed in B can be checked for consistency with respect to models from A using a tool E.

A business analyst uses C to construct business processes. The component interfaces from B are loaded into C (using tool F). Changes to interfaces can be checked with models from B using tool G. Execution of the processes by C is checked against the pre- and post- conditions from A-models via B.

The scenario given above is not the only way in which some or all of A, B and C can be combined to produce useful tools. The extra tools D to G perform translations and analysis on the models produced by A to C and therefore must have access to the models in a suitable format and be able to invoke operations provided by A to C. Furthermore, the processing

performed by D to G is fairly general purpose somehow specialized to the domains implemented by A to C.

## 2.2 Tool-Chain Requirements

Analysis of the tool-chain scenario leads to the following requirements for technology to support tool-chain construction from modelling tools where the languages are non-standard (i.e. DSL modelling tools):

**Req1:** *Tools should be able to persist models.* This is a basic requirement on modelling tools and provides an opportunity for DSL tool-chains in terms of the persisted model data.

**Req2:** *A tool must be able to supply its models to another tool.* The consuming tool will require the model elements, but may also require layout information.

**Req3:** *A tool must make its data format available.* The data formats used by of tools must be available so that tools can transform from models in one tool to models in another tool. Ideally the data formats will be available as a model so that general purpose tools can process them.

**Req4:** *Inter-tool control.* A tool must offer an API that can be called via another program in order to link it into a tool-chain. In the scenario, most of the new tools used existing APIs to invoke existing functionality after transforming models from one format to another.

**Req5:** *Precise model semantics.* The semantics of the models used by a tool should be precisely defined in order that other tools can use them effectively.

## 2.3 DSL Tool Definitions

KISS aims to build a framework within which Industry can make progress towards achieving the requirements on DSL tools that are described in the previous section. In order to describe features of the KISS approach, it is useful to set down some definitions of terms that arise directly from the requirements. This section defines terms that are used throughout the rest of the paper:

**Def:** *Model.* The term model is used consistently with the following dictionary definitions:

- (Noun) a pattern or mode of structure or formation.
- (Noun) a simplified representation of a system or phenomenon, as in the sciences or economics, with

any hypotheses required to describe the system or explain the phenomenon, often mathematically.

- (Verb) to simulate (a process, concept, or the operation of a system), commonly with the aid of a computer.

Within the context of KISS a model consists of model elements that are manipulated by a DSL tool. A model consists of many aspects that are defined by the model's language.

**Def:** *Abstract Syntax.* The abstract syntax of a model is the computer-centric representation of the model elements in terms of data structures. Often this is the fundamental representation of a model. For example tool A in the example above has an abstract syntax that consists of goals, sub-goals etc.

**Def:** *Concrete Syntax.* The concrete syntax of a model is the human-centric representation of the model elements in terms of how they appear on the screen. The concrete syntax can be viewed as a secondary representation since there can be many concrete syntaxes for the same abstract syntax. The concrete syntax can be thought of as a language in its own right where the abstract syntax is related to the concrete syntax via a mapping. For example, tool A may have a concrete syntax where goals are shown as nested boxes; alternatively goals might be given a textual syntax.

**Def:** *Well-formedness.* The syntax (abstract or concrete) of a modelling language must obey certain constraints in order that the models are constructed correctly. These constraints do not state anything about what the models mean. For example, tool A may specify a constraint that goal decomposition trees must contain no loops.

**Def:** *Language Definition.* A modelling language definition is a collection of definitions that includes the concrete and abstract syntax definitions and the semantics of models expressed in the language.

**Def:** *Semantics.* A modelling language has a semantics that defines the meaning of any model. There are many ways in which a model can be given a semantics including: a transition machine; a mapping to a semantic domain. A tool must implement a specified aspect of a model's semantics.

**Def:** *Meta-Language.* A language describes how to represent and manipulate its models as data structures. A meta-language describes how to represent and manipulate a *language* as a data structure. Meta-

languages are important to interoperability because standardization at the meta-level allows generic tools to be constructed. The relationship between models, languages and meta-languages is called a *golden-braid* [4] and occurs in modelling languages such as UML and programming languages such as Smalltalk. The three levels can be replicated to produce systems of n-levels, although often three levels is sufficient. It is usual for the top-level meta-language to be self-describing.

**Def: Generic Tool.** By nature, a DSL tool is specialized to a specific problem domain and contains domain-specific knowledge encoded in terms of model transformations and model analysis. At first sight, there would appear to be little scope for technology reuse across multiple tools in different domains. However, the golden-braid defined above provides scope for general technologies implemented as generic tools. A generic tool is written in terms of a meta-language shared between multiple DSL tools. The generic tool can be populated by supplying information about a particular DSL tool language expressed using the meta-language. Examples include: code generators, model well-formedness checkers; model walkers; model repositories. The key to achieving reuse is to standardize on a shared meta-language. Many of the model processing technologies listed in these definitions can be defined as generic tools.

**Def: Model Transformation** is a key DSL technology that applies in all domains. Transformation may involve turning one model into another or code generation. Transformational technology is often implemented using a generic tool.

**Def: Model Persistence.** A DSL tool must provide a way to save models. This is often done using a file-based storage mechanism in a textual format, although model repositories are also used. The format may be standard (for example XML) or using a proprietary data representation. Both concrete and abstract syntax are persisted.

**Def: Model Weaving.** Multiple models are often used to describe different aspects of a system. A unified view is constructed by weaving the models together. This is an example of a transformational technology.

**Def: Model Execution.** Many DSLs are developed in order to abstract away from the technical details of program code with a view to generating an executable directly from the model. The executable can be

produced by transforming the model into program code or, where the model is sufficiently expressive, by executing the model directly. In order to run the model, a model execution engine is required.

**Def: Model Editor.** A DSL tool supports the construction of a model in a domain specific language. The construction of the model is supported by a model editor that implements the concrete syntax of the language. Model editors often implement a mix of graphical and textual syntax elements.

**Def: Model Reader.** A model that is persisted using the concrete syntax format can be processed in order to create the abstract syntax format. Where the language is purely textual, such a tool is called a parser.

**Def: Model Writer.** A model is exported using a model writer. Each model writer walks the model and produces output. For example, code generation is produced by writing out a model as program code.

**Def: Model Walker.** Model processing usually involves starting at a given point in the model (often designated as a root of the model) traversing model links and performing actions in terms of the model elements as they are encountered. The traversal and associated processing is referred to as a *walk* of the model. Many tools can be constructed as specific types of model walker.

**Def: Tool Interface.** A tool generally provides an interface that responds to events such as button-click or menu-selection. These events are transformed inside the tool to invoke actions defined by the tool API. Often the API is not exposed to external tools. In order to produce DSL tool-chains, the API must be exposed and ideally expressed using a tool interface language. A tool interface meta-language provides a basis for tool frameworks that support a plug-and-play architecture.

**Def: Tool Chain.** A collection of tools that are coordinated to perform a collaborative task. The collaboration may make use of the persistence format of a tool or allow tools to interoperate using in-memory data. In general, a tool chain requires tools to have knowledge of each other's interfaces and data formats.

**Def: Tool Framework.** A tool framework is a general purpose framework for building tool chains. Tools are inserted into the framework which facilitates their interoperability and manages their resources. The framework has no prior knowledge of the tools that can

be inserted into it. Tool interoperability is achieved through scripts that invoke operations in a tool interface.

### 3 Current Approaches and Standards

Technologies for generating DSL based tools and tool-chains date back to Meta-CASE tools in the 1980's (even earlier if textual DSLs are taken into account – e.g. Lisp). These tools are representative of a class of tools that allow the user to construct meta-models of languages and development processes. The models are then instantiated to produce tools that support the language and the process in terms of editors, code generators and wizards. Many of the meta-tools that support this technology suffer from bespoke data representations which makes interoperability via third party developers difficult. Such tools are usually capable of participating in tool-chains, however there is currently little guidance on how the tools can be adapted in order to fit in.

UML was developed in order to address the issues of non-standard model representation. The UML definition is written with respect to a meta-language – MOF that aims to provide a basis for standard model interchange and for standard language extensions. There are several reasons why UML can be argued as unsuitable as a basis for DSL tool-chains. Firstly, its relationship with MOF is weak: UML is an instance of MOF, but not an extension of it; therefore MOF-UML does not implement a so-called *golden braid*. UML extensions are not written using meta-language extensions, they are written using a restricted form of meta-access via profiles and stereotypes. Secondly, UML aims to be a general purpose modelling language which means that it provides a large array of general purpose features, leading to a comprehensive language for expressing application domains rather than languages. Thirdly, the UML interchange technology, XMI, is generally considered to be difficult to implement completely. Few (if any) UML tools achieve complete interoperability.

MOF is a simple meta-language used to define UML and other languages. Therefore, MOF is a suitable candidate as a basis for tool interoperability. However, MOF tends to be instantiated to produce languages (e.g. UML); any semantic definitions in MOF are weakly connected to models expressed in modelling languages defined with MOF.

Eclipse is a platform for developing tools. It provides a plug-in framework whereby tools (typically in Java) are installed using tool-descriptors (typically in XML) that link tools together and embed the tools into the framework. Tool-descriptors include information about the external interface of the tool component and link standard GUI features such as menus, property sheets and data browsers to tool-code.

Eclipse provides tool integration at the interface level – both in terms of the operations required by a program linker and in terms of tool graphical user interface. Eclipse aims to provide a coherent user experience even though the application consists of many tool components chained together. Eclipse does not aim to provide data integration or any specific support for the languages provided by any of the tools in the chain.

The Eclipse Modelling Project [5] provides a collection of open-source technologies that can be used to develop modelling tools. These include EMF and GMF. EMF is a technology that can be used to develop data definitions against a common meta-language called eCore. eCore provides reflection and can be used to raise events when eCore-based data changes. The original aim of eCore was to provide a basis for Java tool chains via data interchange and listening for data change events. EMF is a collection of eCore based tools for simple editor generation. GMF is a collection of EMF based tools that supports graphical editor generation.

eCore is a good candidate as a basis for tool interoperability. Its strength is that it has a clear pragmatic semantics: its implementation in Java. Its weakness is that some of the features are based on implementation concerns and it is incomplete. Both EMF and GMF are very useful and are good examples of the kinds of technologies required when processing meta-models; however, they are only part of the solution.

The DSL tools initiative at Microsoft [6] aims to allow tool-chains to be developed using Visual Studio. The DSLs are mixtures of graphical and textual languages developed using a bespoke meta-language and specifically aimed at generating code into MS solution architectures. Like the Eclipse/EMF/GMF technology described above, this is one part of the tool-chain solution. Unfortunately, the languages and technologies are bespoke and therefore not an open solution for building tool-chains. However, Microsoft has emphasized its commitment to interoperability and has

recently engaged with the OMG stating that it hopes to influence meta-technologies.

Many existing tools for DSLs are *closed* in the sense that it is difficult to access and manipulate the languages and models in ways that were unforeseen by the tool developers. A *tool-centric* view of languages does not readily support interoperability since the use-cases for languages become fixed and difficult to extend. Interoperability would be facilitated by taking a more independent *user-centric* view of languages where each tool contributes part of the language semantics. In this way, no tool could claim exclusive ownership of a language and would be forced to make allowances for other use-cases and extensions to its own use-case via interoperability.

In conclusion, many technologies exist that support working with DSLs. Some have bespoke notations and others are standard but are suboptimal for a variety of reasons. Lessons can be learned from looking at the process of standardisation from a wider angle [7], and from considering the practical realities of lock-in as experienced by software users [8].

Part of the problem is that there are no guidelines together with suitable open technologies for building DSL tool-chains. We believe that the time is right to consolidate the experiences of the last 10 years in terms of modelling, tool-chains and DSLs to produce a set of values, guidelines and open technologies.

KISS is an initiative that aims to achieve this. The key aims, working hypotheses and processes of KISS are described in the rest of this paper.

#### 4 KISS: Aims and Core Values

KISS aims to provide guidelines to support the use of domain specific methods and technologies in industry. In particular, KISS will support the construction of tool-chains that are built by third parties using components consisting of a mixture of commercial and open-source DSL tools.

To support this aim KISS has identified a number of core values that will inform the development of a DSL tool interoperability framework. These values are as follows:

1. We strive to [automate](#) software construction from domain models; therefore we consciously distinguish between building

software factories and building software applications

2. We work with [domain-specific assets](#), which can be anything from models, components, frameworks, generators, to languages and techniques
3. We support the emergence of [supply chains](#) for software services, which implies domain-specific specialization and enables mass customization
4. We see [Open-Source](#) infrastructure software as leveling the playing field for economically viable software supply chains and as a catalyst for Open standards
5. The methodologies we use conform to the values of the [Agile Manifesto](#)

KISS specifically addresses tool-chains constructed from multiple DSL-based tools and technologies. KISS views a *language* as a collection of use-cases driven by business requirements. Each use-case places notational or semantic requirements on the language.

A technical language definition artefact, either commercial or open-source, is by definition a partial view on a language that satisfies a particular business use-case. Such use-cases may be orthogonal or may be related by specialization and generalization. Therefore:

6. We view a [language definition](#) as the union of all use-cases (notational or semantic) driven by business requirements provided by the user-community. Particular DSL-technologies can only *contribute* to a language definition. They cannot *constitute* a complete language definition by themselves.

#### 5 KISS: Tool Interoperability

Imagine a world where software systems are created by composing models and tools that operate on these models. Each tool covers a specific aspect of the problem space or the solution space. Each tool is a component that may have originated from a different supplier: some open-source and some proprietary.

So there are tool developers and there are tool-chain compositors. There is a requirement on developers to provide sufficient information for the compositors to be able to their job. This amounts to the compositor being

able to tell what the tool does and what data it supplies and consumes.

Imagine a tool framework that supports tool interoperability and the construction of tool-chains. In order to specify such a framework it must be possible to manipulate tools as first class citizens and to access the key tool-features that contribute to the construction of a tool-chain. This section develops models that reify the key aspects of tools that would support the specification of the framework. KISS will refine these models.

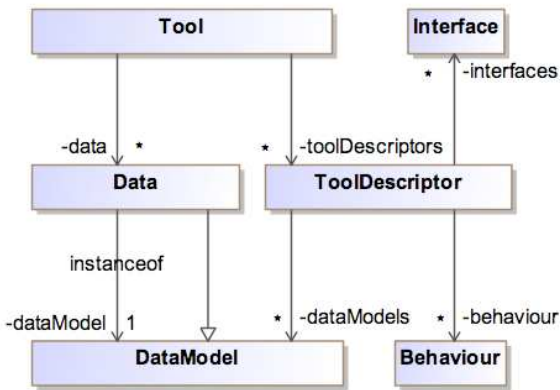


Figure 1: Tool Descriptors

Figure 1 shows a model that includes elements designed to support interoperability. A tool provides externally visible data and interfaces. The key to interoperability is that the data and interface descriptions are in a format that other tools can use. This is achieved through the use of a tool description that provides three components:

1. Data models (meta-models) describe the externally visible data provided and consumed by the tool.
2. Interfaces describe the functionality offered by the tool.
3. A model of the tool behaviour.

The tool descriptor is effectively a meta-model of the tool and as such all the languages used to describe the tool descriptor components must be standardized.

Figure 1 shows that Data is an instance of a model called the DataModel. This is a standard instance-type relationship and intended to show that a tool descriptor must include a model of the data types. We would like to support an n-tier approach where tool-descriptors can be viewed as data at the next level up. Therefore,

data is viewed as a special case of data-model, where a data-model is described using a meta-data-model.

The data and interface models are static. The behavioural model of the tool is dynamic and describes what the tool does. In principle, this can be done to any level of detail and take any form; however, a convenient and universal approach to behavioural models is to provide an open-source implementation of the tool that captures the behaviour.

The use of open-source behaviour models is a key feature in achieving interoperability. The behaviour models may arise in different ways. For example, an open-source *reference implementation* of a new category of tool is an attractive way of achieving consensus. The open nature of the process removes technical obstacles imposed by proprietary interests and is likely to attract the best development talent. In addition, making the process open fosters a Darwinian approach that makes the selection of the best technical solutions more likely.

An open-source behavioural model may be used as-is or may just define sufficient behaviour in order to standardize other implementations. Behavioural models do not restrict other implementations from extending the behaviour and thereby adding proprietary commercial value to the tool, however the additional functionality will achieve lower levels of KISS compliance.

Given tool models including behavioural descriptions, it is possible to develop a notion of tool equivalence. There are various levels of equivalence from very weak to very strong. For example, equivalence based on interfaces tends to be quite weak, data equivalence is much stronger and finally behavioural equivalence is the strongest. Whilst in theory behavioural equivalence is undecidable, it is practical in terms of a collection of specific test-cases that is incrementally built up over time by the tool user community.

An equivalence based on data or behaviour requires a common representation of the information in tools. In turn, this requires a common language to express the equivalence rules – a common meta-language. This issue is discussed further in the following section on DSL tool interoperability.

Tool descriptors allow tools to be composed into tool-chains. A tool-chain is a collection of tools that collaborate in order to achieve a co-ordinated task.

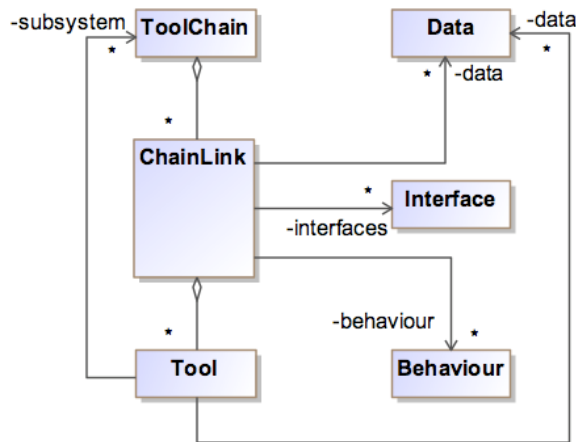


Figure 2: Tool Chains

Figure 2 shows a model of tool chains. A tool chain consists of a collection of tool links that describe the co-ordination of tools in terms of their tool descriptors. A link describes the shared data, the interface operations and the shared behaviour.

A number of constraints must be imposed on the link in order to make it well formed. The data, interface and behaviours involved in the link must all be consistent with the tools (via the tool descriptor) of the tools involved in the link. This implies that there is a *consistent-with* relationship between the descriptor components.

The principle of tool descriptors and tool-chains can be applied at multiple levels. Each tool can be viewed as a collection of co-ordinated sub-tools. Figure 2 shows that a single tool has a sub-system which is a tool-chain.

Given a reified description of both tools and tool-chains it is possible to write tool frameworks that use the tool models to support the construction of composite tools.

Eclipse is an example of a tool framework that uses various models of tools (usually expressed as XML) to support the composition of Java projects. Each Eclipse Java project contains configuration information that describes the exported and required Java packages and

the interface operations that are exported as menu operations. Eclipse checks the dependencies between projects, making appropriate links, and provides diagnostics based on missing tool elements.

Eclipse supports tool building from an implementation perspective; the project view is linked to Java implementation features. The KISS proposal differs from Eclipse in terms of abstraction over the structure of tools, a model of the data being exposed by tools and the use of a behavioural model to enhance reuse.

KISS advocates the use of models to support the selection, reuse, composition and interoperability of tools in tool-chains. The basic features required for modelling tool-chains are as follows:

- a common modelling language for tool descriptions.
- models that describe the data exposed by tools.
- models that describe the interfaces offered by tools.
- models that describe tool behaviour which may pragmatically consist of open-source implementations of tool behaviour.
- tool-chain models including tool composition constraints expressed in terms of tool description models. Where appropriate tool-chain models are recursively compositional.
- frameworks that support the selection and composition of tool-chains based on models of tools and tool-chains.

This section has described the KISS approach with respect to general purpose tools. A structure for tools and tool-chain models has been outlined. It is not possible to expand on the internal structure of the models without knowing more about the type of tools or applications. The next section discusses the KISS principles in the context of DSLs and shows how KISS models are specialized for domain specific technologies.

## 6 DSL Tool Chains

All of the above applies to general tools. We can now specialize this argument for tools based on domain specific languages.

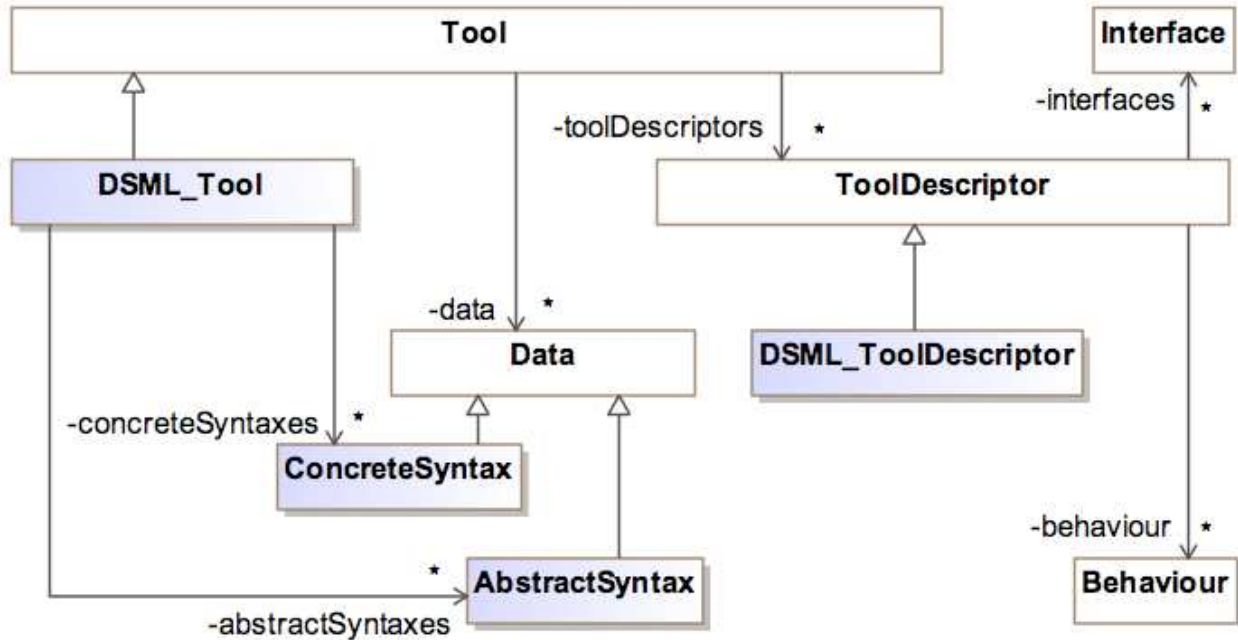


Figure 3: DSML Tool Descriptor

A DSML tool provides the user with a view on application specific data. The tool is a model walker that processes a particular language that consists of a concrete syntax (the human-friendly representation) and the abstract syntax (the computer friendly representation). Figure 3 shows the specialisation of the tool descriptor model for DSML tools.

The two types of DSML data imply various forms of functionality and interoperability. Tools must create and store the data representations and can interoperate in terms of syntax at either the concrete, abstract or both as shown in Figure 4.

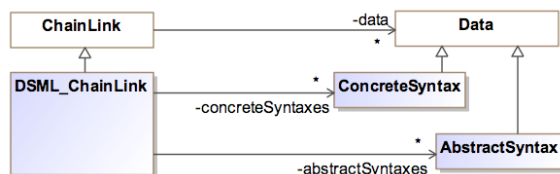


Figure 4: DSML Syntax Interoperability

Note that there is no requirement for the concrete or abstract syntax to take any particular form; so, for example, both textual and graphical DSL tools are supported. For example, Figure 5 shows how the

concrete syntax of a DSML tool can be specialized to accommodate both graphical and textual syntaxes.

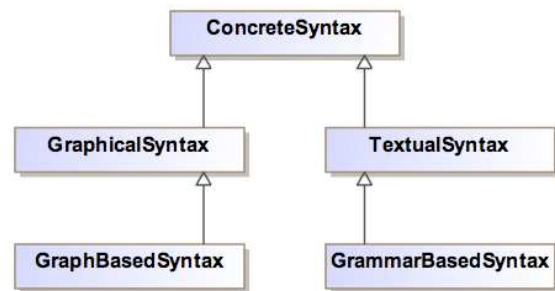


Figure 5: DSML Syntax Categories

Graphical syntaxes tend to be based on graphs consisting of vertices and edges. Many variations are possible based on features such as labelling, decoration, layout and iconization. Textual syntaxes tend to be based on formal grammars such as those described by BNF.

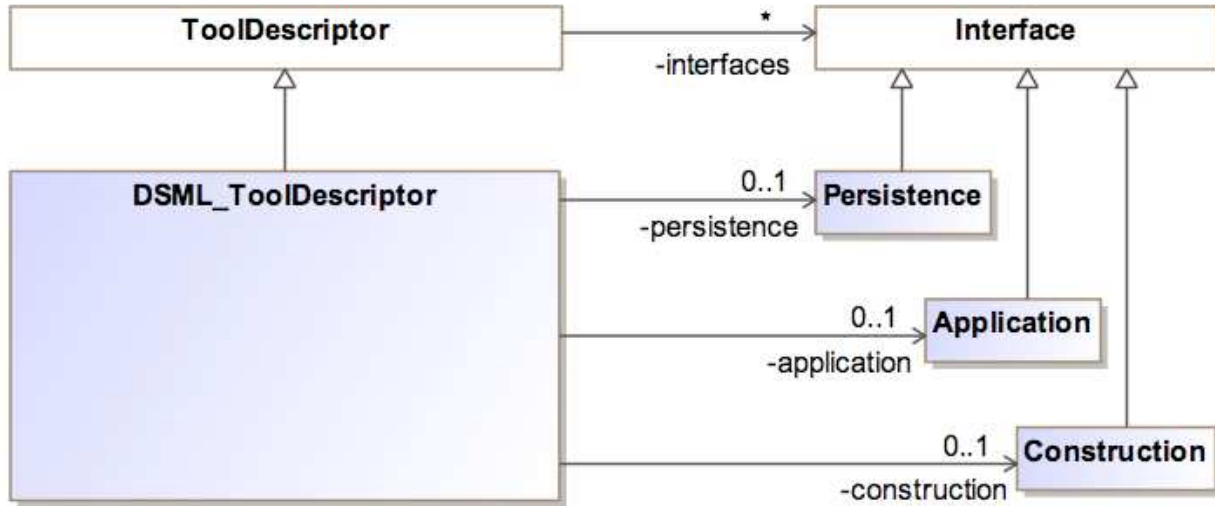


Figure 6: DSML Interfaces

DSML tools must support the construction and manipulation of models in the application specific language. Therefore tool interfaces have a specific structure. Figure 6 shows the specialization of ToolDescriptor to support domain specific tool interfaces. There are three basic aspects to a DSML tool:

- the construction interface allows new domain specific models to be constructed and edited (Construction).
- the persistence interface allows the models to be saved to appropriate storage media and subsequently uploaded (Persistence).
- the application interface supports some domain specific processing of the models (Application).

General tools can have any behaviour. However, there are specific categories of DSL behaviour that are useful to identify as shown in Figure 7.

A DSML tool supports a language which consists of syntax rules. Often these rules are either graphical or textual, however in all cases there are rules of construction that govern well-formedness. The WellFormed model describes how the models in a language can be checked.

A DSML tool language may be executable. This includes running a model in the conventional sense of a programming language, but also includes using the model as a constraint and checking it against

candidate elements. The ModelExecution model describes the rules of execution.

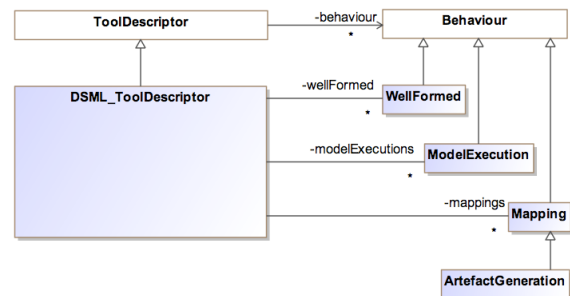


Figure 7: DSML Behaviour

A model is often an abstraction over some implementation technology. The developer constructs models using the DSML tool and then translates the models to a target technology. Targets can include programming languages, database tables, XML-based configuration files etc. The Mapping model describes how to turn an instance of the domain specific model into a lower level specification (source code in a target language or configuration files for a particular target technology). In Figure 7 a specific example of a mapping model is shown which performs artefact (code) generation.

DSL tools are often generated by a tool generator. The features described in this paper imply constraints on the generator, for example the generator should generate a tool descriptor model and may be made interoperable by going one meta-level up.

## 7 Meta-Models

A key feature of the interoperability of tools and DSML tools in particular, is the ability for tool-chains to share data.

General tools may operate on arbitrarily structured data. Therefore, to support general tool-chains:

- all participating tools must agree on a collection of type models
- data in a general tool-chain is described by a two-level system: instances and type models.

We have shown that it is possible to identify a greater degree of structure in the descriptors for DSML tools. Such tools consist of user models which are constructed in domain specific languages. The type model for the user data is specific to the DSML tool. The DSML tool offers a specific language as shown in Figure 8.

In order to achieve DSL interoperability, tool-descriptors must make both user data and the language definition available. Therefore, instead of requiring a common data type model, DSL tool-chains must support a common *meta-model*, i.e. a language for describing languages [9].

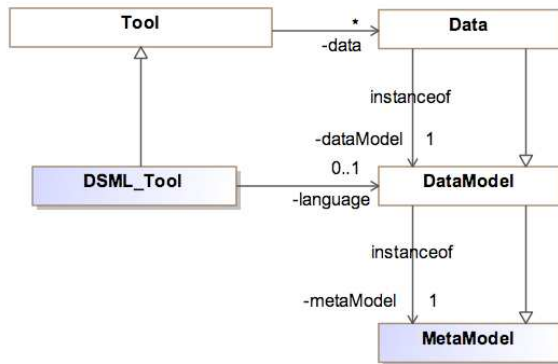


Figure 8: Meta-Models

A number of candidate meta-models exist, including MOF from the OMG and eCore which is the meta-model for the EMF language. KISS will propose a standard meta-model for DSML tool interoperability and will aim to make this language as simple and universally applicable as possible.

KISS will provide an open-source implementation of the meta- language, possibly by adopting one of the existing open-source candidates.

## 8 Compliance

The KISS interoperability levels can be used to rate interoperability between any two specific tools for implementing and using domain specific languages.

Two tools are deemed to be interoperable at level n if the tools have sufficient knowledge about each other in order to interact in order to achieve a co-ordinated task.

This allows pair-wise tool interoperability to be certified. Furthermore, the levels of interoperability support a definition of equivalence between tools since the knowledge that one tool has about another tool constitutes a specification that can be satisfied by any number of concrete implementations; each implementation is equivalent to another up to a given levels of interoperability.

Note that KISS interoperability at the syntax level is not limited to interoperability for model instances conforming to one particular meta-model, but is defined to include interoperability for model instances conforming to any meta-model. This implies a shared meta-meta-model implementation between interoperable tools.

Consider general tool interoperability. We can identify the following levels:

0. The tools have no knowledge of each other and therefore cannot inter-operate.
1. Tools are interoperable at the interface level and can pass simple unstructured data. Tools can invoke each other's services, supply values and get results. However, the lack of structured data means that no significant information can be passed.
2. One tool can access information that has been serialized by another tool. Therefore, information can be communicated between tools. This mechanism is rather unwieldy; however it does not require the tools to share the same address space, or to conform to a given framework. In order to communicate the tools must have knowledge of the data structures; therefore, this implies some form of universal meta-language

(not necessarily shared between the two tools – possibly used as a translation service).

3. Tools communicate by sharing data. This implies a shared meta-language. Note that the shared data may be interpreted differently by the two tools, however they agree on the shared structure. There is not necessarily any shared knowledge about when or why changes are made to the data by any of the tools sharing it. Tools can invoke each others interface operations and pass shared information.
4. Tools co-ordinate with respect to the shared data. This is a limited form of behaviour conformance since each tool informs the others that they intend to process the information in fixed ways (for example when they have changed the data). This may be provided in terms of a shared service (as in the case of eCore).
5. Tools have knowledge of the behaviour of each other and can understand when and why information is being processed. There is scope of knowledge-based tool co-ordination.

These levels of interoperability are very general and most tools and tool-chains do not get beyond level 2 or 3.

Consider these levels with respect to DSL tools. A DSL tool specializes the notion of a tool by identifying specific interfaces, and specific features of the data representation. For example:

0. No specific features are identifiable.
1. DSL tools have specific interfaces, for example persistence and model definition. Therefore, it is possible to be interoperable with respect to different aspects and different phases of the DSL tool life-cycle.
2. DSL tools can store at least two types of information: the language definition and the

models. Interoperability may be at either level. For example, one language definitions may be imported into another tool. Models expressed in an imported language may also be imported. Models may take the form of abstract syntax or concrete syntax. DSL data interoperability may be defined in terms of either type of syntax.

3. Shared data in DSL tools implies a shared language definition either at the language definition level or at the model level. In turn this implies a shared meta-meta language.
4. Co-ordination between DSL tools can occur at the language definition or model level. This implies a common language manipulation service that tools register with.
5. Behaviour of DSL tools can take a number of forms. DSLs are made active by translation to other languages or by direct execution. In either case, the problem of interoperable behaviour is made more tractable because the information is reified.

Beyond KISS interoperability level 5 further levels are imaginable for the distant future that address comparability of quality of services attributes. In addition each of the levels 0 – 5 may be decomposed further to identify detailed aspects of DSL based tools.

Figure 9 shows the different KISS compliance levels and the key interoperable features that must be achieved by tools.

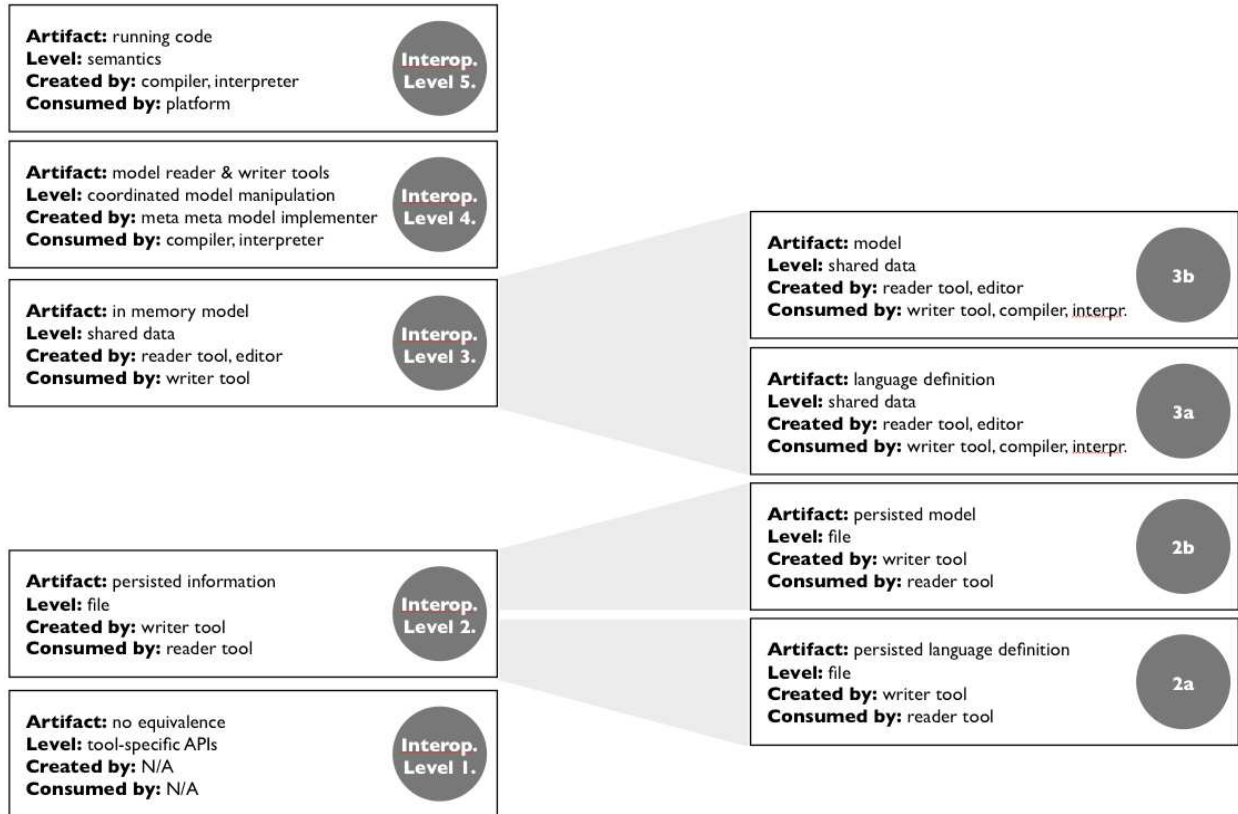


Figure 9: Compliance Levels

## 9 KISS Organization and Road Map

The KISS initiative consists of a collection of organizers and supporters. Over the next 18 months, the organizers aim to run a series of workshops to identify industrial requirements for DSL tool-chains and to examine candidate solutions.

The end result of this phase of the KISS process will be a framework of key values and definitions that can be used to analyse and evaluate approaches to industrial strength DSL technologies for building tool chains. This paper is an initial proposal for discussion.

More information about KISS, its organizers, processes and current progress can be found at: <http://www.industrialized-software.org/kiss-initiative>. There are several ways to participate in KISS: by contacting an organizer and signing up as a co-organizer or supporter; by attending one of the workshops to contribute to the debate; by presenting a paper describing a contribution to the aims of KISS.

The initial phase of KISS will shape any subsequent phase. For example, this may take the form of an agreement amongst KISS members to build key infrastructure components to support a KISS-based framework for DSL tool-chains.

## 10 Review and Related Work

This paper has identified problems with the current state of the art when building tool chains using DSL technologies. We have proposed an approach to building DSL tool-chains based on taking a user-centric approach to defining interoperable DSL tools by modelling the appropriate features. An industry led community of practitioners that apply KISS values and principles in their work has been formed to develop these ideas further.

Tool integration has a long research history: [10] is a comprehensive survey of the field and [11] describes the current trend towards integrated tool environments and an approach to integration using general tool interfaces.

The Eclipse environment was specifically designed for tool integration: [12] describes levels of integration within the Eclipse environment that makes a distinction between interface integration and data integration and which correspond to the interoperability levels identified in this paper. In [13] a distinction between data integration and control integration in tools is made and the paper also argues against the organization in a central repository, favouring instead point-to-point integration.

[14] provides motivation for software engineering environments that are based on multiple domain specific modelling languages and the distinction between interfaces for model management and for model processing. This paper is more recent than [13] and argues for a single data repository that contains all model data.

The pitfalls of consortium based standardisation initiatives are described in [7] which illustrates the increasingly strategic role of the open-source concept for achieving practical interoperability.

An analysis of the implications of different flavours of lock-in that can occur when building solutions that make use of external components and industry standards is described in [8].

[14] Discusses the requirement for meta-models in each domain specific tool and the use of a single meta-meta-model that unifies all the tool languages.

An approach to integration that does not require a universal language is described in [15]. Third party tools are integrated via meta-level relationships that are used to synchronize models. An example is given that integrates UML and DSL. This approach is necessary if the tools cannot use a single meta-meta-language. [16] agrees with the approach and describes consistency relationships between tools; this paper also argues for non-centralized data.

MOF and EMF are compared in [17] using a reversible mapping. This paper makes an interesting point that MOF was developed top-down (and therefore from a modeling perspective) and that EMF was developed bottom up (and therefore from a programming perspective).

The XMI format for UML models is analyzed in [18]. The authors conclude that many tool vendors compromise the XMI standard by implementing tool-

specific extensions. This would imply that a good candidate for a universal meta-data interchange format should have some form of extensibility built-in.

Integration involves many forms of data exchange. The paper [19] describes many different types of information exchange where the type definition (schema) is part of the interchange. The analysis is based on information representing programs, however the results are general. The paper analysis the properties of the information exchange under different implementation mechanisms.

Tool integration experiences are described in [20] which suggests that tool integration standards have not been successful. UML is criticized and DSLs are proposed in conjunction with 100% code generation.

Software Factories are described in [21] which is an approach to developing software systems that addresses the scale and scope required by modern development processes. In particular it proposes that software should be developed in terms of tool-chains using model-driven DSL technologies.

Bezin et al. [22] describe an approach to interoperability between two major approaches: MS DSL Tools and EMF. Their approach uses a common meta-meta-language as advocated by KISS and points out many of the practical issues in achieving interoperability that KISS aims to address.

## 11 References

- [1] J. Visser A. van Deursen, P. Klint. Domain-specific languages: An annotated bibliography. ACM SIGPLAN Notices, 35(6):26–36, 2000.
- [2] M. Hernik, J. Herring, and A. Sloane. When and how to develop domain specific languages. ACM Computing Surveys, 37(4):316–344, Dec. 2005.
- [3] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In International Conference on Software Engineering, 2007, Future of Software Engineering, pages 37–54, 2007.
- [4] G. Kiczales. The Art of the Metaobject Protocol. MIT Press, 1991.
- [5] The Eclipse Modelling Project <http://www.eclipse.org/modeling/>.

- [6] S. Cook and G Jones and S Kent and A Wills. Domain-specific development with visual studio DSL tools. Addison-Wesley Professional. 2007.
- [7] J. Bettin. IT Standards and Beyond: managing commodity products and services. SDA Asia, Vol 9 2006. An updated version is available from <http://www.industrialized-software.org/it-standards:managing-commodity-products-and-services>.
- [8] J. Bettin. Different Shades of Grey of Vendor Lock-In. IBRS research paper, 28 June 2006. Available to IBRS customers from <http://www.ibrs.com.au>.
- [9] A. Kleppe. Software Language Engineering: Creating Domain-Specific Languages Using Metamodels. Addison-Wesley, 2008.
- [10] Mike Wicks. Tool Integration in Software Engineering: The State of the Art in 2004. Available from <http://www.macs.hw.ac.uk:8080/techreps/docs/files/W-MACS-TR-0021.pdf> and <http://www.macs.hw.ac.uk:8080/techreps/docs/files/H-W-MACS-TR-0041.pdf>.
- [11] Cornelia Boldyreff, et al. Environments to support collaborative software engineering. In Cooperative Methods and Tools for Distributed Software Processes, vol 380.222 of RCOST / Software Technology Series. 2003.
- [12] J Amsden. Levels of Integration. Object Technology International, 2001. <http://www.eclipse.org/articles/Article-Levels-Of-Integration/Levels-Of-Integration.html>.
- [13] A W Brown. Control integration through message-passing in a software development environment. Software Engineering Journal, 8(3):121-131, May 1993.
- [14] Jad El-khoury, Ola Redell, and Martin Törngren. A tool integration platform for multi-disciplinary development. In EUROMICRO-SEAA, pages 442-450. IEEE Computer Society, 2005.
- [15] Sven Burmester et al. Tool integration at the meta-model level within the fujaba tool suite In Proc. of the Workshop on Tool-Integration in System Development. 2003.
- [16] R Freude and A Königs. Tool integration with consistency relations and their visualization. In the Workshop on Tool Integration in System Development (TIS 2003), 2003. the 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003).
- [17] Anna Gerber and Kerry Raymond. MOF to EMF: there and back again. In Eclipse '03: Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange, pages 60-64, New York, NY, USA, 2003. ACM Press.
- [18] Juanjuan Jiang and Tarja Systä. Exploring differences in exchange formats - tool support and case studies. In CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering, page 389, Washington, DC, USA, 2003. IEEE Computer Society.
- [19] D Jin, J R Cordy, and T R Dean. Where's the schema? A taxonomy of patterns for software exchange. In 10th International Workshop on Program Comprehension, pages 65-74, 2002.
- [20] S Kelly. Improving the integration of a domain-specific modelling tool. In the Workshop on Tool Integration in System Development (TIS 2003), 2003. the 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003).
- [21] J. Greenfield, K. Short. Software factories: assembling applications with patterns, models, frameworks and tools. Conference on Object Oriented Programming Systems Languages and Applications, 2003.
- [22] Bézivin, J., Hillairet, G., Jouault, F., Kurtev, I., Piers, W. Bridging the MS/DSL Tools and the eclipse EMF Framework. OOPSLA Workshop on Software Factories. 2005.