A Model-Driven Engineering Framework for Constrained Model Search

Mathias Kleiner

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

A Model-Driven Engineering framework for Constrained Model Search

Mathias Kleiner

N° 6982

Juillet 2009

Thème COM

Rapport de recherche

A Model-Driven Engineering framework for Constrained Model Search

Mathias Kleiner

Abstract: This document describes a formalization, a solver-independant methodology and implementation alternatives for realizing constrained model search in a model-driven engineering framework. The proposed approach combines model-driven engineering tools ((meta)model transformations, models to text, text to models) and constraint programming techniques. Based on previous research, motivations to model search are first introduced together with objectives and background context. A theory of model search is then presented, and a methodology is proposed that details the different involved tasks. Concerning implementation, three constraint programming paradigms are envisionned and discussed. An open-source implementation based on the relationnal language Alloy is described and available for download.

Key-words: Model-driven engineering, constraints, finite models, metamodels, relational languages

Un cadre d'ingénierie des modèles pour la recherche contrainte de modèles

Résumé : Ce document décrit une formalisation, une méthodologie (indépendante du moteur de résolution) et des alternatives d'implémentation pour effectuer de la recherche contrainte de modèles dans un cadre d'ingénierie des modèles. L'approche proposée combine les outils de l'ingénierie des modèles (transformation de (méta)modèles, modèle vers texte, texte vers modèles) avec des techniques de la programmation par contraintes. Sur la base de recherches précédentes, les motivations, objectifs et le contexte de la recherche de modèles sont introduits. Une théorie est présentée, ainsi qu'une méthodologie détaillant les différentes opérations nécessaires. En ce qui concerne l'implémentation, trois paradigmes de la programmation par contraintes sont envisagés et discutés. Une implémentation opensource basée sur le langage relationnel Alloy est décrite et disponible en téléchargement.

Mots-clés : ingénierie des modèles, contraintes, modèles finis, métamodèles, langages relationnels

Contents

1   Context and Objectives

The AtlanMod team recently demonstrated [12] how constraint resolution techniques can be integrated into model-based software architectures in order to realize combinatorial transformations. The proposed approached[1] combined model-driven open-source tools (ATL) to an advanced constraint programming technique called configuration. The experiments were conducted using a proprietary configuration tool (JConfigurator). The objective of this research report is three-fold: formalize the theory of the approach, present a solver-independant methodology for realizing constrained model search, and study the implementation and the differences between different open-source constraint solvers.

1.1   Brief introduction to MDE and model transformation

Model Driven Engineering is an emerging research area that considers the main software artifacts as typed graphs. This comes from an industrial need to have a regular and homogeneous organization where different facets of a software system may be easily separated or combined. The basic assumption of MDE is that the classical programming code is often not the right representation level for managing all these facets even if, at some point of the process, executable code will usually be generated from some abstract representation level.

In MDE, models are considered as the unifying concept. Traditionally, models have often been used as initial design sketches mainly aimed for communicating ideas among developers. On the contrary MDE promotes models to primary and precise artifacts that drive the whole development process. The notion of model goes beyond the narrow view of semi-formal diagram thus requiring much more precise definitions and implementations that will allow partial or full automation. The MDE community has been using the concepts of terminal model, metamodel, and metametamodel for quite some time. A terminal model is a representation of a system. It captures some characteristics of the system and provides knowledge about it. MDE tools act on terminal models expressed in precise modeling languages. The abstract syntax of a modeling language, when expressed as a model, is called a metamodel. A language definition is given by an abstract syntax (a metamodel), one or more concrete syntaxes, and a definition of its semantics. The relation between a model expressed in a language and the metamodel of this language is called conformsTo. This should not be confused with the representationOf relation holding between a terminal model and the system it represents. Metamodels are in turn expressed in a modeling language called metamodeling language. Its conceptual foundation is itself captured in a model called metametamodel. Terminal models, metamodels, and metametamodel form a three-level architecture with levels respectively named M1, M2, and M3. A formal definition of these concepts may be found in [7]. The principles of MDE may be implemented in several standards. For example, OMG proposes a standard metametamodel called Meta Object Facility (MOF).

---

[1]source code is available as an ATL usecase at http://www.eclipse.org/m2m/atl/

The main way to automate MDE is by providing transformation facilities. The production of model Mb from model Ma by a transformation Mt is called a model transformation. When the source and target metamodels are identical ($MMa = MMb$), we say that the transformation is endogenous. When this is not the case we say the transformation is exogenous. A model transformation may itself be considered as a model. This means that a transformation program Mt conforms to a metamodel MMt. The consequences are quite important since this allows for example uniformly storing and retrieving different kinds of terminal models including transformations. Beside storage and retrieval, many other common operations may also be applied to such different kinds of models. In this work we use ATL (AtlanMod Transformation Language), a QVT-like model transformation language [9] allowing a declarative expression of a transformation by a set of rules.

1.1.1 Definitions

We use in this report the definitions introduced in [7]:

Definition 1 (model) *A model M is a triple $< G, \omega, \mu >$ where:*

- *G is a directed multigraph,*

- *ω is a model (called the reference model of M) associated to a graph G_ω*

- *μ is a function associating nodes and edges of G to nodes of G_ω*

Definition 2 (conformsTo) *The relation between a model and its reference model is called conformance and noted $conformsTo$ (or abbreviated C2).*

Definition 3 (metametamodel) *A metametamodel is a model that is its own reference model (i.e.it conforms to itself).*

Definition 4 (metamodel) *A metamodel is a model such that its reference model is a metametamodel.*

Definition 5 (terminal model) *A terminal model is a model such that its reference model is a metamodel.*

1.2 Constrained metamodels

The notion of constraints is closely tighted to MDE. Engineers have been using constraints to complete the definition of metamodels for a long time, as illustrated by the popular combination UML/OCL. Constraints can be, for instance, checked against one given to model in order to validate it. We propose the following definitions:

Definition 6 *A constrained metamodel CMM is a pair $< MM, C >$ where MM is a metamodel and C is a model representing a set (a conjunction) of predicates over elements of the graph associated to MM. C is an oracle that, given a model $M = < G, MM, \mu >$, returns true (noted $CMM(M)$) iff M satisfies all the predicates.*

Definition 7 *A model M conformsTo a constrained metamodel CMM if and only if $CMM(M)$.*

Technically, many languages can be used to model constraints, with different levels of expressiveness. OCL is a popular language, widely used in the industry. OCL supports operators on sets and relations as well as quantifiers (universal and existential) and iterators. In this report, we will be using the ICFG language [5]. ICFG is an adapted version of OCL that focuses on metamodel static constraints. ICFG is itself defined by a metamodel (available as KM3 and ECORE) and a parser (generated with TCS [8]).

1.3   Brief introduction to constraint programming

Constraint programming (CP) is a declarative programming technique, where constraints play a central role, to solve combinatorial (usually NP-hard) problems. A constraint, in its wider sense, is a predicate on elements (usually represented by variables). A CP problem is thus defined by a set of elements and a set of constraints. The objective of a CP solver is to find an assignment (i.e a set of values for the variables) that satisfy all the constraints. There are several CP formalisms and techniques which mainly differ by their expressiveness, the abstractness of the language and the solving algorithms. In this report we will focus on the language part, i.e what kind of elements and constraints can be represented and reasoned about. In order to narrow the scope, we will briefly present three important CP formalisms: SAT (boolean SATisfiability problem), CSP (Constraint Satisfaction Problem), and object-oriented configuration.

1.3.1   The SAT formalism

SAT problem is to decide if, for a given boolean formula, each boolean variable can be given an assignment such that the formula evaluates to true. SAT is known as being a NP-complete problem[2].

Definition 8 (SAT instance) *A SAT instance $\mathcal{S}$ is defined by $\mathcal{S} = (\mathcal{X}, \mathcal{C})$ where $\mathcal{X}$ is a set of boolean variables and $\mathcal{C}$ is a set of clauses. A clause is a finite disjunction of literals and a literal is either a variable or its negation.*

Alloy   Alloy [6] is a relational language that offers automatic compilation to SAT problems. The proposed language, that can be seen as a subset of the Z language, allows for expressing complex predicates using atoms (undividable elements), sets (of atoms), relations, quantifiers (universal or existential), operators for relations traversal, etc. However, due to the properties of SAT problems, Alloy cannot be considered as a true first-order logic solver. Indeed, to be able to translate the problem into SAT, a *scope* needs to be given to each set, that limits the number of atoms that can be contained in the set.

1.3.2   The CSP formalism

CSP extends SAT in that it does not restrict variable domains to binary values.

Definition 9 (CSP instance) *A CSP instance is well-defined by a triplet $< X, D, C > :$*

- *X is a finite set of variables $X_1, ..., X_n$*

- *D is a finite set of domains $D_1, ..., D_n$ where D_i is a set of possible values for X_i*

- *C is a finite set of constraints where each constraint is an assertion on a subset of $X = X_j, ... X_k$ defined by a subset of $D_j, ..., D_k$*

Solving a CSP consists in assigning a value V_i of the domain D_i to each variable X_i such that it satisfies all the constraints in C.

1.3.3 Object-oriented configuration

Configuring is the task of composing a complex system out of generic components [10]. Components, also called objects or model elements, are defined by their types, attributes, known mutual relations and predicates over those elements. The acceptable systems are further constrained by the request: a set of problem-specific and/or user-specific requirements, represented by a fragment of the desired system (i.e interconnected objects). From a knowledge representation perspective, configuration can be viewed as the problem of finding a graph (i.e a set of connected objects) obeying the restrictions of an object model under constraints.

Various formalisms or technical approaches have been proposed to handle configuration problems: extensions of the Constraint Satisfaction Problem paradigm [14, 19, 16], knowledge-based approaches [18], logic programming [17], object-oriented approaches [13, 11]. Configuration has traditionnaly been used with success in a number of industry applications such as manufacturing or software engineering. More recently, the expressive power of configuration formalisms has proven their usefulness for artificial intelligence tasks such as language parsing [3]. A deeper introduction to configuration can be found in [10].

Definition 10 (Configuration model) *A configuration model is well defined as a set of types, attributes, ports (i.e unidirectional relations) and constraints. We define a configuration model with a quintuplet $< T, A, P, D, C > :$*

- *T is a finite set of types*

- *A is a finite set of attributes*

- *P is a finite set of ports*

- *each type $t \in T$ has a finite set $subtypes(t) \in T$, a finite set $attributes(t) \in A$ and a finite set $ports(t) \in P$*

- *each attribute $a \in attributes(t)$ has a finite domain $D(a, t)$ (domain types include booleans, integers, floats and enumeration of strings)*

- *each port $p \in ports(t)$ has a target type $targettype(t,p) \in T$, a minimum cardinality $card_{min}(t,p)$ and optionally a maximum cardinality $card_{max}(t,p)$*

- *each constraint $c \in C$ is a predicate on elements of T, A et P.*

Solving a configuration problem consists in generating a set of elements E having a type $type(E) \in T$ et satisfying all constraints. In our context, le source model of the transformation defines a subset of E that must belong to the solution.

2 Model search: reasoning on constrained metamodels

In this work we claim that usual deterministic rule-based model transformations are not sufficient for a large set of applications. This need is directly illustrated in [12], where it is shown that controlled natural language parsing requires non-deterministic techniques. Furthermore, the article presents how a CP-based technique (configuration) can realize such combinatorial transformations. In this report we formalize these transformations as *model search*, present a solver-independant methodology and discuss implementation alternatives based on different CP techniques, languages and tools.

2.1 Relaxed metamodels and partial models

In order to formally define model search, we first define a set of notions that relate to constrained metamodels.

Definition 11 (Relaxed metamodel) *Let $CMM = < MM, C >$ be a constrained metamodel. $CMM' = < MM', C' >$ is a relaxed metamodel of CMM (noted $CMM' \in Rx(CMM)$) if and only if $G_{MM'} \subset G_{MM}$ and $C' \subset C$.*

In other words, a (minimal) relaxed metamodel can be obtained by the removal of all constraints: minimum cardinalities are set to zero, attributes are optionals and predicates are removed. Computing such a relaxed metamodel can obviously be done easily with existing (meta)model transformation techniques.

Definition 12 (Partial model, p-conformsTo) *Let $CMM = < MM, C >$ be a constrained metamodel and M a model. M p-conformsTo CMM if and only if it conforms to a metamodel CMM' such that CMM' is a relaxed metamodel of CMM ($CMM' \in Rx(CMM)$). M is called a partial model of CMM.*

2.2 Model search

Definition 13 (Model search) *Let $CMM = < MM, C >$ be a constrained metamodel, and $Mr = < Gr, MM, \mu_r >$ a partial model of CMM. Model search is the task of finding a (finite) model $M = < G, MM, \mu >$ such that $Gr \subset G$, $\mu_r \subset \mu$, and M conformsTo CMM.*
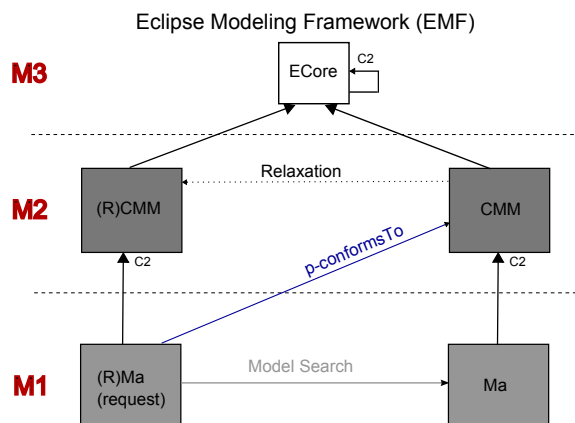
Figure 1: Model search

This model operation task is illustrated in Figure 1. In other words, we consider model search as a model transformation where the source (metamodel and model) is an instance of a non-deterministic (combinatorial) problem and the target model is a solution (if any exists). From the CP point of view, the target metamodel acts as the constraint model whereas the source model (the request) is a given partial assignment that needs to be extended. In CP, a problem does not necessarily introduce a partial assignment. However, in such particular cases, which from the MDE point of view corresponds to an empty source model, it is always possible to artifically add a root element to the metamodel and define the request being only an instance of this root element.

Sidenote on general transformations Model search can be extended to a general transformation scheme(i.e with different source and target metamodels). Although it is out of the scope of this report, we may quickly sketch the process. The main idea is to realize the union of both (source and target) metamodels, over which the transformation is then defined as a set of relations and constraints between both metamodel elements. The source model therefore $p - conformsTo$ to a relaxed version of this unified metamodel (obviously, since it conformsTo the subpart of it corresponding to the source metamodel). Model search on this unified metamodel, by extending the source model, will produce a model which contains both the source and the target model and from which the target model can be isolated. Such a method would fall clearly in the relational approaches to model transformation.

3   A solver-independant methodology for model search

The goal of model search is to generate a complete and valid model Ma of a constrained meta-model $CMMa$ out of a partial (possibly empty) model $(R)Ma$ (the *request*). Figure 2 illustrates the whole process in a model-driven engineering framework. This process is composed of 6 main tasks.
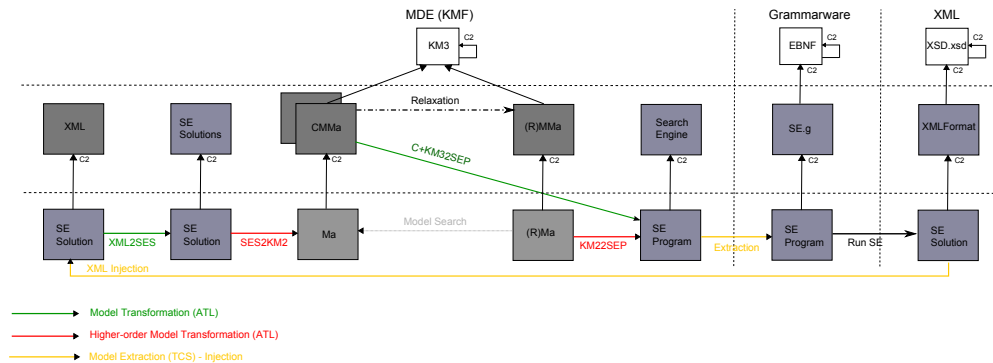

Figure 2: Model search methodology

The first task, realized by the CKM32SEP transformation, is to express the constrained metamodel as a model of the search engine language metamodel. The Figure is a simplified view of the transformation process since there are actually two source models (represented by the doubled square) to the transformation. Figure 3 illustrates the real process and its simplified view.

The difficulty of expressing a constrained metamodel in the search engine language is highly dependant on the abstractness and basic elements offered by the language. Differences between search engines and implementation issues will be throughly discussed in Section 4.

The second task, realized by the KM22SEP transformation, is to express the partial model as a model of the search engine language. Most of the search engines do not differentiate the problem and a partial solution of this problem: they are expressed all together using the same language. Since we want to define this transformation only once, i.e we do not wish to write a transformation for each metamodel MMa, a higher-order transformation (HOT) is required. The HOT takes MMa as source and generates a transformation from MMa to the search engine metamodel.

The third task, realized by the extraction, is to generate the search engine program from its model. This can be done using TCS [8], an EBNF parser generator integrated into the model-driven engineering framework.

The fourth task is to run the search engine for the generated program. When the search succeeds (i.e there is at least one solution), we obtain a solution model in the search engine
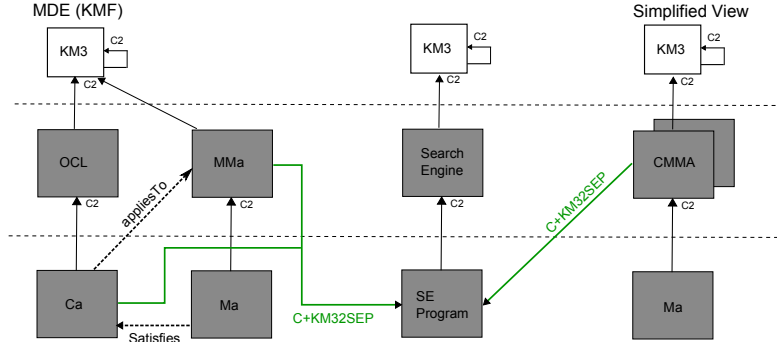
Figure 3: Constrained metamodels simplified view

output format. Here we consider XML since most solvers support this format. This solution is then injected as an XML model. If no XML output is available, the process is still seemingless the same, but a more advanced injector (such as TCS) must be used.

The fifth and sixth task, realized by the XML2SES and SES2KM2 transformation, express the solution as a model of the original metamodel MMa. Although those transformations could be merged, it is natural to decompose the operation into two tasks: expressing the XML model as a model of a metamodel of the search engine solutions, then transform it to a model of MMa. For the same reasons as the KM2SEP transformation, SES2KM2 requires a higher-order transformation. The HOT takes MMa as source and generates a transformation from SES to MMA.

KMF vs EMF The presented methodology assumes the use of KM3 as the metameta-model language. Effective implementations, as the ones described later in this report, are realized using EMF's ECORE language. Since KM3 offers automatic translation to ECORE, the conversion from one framework to another does not introduce any difficulty. Figure 4 illustrates the process.

4 Implementation alternatives

The presented methodology is solver-independant. However a number of difficulties arise with actual implementations. We discuss in the following the main challenges and difficulties that we encountered for each of the three CP formalisms presented in Section 1.3: Object-oriented configuration, CSP, Alloy/SAT. While there are important obstacles to the two first alternatives, we show that the Alloy/SAT solution is well adapted and present a partial implementation.
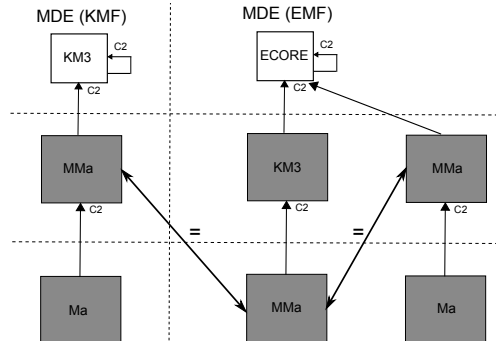
Figure 4: From KMF to EMF

4.1 Implementation with an object-oriented configurator

In [12], we showed that object-oriented configuration is well-adpated to model search. Indeed, the formalism directly contains all the required atomic elements: classes, attributes, relations and an expressive constraint language. Moreover, a tool like JConfigurator[11] provides an API that allows direct access to those elements. However JConfigurator is not an open-source tool and is not anymore maintained by the ILOG company. Very few open-source alternatives exist to JConfigurator that provide the same level of expressiveness and efficiency.

4.1.1 Generic expression of constrained KM3 metamodels

The mapping between KM3 metamodels and the configuration quintuplet $< T, A, P, D, C >$ is straightforward:

- a KM3 class C is mapped to a type $t_C \in T$. C inherited classes are mapped to $subtypes(t_C) \in T$.

- a KM3 attribute A of class C is mapped to $a_A \in attributes(t_C)$

- a KM3 reference R of class C is mapped to $p_R \in ports(t_C)$. The target type of R is mapped to $targettype(t_C, p_R) \in T$, the minimum cardinality to $card_{min}(t_C, p_R)$ and the optional maximum cardinality to $card_{max}(t_C, p_R)$. KM3 references properties, containment and opposite references, can be encoded as predicates in C.

The constraint model that applied to the metamodel can be mapped to elements in C. Obviously, the mapping depends on the constraint metamodel. However, previous studies exhibited only few differences between languages expressivity, such as [4] that maps most of OCL constructs to a configurator.

4.2 Implementation with a CSP solver

Unlike configuration tools, there are several open-source tools based on the CSP formalism. We considered the Choco java library[2] for the implementation with a CSP solver.

The Choco API offers 4 types of discrete variables (booleans, integers, and set domains) as well as a set of logical and arithmetic constraints on those elements. Choco thus lightly extends the CSP formalism by adding the support for set-variables (this is sometimes called a set-CSP problem). It is also possible to define new constraints by specifying their propagation mechanism (how the variables domains are modified after assigning a value to one of them).

4.2.1 Generic expression of constrained KM3 metamodels

A constrained metamodel is composed of two parts: the metamodel and its constraints. The metamodel includes the following elements: classes, attributes and relations. The relations properties (such as containment) can be expressed in the constraints.

Generating Choco variables from metamodels We previously showed how to represent a KM3 metamodel using the configuration quintuplet $< T, A, P, D, C >$. It is possible to associate an element $e \in E$ with a set of Choco variables that represent the quintuplet properties:

- $subtypes(e, t)$ represents the type of the element e. Its domain is a finite set of types $subtypes(t) = t_1, ..., t_m \in T$. We can thus model $subtypes(e, t)$ with an integer variable $e_t \in X$ for which the domain is $D_{subtypes(e,t)} = 1, ..., m$

- each attribute $a \in attributes(e, t)$ has a finite domain $D(a, t)$. We can thus model a with an enumerative variable $e_a \in X$ for which the domain is $D_a = D(a, t)$

- each port $p \in ports(e, t)$ has a domain composed of the set of elements $e_j, ..., e_k \in E$ for which the type belongs to $targettype(t, p)$. We can thus model p with a set-variable $e_p \in X$ for which the domain is $D_p = j, ..., k$

In model search, the set E of solution elements is not bounded: a search engine should generate the elements $e \in E$ during search. Since the set X of Choco variables needs to be finite, it is necessary to bound a set E of candidate elements in order to generate a finite set of Choco variables. A special boolean variable, called e_s, represents the choice of including the element e in the solution.

Let $E = e_1, ..., e_n$ be a finite set of elements. Solving the model search problem with Choco then consists in assigning a value to the set of variables associated to each element: $X(e) = \{e_s, e_t, \bigcup_{a \in attributes(e,t)} e_a, \bigcup_{p \in ports(e,t)} e_p\}$.

We can now define the set $X = \bigcup_{i=1}^{n} X(e)$ of Choco variables and their respective domains D.

---

[2]Choco is being developped at Ecole des Mines de Nantes

Generating Choco constraints We may classify constraints in two categories: those that apply to identified elements of the set E, and those that apply on an *a priori* unknown set of elements satisfyng a given condition (for instance, a universal quantification on elements having a given type).

Since Choco (or more generally CSP solvers) does not allow for the use of universal quantifiers, these constraints need to be *unfolded* on each element of the previously finite set E. However, not all elements of E will belong to the solution, and the implicit semantic of such constraints is that it only applies to solution elements. Therefore each of these constraints must depend on the condition that e_s (the solution variable) evaluates to true. The Choco library does not allow for such conditions in any constraint (in particular it cannot be done in constraints on set-variables). This is also the case for most (if not all) CSP solvers that either do not support the construct or behave very poorly (in terms of computation efficiency) in the presence of such constraints [14].

When it comes to the body of the constraints (outside of quantifiers), the automatic translation also raises the problem of the Choco library (and more generally CSP solvers) expressivity. Indeed the notions of types, attributes and relations being simulated by independant variables, the library does not offer operators that allow for accessing or traversing directly these properties of an element. Automatically translating a constraint including a construct such as "the sum of the values of the (integer) attribute a of each element e_1 connected to element e_2 by the relation p" is a tedious task. In order to simplify the mapping, the notions of elements, types, ports and attributes should be superimposed to the existing library (i.e linked to the variables) and a set of high-level operators on those elements should be available in the constraint language.

The difficulties we pointed out suggest that CSP solvers are not currently well adapted to express constrained metamodels. Although the *a priori* unknown size of model search solutions can be partially circumvented by bounding the candidate elements, the low-level of CSP solvers languages does not turn them into a natural choice.

4.3   Implementation with Alloy/SAT solver

The SAT paradigm shares most of the limits encountered with CSPs since it only offers a finite set of boolean variables and a low-level predicate language (only negation, disjunction and conjunction are supported). However, [6] introduced an expressive relational language with a built-in compilation that allows the use of many recent SAT solvers.

In Alloy, every element is either an atom or a relation. However the proposed language is exclusively based on relations. A set is itself a relation from an atom to the contents of that set (which in turn are also atoms). The main artifacts that we will manipulate in the Alloy language are:

- *Signatures*, declarations of sets, for which the body may contain fields as *relations* to other signatures. Attributes are treated the same as any relation. Scalars, as for signatures, are treated sets of atoms. Signatures also support a form of inheritance.

- *Facts*, declarations of predicates, with quantifiers and an important number of logical, scalar and set operators available.

- *Functions.*

4.3.1 Generic expression of constrained KM3 metamodels

We developed a metamodel of the Alloy language containing the necessary constructs to represent KM3 metamodels and ICFG constraints. Figure 5 shows an overview of the metamodel. The complete metamodel is written in KM3. We also developed a TCS parser generator allowing to inject/extract between the textual version of the language and our metamodel. Both the metamodel and the TCS are freely available, submitted as a TCS usecase (under the form of an Eclipse project), and can be downloaded from [1]. The ICFG metamodel is also written in KM3, the metamodel and the TCS are freely available and can be downloaded from [5]. An overview is presented in Figure 6.
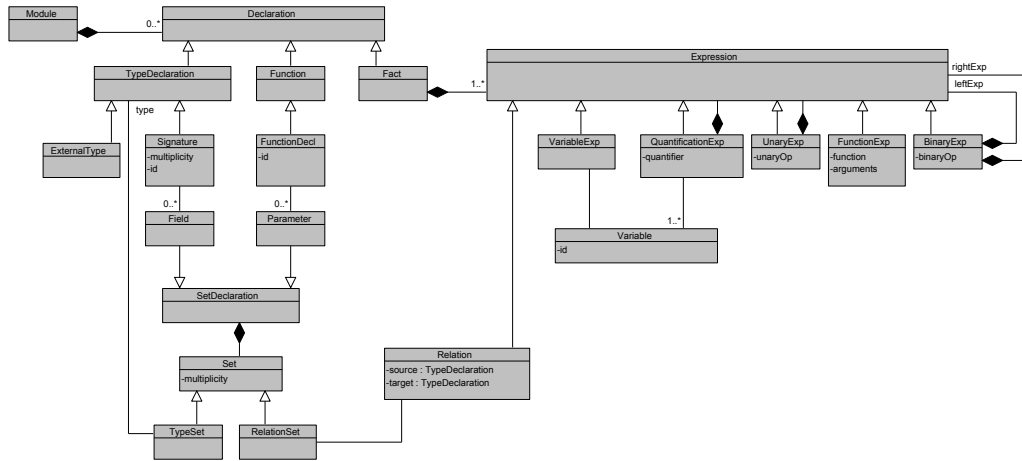

Figure 5: Overview of the Alloy metamodel

On this basis, we defined a mapping from KM3 to Alloy and developped the corresponding ATL transformation. An excerpt of the mapping is presented in Table **??**. In short, KM3 classes are mapped to Alloy signatures, KM3 attributes and references are mapped to Alloy fields, references properties are turned into facts. We also developped an ATL transformation from ICFG to Alloy so as to express metamodel constraints. An informal excerpt of these mappings are presented in Table 1. Both the transformations are merged into a unique transformation using two source models and able to resolve the links between the constraints and the metamodel elements on which they apply. This last transformation
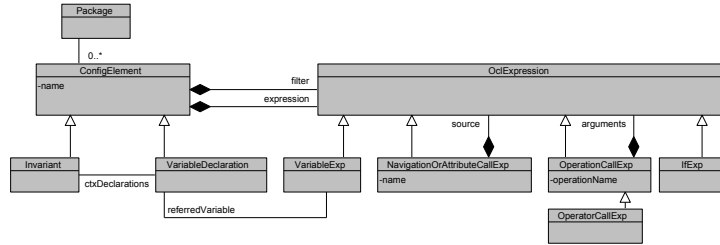
Figure 6: Overview of the ICFG metamodel

KM3 concept	Alloy concept
Metamodel	Module
DataType	ExternalType
Class	Signature
Attribute	Field
Reference	Field
StructuralFeature multiplicity	Quantifier or Fact
Reference containment	Fact
Reference opposite	Fact
ICFG concept	Alloy concept
Invariant	Fact and QuantificationExpression
Invariant declarations	QuantificationExpression variables
VariableDeclaration	Variable
VariableExp	VariableExpression
IfExp	ImpliesExpression
NavigationOrAttributeCallExp	NavigationExpression
OperatorCall	BinaryExpression
OperationCall (size)	SetCardinalityExpression
OperationCall (isIn)	ComparisonExpression
OperationCall (others)	ExternalFunction

Table 1: Excerpt of the mapping from KM3 and ICFG concepts to Alloy concepts

corresponds to the *CKM32SEP* of Figure 2. It is freely available, submitted as an ATL usecase (under the form of an Eclipse project), and can be downloaded from [15].

The whole project is a partial implementation (all but the two high-order transformations) of the model search methodology (presented in Section 3) using Alloy as the search engine. Thanks to its language expressivity, Alloy bridges the gap between constrained

metamodels and low-level languages. As an open-source tool, it becomes a good alternative to configuration tools.

4.3.2 Limitations and future work

The first limitation of this implementation concerns the Alloy metamodel. The metamodel is adapted to work directly with the TCS parser, and as such has some syntactical constructs that would deserve a semantical analysis to completely check the validity of a textual input during model injections. For the same parsing reasons, some of the Alloy metamodel concepts are purely syntactical and may be confusing when developping transformations from/to Alloy. We plan to realize a more fitted metamodel that leaves aside the syntactical constructs that were necessary for the TCS parser, and to develop a two-direction (ATL) transformation between the two metamodels. By transparently chaining the TCS process and the transformation, the syntactical metamodel would not be visible anymore for external users. This notion of syntactical and semantical analysis is well-known in the field of compilation, and we believe that, from a general point of view, it also applies to metamodels and their textual versions.

Future work also includes the development of the missing parts of the Alloy-based implementation of model search, i.e the two high order transformations. One of them allows to transform models to partial instances of the considered Alloy problem (*KM22SEP* in Figure 2). The second one allows to retrieve the Alloy solution and inject it as a model (*SES2KM2* in Figure 2).

5 Conclusion

In recent work, we showed the combinatorial properties of some model transformations. In this report, we presented a theory of those transformations called *model search*, a solver-independant methodology to realize model search in a MDE framework, and discussed different implementation alternatives. Whereas configuration and CSP based alternatives both have drawbacks, the Alloy relational language and its SAT compilation feature bridge an important gap. Based on alloy, we thus developped an open-source partial implementation of model search. Future work includes the full implementation of the methodology and its application to currently known combinatorial transformations.

References

[1] *Alloy usecase: http://www.emn.fr/x-info/atlanmod/index.php/Alloy_ TCS_usecase*, 2009.

[2] Stephen A. Cook. The complexity of theorem-proving procedures. In *STOC*, pages 151–158. ACM, 1971.

[3] Mathieu Estratat and Laurent Henocque. Parsing languages with a configurator. In *Proceedings of the European Conference for Artificial Intelligence ECAI'2004*, pages 591–595, August 2004.

[4] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Configuration knowledge representation using uml/ocl. In *UML*, volume 2460 of *LNCS*, pages 49–62. Springer, 2002.

[5] *ICFG usecase: http://www.emn.fr/x-info/atlanmod/index.php/ICFG_ TCS_usecase*, 2009.

[6] Daniel Jackson. Automating first-order relational logic. In *SIGSOFT FSE*, pages 130–139, 2000.

[7] Frédéric Jouault and Jean Bézivin. Km3: A dsl for metamodel specification. In *FMOODS*, volume 4037 of *LNCS*, pages 171–185. Springer, 2006.

[8] Frédéric Jouault, Jean Bézivin, and Ivan Kurtev. TCS: a DSL for the specification of textual concrete syntaxes in model engineering. In *GPCE*, pages 249–254. ACM, 2006.

[9] Frédéric Jouault and Ivan Kurtev. Transforming Models with ATL. In *MoDELS Satellite Events*, volume 3844 of *LNCS*, pages 128–138. Springer, 2005.

[10] Ulrich Junker. *Configuration*, volume Handbook of Constraint Programming, chapter 26. Elsevier, 2006.

[11] Ulrich Junker and Daniel Mailharro. The logic of (j)configurator : Combining constraint programming with a description logic. In *IJCAI'03*. Springer, 2003.

[12] Mathias Kleiner, Patrick Albert, and Jean Bezivin. Parsing sbvr-based controlled languages. In *Models'09*, 2009. to be published.

[13] Daniel Mailharro. A classification and constraint-based framework for configuration. *AI in Engineering, Design and Manufacturing, (12)*, pages 383–397, 1998.

[14] Sanjay Mittal and Brian Falkenhainer. Dynamic constraint satisfaction problems. In *Proceedings of AAAI-90*, pages 25–32, 1990.

[15] *Model search usecase: http://www.eclipse.org/m2m/atl/*, 2009.

[16] Daniel Sabin and Eugene C. Freuder. Composite constraint satisfaction. In *AI and Manufacturing Research Planning Workshop*, pages 153–161, 1996.

[17] Timo Soininen, Ilkka Niemela, Juha Tiihonen, and Reijo Sulonen. Representing configuration knowledge with weight constraint rules. In *Proceedings of the AAAI Spring Symp. on Answer Set Programming*, pages 195–201, 2001.

[18] Markus Stumptner. An overview of knowledge-based configuration. *AI Communications*, 10(2):111–125, June 1997.

[19] Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. In *Advances in Artificial Intelligence: Proceedings of AI*IA '93*, pages 302–313. Springer, 1993.