Constraint Validation-Based Performance Optimizations in Domain-Specific Modeling Environments

Kényszerellenőrzés alapú teljesítményoptimalizálás szakterület-specifikus modellezőkörnyezetben

by

Tamás Vajk

Supervisor:
Gergely Mezei Ph.D.

Advisors:
László Lengyel Ph.D.
Tihomir Levendovszky Ph.D.

Budapest, 2014.
Ph.D. Thesis Booklet

Tamás Vajk

Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Automation and Applied Informatics

1117 Budapest, Magyar Tudósok krt. 2. Building Q, Room B229.

e-mail: tamas.vajk@aut.bme.hu
tel: +36-1-463-42-25

Supervisor:
Gergely Mezei Ph.D.

Advisors:
László Lengyel Ph.D.
Tihamér Levendovszky Ph.D.
1 Introduction

Software engineering has significantly improved with the use of model-based approaches [BCW12]. Models of programs help in understanding both the overall structure and specific parts of the described software systems, thus, they have played an integral part of the documentation process for a long time. However, with the use of models in design time, the raised abstraction level results in better understandability of the software, and as a consequence better quality applications can be produced. Furthermore, with carefully elaborated software components, it becomes easier to reuse already implemented functionality in different applications, thus, the development time can be significantly reduced [HC01]. With the proliferation of mobile and web applications that should be kept up-to-date in rapid development cycles [Lan12], the reduced development time is a key factor for success.

Software Modeling

One of the most widespread modeling languages is the Unified Modeling Language (UML) [UML05, RJB04, SP99]. UML is a standardized, general-purpose modeling language, which allows mainly visual model definition. UML includes several diagram types to describe both the structure and the behavior of a system. For instance in the former group, there are class, component, and deployment diagrams to specify the system structure for different purposes and on different abstraction levels. The latter group describes the communication patterns between components and the functionality of software systems; notations vary from activity and use case diagrams to sequence diagrams. Due to the facts that UML diagrams cover the whole range of aspects that may be used in a software system, and that it was available quite early as a standardized way of modeling a system, it has many users and provides a de-facto standard in communicating software-related design ideas, solutions between software engineers.

Although UML is widely used in the industry, it has many weaknesses. To overcome the issues, using Domain-Specific Modeling (DSM) [KT08, Kle09, Fow10] approaches have been widely utilized in recent years. Contrary to the general-purpose UML, in DSM, a customized language – called Domain-Specific Modeling Language (DSML or DSL) – is created for the examined problem domain. Thus, the newly created language can be as compact as the abstraction of the problem space allows, while still having all the elements, connections and restrictions specified by the domain. Note that, in DSM, the actual problem is expressed as a model, and a language defines the rules of that model. If the rules are also given as a model, then that model is called a metamodel. This hierarchy closely resembles the one in formal language theory, where production rules specify the language rules, and sentences of the language respect them. And exactly as in formal languages, the metamodel also follows a rule-set. In case of formal languages the notation might be an EBNF notation [Att03], while in DSM, the language of a metamodel might be another customized model, a meta-metamodel, or simply a pre-existing model, such as a UML class diagram. Considering object-oriented programming, the actual domain statements are instances of the defined DSML, thus, they are called instance models of the metamodel. Having models customized to the actual problem domain is not only beneficial because of the compact size of the model, but the precision
in which they describe the problem facilitates powerful tool support. In a flexible DSM tool, each model item can be visually customized to match the notation used by the domain experts, which helps introducing a new development tool into the workflow of a company, and finally the most significant benefit of using DSM tools is that they allow flexible and automatic artifact generation from the models. These artifices might be other model representations, documentation or most notably, output code.

If models of a software system is used not only for documentation purposes, they should be integrated into the development cycle as well. The Model-Driven Engineering (MDE) [BCW12, SVC06] approach places models into the key position in the engineering lifecycle. The system is specified as models, and from them code is generated automatically. MDE can be considered a family of approaches and concepts, Model-Driven Software Development (MDSD, MDD) focuses only on software development, Model-Integrated Computing (MIC) [SK97] focuses on the formal representation, composition, analysis, and manipulation of models during the design process, while Model-Driven Architecture (MDA) [MDA13, KWB03] provides guidelines for integrating different modeling standards by the Object Management Group into the software development process.

Cloud-Based Systems

Cloud computing has received significant attention recently. Companies can store their data and perform their computations off-premise in a highly available and scalable environment, where they only pay for the resources that they actually use. Compared to traditional infrastructures that only use in-house resources, cloud computing has many advantages that have been transforming the computational solutions used at companies. Naturally, already existing on-premise resources can still be used as part of the infrastructure by connecting them to cloud services and forming a hybrid environment [MPF10].

From the service provider’s perspective, companies with over-scaled computation and storage capacity can now profit from utilizing their unexploited resources and offering a cloud computing provider solution. Their previously gained knowledge of building and maintaining a large infrastructure is now being offered as a service to customers. The most widespread cloud computing platforms include Amazon AWS [Ama14], Google App Engine (GAE) [Sev09], The Rackspace Cloud [Rac13], and Microsoft Windows Azure [Dav10]. Cloud providers offer services at different architectural layers [Bru11, Rho09]. The most basic cloud service model – Infrastructure as a Service (IaaS) – offers computers on demand, and the cloud users need to install the operating systems and applications to the machines. In this case, upgrading and maintaining the system is handled by the user. In the Platform as a Service (PaaS) model, these tedious tasks are handled by the provider, who delivers a computing platform whose API can be used in the applications. PaaS providers typically offer different storage solutions that scale automatically with the demand. Finally, in the Software as a Service (SaaS) model, providers operate applications (such as e-mail services) that can be utilized in client solutions.

2 Motivation

The role of model-driven engineering in software development is similar to that of automation in factories, the overall reliability and quality of the products can be improved,
high complexity can be handled efficiently, and development time can be reduced. However, MDE can be considered a relatively new concept in software engineering and as such, applying it in different scenarios results in different levels of success. Overall, it is a promising concept in automated software engineering, but needs further elaboration. This thesis strives to contribute to MDE by improving the performance, reliability and functionality of domain-specific environments.

Graphical domain-specific modeling is an excellent way of formulating the structure of a software system on a chosen abstraction level. However, there are restrictions that cannot be efficiently described in a visual language, or they might be expressed in a significantly easier way in a textual representation. Also, in case of model transformations, attribute value changes are hardly expressible without a textual language. As a result, text-based languages play a complementary, but essential part in visual modeling. Although several languages have been integrated into modeling tools, the Object Constraint Language (OCL) [OCL05, WK03], which is part of MDA, is the most widespread, probably because it has been developed as an auxiliary language for DSLs from the beginning.

The focus of this thesis is the examination, improvement and utilization of OCL. Both previous and new technologies, theoretical results, and algorithms have been woven into domain-specific environments to integrate OCL more deeply into the modeling process and as a result to gain better modeling experience. As domain-specific modeling is gaining significant ground in software engineering, the automatically produced software artifacts and even the supporting tools need to perform better, otherwise the industry will not utilize these solutions. In this thesis both design and run-time improvements have been achieved to answer performance issues on industry-size domain models.

As mentioned, performance optimization and appropriate tool support is vital for the spreading of Model-Driven Engineering. Although OCL plays only an auxiliary part in model-based solutions, the language provides a good basis for various modeling features and performance optimization. One of the most useful design-time OCL related features is the OCL to SQL compiler available in the Dresden OCL Toolkit [Dre13]. The design-time improvement introduced in this thesis, however, falls closer to generic compilation techniques, such as the incremental attribute value evaluation in noncircular attribute grammars presented in [Rep82]. Although these generic algorithms are promising, complex analysis procedures, such as name analysis that requires language-specific information, can hardly be expressed in standard attribute grammars; thus, they cannot reach the desired performance improvement. In case of run-time optimizations, the incremental constraint evaluation in [CT06] is one of the most advanced ones. The basic idea is to perform a normal constraint evaluation first on the model, and afterwards - based on the monitored model element modifications - only a minimal set of model elements are validated again. The only drawback of this solution is that it requires a method for tracking the changes in the model, which is usually not possible outside the modeling tool. In this thesis, the proposed run-time performance improvement does not build on model element tracking, thus, the achieved performance gain in the constraint validation programming interface is available for any kind of utilization.

The combination of Model-Driven Engineering and cloud computing is a promising new field of research [BCJ10]. Two distinct aspects can be considered: (i) automating cloud-based application development with MDE techniques, (ii) supporting modeling with cloud technologies. In the former option, the powerful code generation capabilities
of MDE tools are utilized, already existing domain-specific models of systems can be translated into executables in the cloud with a new model-to-code translator. In the latter option, the immense computational power, and storage space given by cloud providers can be exploited by the modeling environments, models and transformations can be moved to the cloud, which helps the spreading of model transformation based solutions in the ubiquitous mobile or embedded systems. Furthermore, new computational techniques, such as highly distributed graph algorithms based on MapReduce [LS10] can aid the processing of large models. In the thesis, we mainly follow the former option, where modeling aids cloud-based solutions.

Cloud-based solutions not only provide new business models with several system abstraction levels, but makes previously rarely used technologies easily adaptable to our systems. The popularized new technologies are mainly related to highly distributed computation and storage systems. The less experience available with these technologies, and the lack of best practices are generally overcome by adapting and reusing practices from other, more understood technologies. The thesis also follows this path by bridging relational database design and non-relational databases. Non-relational (NoSQL) data store is a collective name for a huge number of different storage solutions from simple key-value databases to graph-based ones [Tiw11]. Although the presented results are applicable to NoSQL solutions, the selected data store type is specific to cloud environments. In general, non-relational data stores can be fine-tuned to match the needs of the solved problem, however, in case of cloud services we are faced with a fixed NoSQL variant that is – in order to be applicable to many different scenarios – generic. The enforced restrictions cannot be changed, thus, the systems need to be adapted to match these constraints. One of the questions answered in this thesis is how NoSQL database design can be automated for the selected subset of data stores. The proposed solution utilizes domain-specific modeling and OCL to formalize, analyze and transform the abstraction layer of the data store.

There are several database optimizers, such as the Oracle SQL Tuning Advisor [Ora12] and the SQL Access Advisor [KAP11] that analyze database query frequency and suggest schema optimizations and denormalization options based on them. Although these algorithms are similar to the one introduced in this thesis, they build on SQL features, such as multiple indexes on the same table, that are not available in the examined NoSQL solutions, thus, they cannot be applied directly in the selected environment. Other approaches try to enable SQL functionalities over NoSQL stores such as SimpleSQL [CdSM12] that provides a relational layer over Amazon SimpleDB, but these solutions usually rely on specific features of the underlying NoSQL stores. This thesis also fall into this category as the results are applicable to a specific, but widely used NoSQL type.

3 Summary of Contributions

The scientific results of my research are summarized in three theses that are further divided into subtheses. I have proved the theoretical results by mathematical and engineering methods, and they are illustrated on case studies to prove their practical relevance. The theses are outlined in the following.

- The first contribution focuses on the design-time optimization of an OCL compiler.
In order to decrease the OCL to C# compilation time, incremental semantic analysis and code generation has been elaborated. The approach utilizes the internal data structures of previous builds, and a change log from the integrated development environment. Based on these artifacts, the unmodified parts of the new compilation are not reanalyzed, thus improved performance can be gained. The introduced algorithms are analyzed for correctness and complexity, while the applicability is shown on both OCL to C# and OCL to Prolog compilation.

- The second contribution provides a run-time optimization of the OCL compiler based-on parallelism. The execution semantics of OCL is given with the Communicating Sequential Processes (CSP) formalism. The CSP definitions are translated to equivalent ones that can be executed in a parallel manner. The introduced sequential and parallel CSP versions are simulated to illustrate their applicability in real-world scenarios.

- In the third contribution, the automatic schema proposition for non-relational cloud-based data stores is introduced. Contrary to relational databases, which perform well with arbitrary queries due to index usage and normalization, non-relational stores are highly optimized for the queries they have to support. Based on a fix set of queries defined in OCL, an initial normalized data structure, and a provisioned query load the illustrated algorithms provide the cost-optimal data storage schema.

**Thesis I – Incremental Semantic Analysis and Code Generation**

In this thesis, I have dealt with the performance optimization of the semantic analysis and code generation processes of a compiler. In my approach, I reuse data artifacts from previous compilation cycles, and only the necessary segments of the abstract syntax trees are processed. The process assumes that internal structures of the previous builds are available in memory in later builds, furthermore, the given method utilizes a log of changes performed on the source, which comes from the integrated development environment.

Thesis I is contained by Chapter 3 of the dissertation. Related publications: [46, 43, 36, 38, 41, 37, 24, 22, 44, 45].

The basic definitions of graphs, trees, attributed graphs, abstract syntax trees (AST) are given in the dissertation, the main definitions used in the analysis-related subtheses are the following.

**Definition 1** (Syntax analysis). Syntax analysis on a grammar \( Gr \) is a function: \( \Sigma^* \rightarrow AST \), where if the input \( \Sigma^* \) can be derived from \( S \) over the grammar \( Gr \) (\( S \xrightarrow{Gr} \Sigma^* \)), it returns a non-empty AST, otherwise it returns an empty AST.

**Definition 2** (Attribute evaluation). Attribute evaluation is a function \( AST \rightarrow AST \), in which the attribute values of the target AST have been evaluated. A part of this function evaluates the fixed attribute \( a \) of an attributed vertex \( AV \) (\( eval_a : AV \rightarrow AV \)), thus, \( eval_a(v) \) calculates value \( v_a \).

**Remark 1.** Evaluation types are differentiated in [Muc97, Knu68]:
• Synthesized: \( \text{eval}_a(v) = \text{process}_a(v, \text{children}(v)) \) that is: only the attributes of the child vertices and the current vertex are used during the evaluation,

• Inherited: \( \text{eval}_a(v) = \text{process}_a(v, \text{parent}(v) \cup \text{siblings}(v)) \) that is: attribute values may depend on the attributes of the parent vertex and the siblings additionally to the current attributed vertex,

where \( \text{process}_a : \mathbb{AV} \times 2^{\mathbb{AV}} \rightarrow \mathbb{AV} \) sets the actual attribute value of \( a \in A \) of the attributed vertex \( v \in \mathbb{AV} \).

The incremental code generation algorithm depends on the expressive power of the output language. The following definitions allow differentiating between languages.

Definition 3 (Expression-preserving compilation). An expression-preserving compilation is a total function \( C_{E_{Gr_1}, E_{Gr_2}} : L_{Gr_1} \rightarrow L_{Gr_2} \) that compiles from \( L_{Gr_1} \) to \( L_{Gr_2} \). \( E_{Gr_1} \in N_{Gr_1} \) and \( E_{Gr_2} \in N_{Gr_2} \) are non-terminal symbols representing all the expressions in the given languages. A compilation is expression-preserving if and only if each substring of the input that has its derivation from \( E_{Gr_1} \) can be translated into a substring that is derived from \( E_{Gr_2} \).

Definition 4 (Weak expression-preserving compilation). A weak expression-preserving compilation is a total function \( C_{E_{Gr_1}, E_{Gr_2}, S_{Gr_2}} : L_{Gr_1} \rightarrow L_{Gr_2} \) that compiles from \( L_{Gr_1} \) to \( L_{Gr_2} \). \( E_{Gr_1} \in N_{Gr_1} \) and \( E_{Gr_2} \in N_{Gr_2} \) are non-terminal symbols representing all the expressions in the given languages and \( S_{Gr_2} \in N_{Gr_2} \) non-terminal symbol represents the statements in \( L_{Gr_2} \). A compilation is weak expression-preserving if and only if each substring of the input that has its derivation from \( E_{Gr_1} \) can be translated into an expression in \( L_{Gr_2} \) and an ordered set of statements in \( L_{Gr_2} \), where the compiled statements calculate the value of the generated expression.

Subthesis I.1 – Properties of standard analysis

I have proved that syntax analysis has the following properties

P1. \( \forall \Sigma^* : \) substring of the input if \( \exists \text{OCLExpr} \xrightarrow{*} \Sigma^* : \exists \text{AST} \) generated for \( \Sigma^* \) and \( \text{val}(\text{VertexType}, \text{AST}) = \text{OCLExpr} \),

P2. if \( \exists \text{OCLExpr}_1 \xrightarrow{*} (\Sigma \cup N)^* \text{OCLExpr}_2 (\Sigma \cup N)^* \) then \( \text{AST}_2 \subseteq \text{AST}_1 \), where \( \text{AST}_1 \) is generated for \( \text{OCLExpr}_1 \) and \( \text{AST}_2 \) is produced for \( \text{OCLExpr}_2 \),

P3. if \( \exists S \xrightarrow{*} (\Sigma \cup N)^* \text{OCLExpr} (\Sigma \cup N)^* \) then \( \text{AST}_2 \subseteq \text{AST}_1 \), where \( \text{AST}_1 \) is generated for \( S \) and \( \text{AST}_2 \) is produced for \( \text{OCLExpr} \),

where \( N \) is the set of non-terminal symbols, \( \Sigma \) is the set of terminal characters in the grammar. \( \text{OCLExpr} \in N \) is the base for all the expressions in the OCL grammar. I have proved that type analysis does not change the AST structure, only ‘Type’ attribute values are evaluated. I have shown that in \( \text{OclGr} \), only synthesized attribute evaluation has to be executed during the ‘Type’ attribute evaluation of \( \text{OCLExpr} \) vertices.
Subthesis I.2 – Properties of incremental analysis

Built onto the results of Subthesis I.1, I have given an incremental compilation algorithm to analyze partly modified OCL abstract syntax trees. I have proved that the introduced algorithm provides the same AST structure as the AST produced by a regular compilation process. I have shown that intact subtrees in the modified tree (AST$'$) do not have to be reanalyzed, and type(AST$'_\text{part}$) = type(AST$\text{part}$). Let $v'_\text{orig}$ be an unchanged vertex in the modified tree (AST$'$), and suppose that there exists a path in AST$'$ from $v'_\text{orig}$ to a modified vertex ($p_1 = v'_\text{orig}$, $p_n = v'_\text{mod}$ and $v'_\text{mod}$ has been modified). I have proved that $v' \in AST'$ has to be reanalyzed only if the type of each vertex in $p$ have changed. I have proved that the asymptotic efficiency of the standard semantic analysis can be estimated by $T_1 b \log L = T_1 L$. Also, I have shown that incremental version can be approximated by $T_1 (l + \log L) + T_2 (l * \log L)$, where $T_1$ and $T_2$ are the time intervals required to process a vertex and to visit a vertex respectively; $L$ is the length of the input, and $l$ is the accumulated length of the modifications. I have also performed average-case analysis on the incremental compilation, that yielded the execution time in $T_1 \left( l + \frac{\log L}{2} \right) + T_2 \left( \frac{\log L}{2} + kb \log l + 2l \right)$. In the formula, $k$ is the number of changes, $b$ is the average number of branches, and $T_1, T_2, L, l$ are as before.

Subthesis I.3 – Incremental code generation

I have given an algorithm for incremental output code generation. I have given an injective transformation from OCL expressions to their C# equivalents to prove that an expression-preserving compilation $C_{\text{OCL, C#}}^{\text{OCL Expr, expr}}$ from OCL to C# can be constructed. Similarly, I have given a constructive proof to shows that a weak expression-preserving compilation $C_{\text{OCL, Prolog}}^{\text{OCL Expr, V, C\#}}$ from OCL to Prolog can be constructed, where $V$ represents the value types available in Prolog (variables and constants) and $C$ stands for the non-terminal of clauses and compound terms.

Thesis II – Parallel Object Constraint Language

In this thesis, I have dealt with the output code optimization of an Object Constraint Language compiler. To improve the performance of model validation, parallel execution has been considered. I have extended the OCL language with parallel notation for compiler directives, thus parallel and sequential code constructs are interchangeable. In order to prove the equivalence of the sequential and parallel evaluation, I have formalized the execution with Communicating Sequential Processes.

Thesis II is presented in Chapter 4 of the dissertation. Related publications: [29, 23, 38, 22, 44].

Definition 5 (Parallel equivalence). Let CSP processes $P_1$ and $P_2$ be parallel equivalent if and only if actions of $P_1$ and $P_2$ perform the same operations on variables. Furthermore, the order of the actions can only be changed if for all parallel processes ($P_i^k$) in process $P_i$, $\text{var}(P_i^1) \cap \text{var}(P_i^k) = \text{acc}(P_i^1) \cap \text{var}(P_i^k) = \text{acc}(P_i^k) \cap \text{var}(P_i^1) = \{ \}$, where acc($X$) defines variables accessible by $X$, and var($X$) marks variables assignable by $X$. 

7
Subthesis II.1 – Formalization of the execution

I have formalized the execution semantics of OCL in sequential and parallel ways with the Communicating Sequential Processes. Both the standard OCL type library and the metamodeled domain-specific model element have been transformed to CSP. I have elaborated a method to handle value and reference types uniformly in CSP. I have proved that the two formalization options of binary operations are parallel equivalent. Also, the equivalence of the sequential and parallel execution semantics of iterators are proved. Furthermore, I have shown that the parallel execution of the generic \texttt{iterate} expression can be parallelized if its body can be separated into a mapping $m$ and a reduction $r$ function, such that $(T_{acc}, r)$ is a commutative monoid and $m$ is independent from $acc$, where $T_{acc}$ is the type of the accumulator.

Subthesis II.2 – Performance

I have analyzed an OCL fragment in the Process Analysis Toolkit [SLDP09] to illustrate that the proposed formalization is applicable in practice. I have shown with measurements that the performance of the generated output significantly depend on the chosen parallelized constructs. The achieved performance gain is shown to be close to the theoretical maximum; constraint evaluation on dual- and quad-core processors resulted in 1.75 and 2.8 times better performance compared to the single core execution.

Thesis III – Automatic NoSQL Schema Optimization

In this thesis, an algorithm that translates relational schemas and selection queries to their NoSQL equivalent has been introduced. The proposed solution can automatically denormalize the initial data structure, and also translate the predefined queries along to meet the new schemas. As an overall result, if all the schema options are generated, the cost-optimal version based on the provisioned query load can be selected.

Thesis III is contained by Chapter 5 of the dissertation. Related publications: [30, 32, 31, 9, 19].

Definition 6 (Inclusion dependency). An inclusion dependency $IND$ over a set of relations $S$ is an expression of the form $R_1[A_1, \ldots, A_m] \subseteq R_2[B_1, \ldots, B_m]$, where $R_1, R_2 \in S$, attributes $A_i$ ($1 \leq i \leq m$) are attributes of $R_1$, and attributes $B_i$ ($1 \leq i \leq m$) are attributes of $R_2$ respectively; and $R[AS]$ marks the projection of $R$ over the set of attributes $AS$.

Although the definition allows multiple attributes being used in inclusion dependencies, in this work, only single-attribute inclusion dependencies are used. In the literature, these are called unary inclusion dependencies [MLP09], and these are the only supported forms of inclusion dependencies in database systems. Namely, a foreign key is a kind of unary inclusion dependency where the target of the constraint happens to be a (candidate) key of the relation.

Definition 7 (NoSQL Relation). A NoSQL relation $R$, is a relation $R(A_1, A_2, \ldots, A_n)$, $n > 0$ extended with single attribute $A_I$ from $R$, along which (exact and range) selection of data can be performed efficiently (without the need of reading all tuples in $R$).
Subthesis III.1 – Algorithm for NoSQL Schema Optimization

I have given an algorithm that generates all possible denormalized versions of an initial relational data schema. I have provided an algorithm that transforms predefined queries to new queries based on the current denormalization step. I have given an algorithm for selecting the indexed column of NoSQL relations to aid query performance. Contradicting indexes in relations have been handled by two alternatives with different storage and query costs. I have illustrated the algorithms on a practical example. I have proved that denormalizing along the inclusion dependencies created in the index selection algorithm does not yield new schema options, thus, sequentially performing firstly the denormalization and secondly the index selection result in all viable schema options.

Subthesis III.2 – Analysis of the Algorithms

I have formalized the denormalization algorithm with graph transformation. On the formalized algorithm I have proved with category theory that the algorithm terminates, if there are no directed loops in the initial schema. I have shown that with loops in the initial schema, the denormalization runs infinitely. I have provided an upper bound to the depth of the denormalization problem-space that resolves the non-termination issue with inclusion dependency loops.

Subthesis III.3 – Cost and Performance Evaluation

I have given an algorithm to calculate the cost of each schema over a defined query load. Both the maximal and the average cost have been evaluated. The calculations have been demonstrated on a practical example. In addition to denormalization, I have considered merging different types of entities into a single table, and special key selection as well, where semantic information is stored in the key. I have shown with measurements that using separate tables for the different types of entities may result in improved performance. Furthermore, I have shown that with appropriate key selection significant execution time improvement can be achieved.

4 Application of the Results

The theoretical results of this research have been realized in the Visual Modeling and Transformation System (VMTS) [VMT13, LLMC05] and have been applied in industrial and R&D projects.

The first contribution achieves design-time performance improvement, the overall modeling experience is improved by reducing the OCL code compilation time. A typical model-based development process is iterative, firstly, the model is created with an initial set of constraints, and then they are adjusted further to fine-tune the system. Thus, small changes are applied to the input OCL code in each iterations, which can be efficiently handled by the proposed incremental compiler.

The second contribution extends the code generation segment of the compiler with parallel programming constructs. This optimization provides a run-time performance gain in the modeling environment as the executed OCL-based validation can exploit the
multiplied processing power of the universally available multi-core processors. In addition to the performance improvement in VMTS, the results of this contribution are also available outside the modeling framework, because the domain-specific API generated from the metamodel is extended with the C# equivalents of the OCL constraints. Thus, the parallel versions of the constraints are available for the user of the API outside the VMTS.

Finally, the third contribution uses VMTS as a development environment, where if the data layer of an application is modeled, an automatic generation process produces a cost optimal and performance efficient data layer that can be used in cloud-based environments. The Windows Azure Table Storage and the Amazon DynamoDB NoSQL storage solutions have been targeted as both of them have similar constructs and restrictions.

5 List of Publications


[29] Tamás Vajk, Zoltán Dávid, Márk Asztalos, Gergely Mezei, and Tihomér Leventovszky. Runtime Model Validation with Parallel Object Constraint Language. In


[40] Tamás Vajk and Gergely Mezei. Sierpinski Triangles with VMTS. In Applications of Graph Transformation 2007 (AGTIVE) - Graph Transformation Tool Contest, Kassel, Germany, 2007.


6 Bibliography


