Abstract

This project is connected to the SMILE project at the University of Agder. In order to deal with the continuously increasing level of complexity in software, we need to use higher levels of abstraction. By using models throughout the development process in language development, we can better handle this complexity. This report deals with modeling Sudoku by SMILE's six aspects of a meta-model: structure, constraints, textual representation, graphical representation, run and transform. Both Eclipse and Visual Studio are used in the development, thus two different solutions are presented. The results are meta-models created in Eclipse and Visual Studio following the SMILE methodology. These meta-models provide an example of what can be achieved by applying SMILE. In addition the report provides some insight into several tools relevant to the area of work as well as a small user guide to some features of the software used.

Preface

The development of software is increasing rapidly, and so is the complexity of new software systems. When creating these advanced systems, it can be very useful to take higher levels of abstractions into use. Models are the solution, and by using models throughout the development process even the most complex systems can become more manageable. The SMILE project coordinated by Prof. Andreas Prinz at the University of Agder aims to provide a methodology and tools for model driven engineering of languages and their semantics. The SMILE projects main purpose for this project is to have Sudoku modeled using the SMILE methodology by describing Sudoku by the aspects of a meta-model/language: structure, constraints, textual representation, graphical representation, run and transformation. The solution will be created in both Visual Studio and Eclipse, both very advanced and popular development environments.

Table of Contents

List of Figures

List of Tables

List of Code examples

1 Introduction

This chapter provides a short introduction to the background and problem area of this project.

1.1 Problem owner

This master thesis is connected to the Semantic Model-based Integrated Language Environment [23] (SMILE) project, which is coordinated by Andreas Prinz at the University of Agder (UiA).

1.2 Background and problem area

Sudoku is a very popular puzzle these days, and has been for some years now. The game of filling in the numbers from 1 to 9 into a 9x9 celled square of 3x3 celled sub-squares is fascinating people all over the world. Both smaller and larger Sudoku's have become available, where letters and /or symbols are included in the game and also much more advanced problems exist with colors, non-square solutions and so on. These small puzzles can be found all over the web, newspapers and magazines; and books and computer games filled with Sudoku's are widely available. By using the SMILE framework and methodology, a Sudoku meta-model can be created by describing Sudoku by structure, constraints, textual representation, graphical representation, transform and execution.

1.3 Motivation

The SMILE project aims to provide an integrated platform where the structure, behavior, representation and constraint aspects of a meta-model/language can be handled. The motivation for this project is to provide the SMILE project with an executable specification of Sudoku which covers the aspects of a meta-model/language: structure, constraints, behavior and representation. I find this project interesting as it differs from projects I have been involved in before, and I consider it to be an excellent opportunity to learn more about meta-modeling as well as learning to use new tools.

1.4 Acknowledgements

I would like to thank Andreas Prinz, my supervisor for all good advice and his extreme patience. I would also like to thank Terje Gjøsæter, co-supervisor, and Merete Skjelten Tveit, for helping me when I needed some extra input.

1.5 Project and report outline

After presenting the project problem statement and some background information, this project will deal with investigating if and/or how Sudoku can be modeled by the aspects of a meta-model/language. Visual Studio and Eclipse with plug-ins will be tested, evaluated and compared to each other.

This report has 6 chapters. Chapter 1 gives an introduction to the project. Chapter 2 provides a detailed problem description while chapter 3 gives thorough background information and literature review. The solutions are given in chapter 4 and a discussion of the results in chapter 5. An overall conclusion is provided in chapter 6.

2 Problem description

This chapter gives a short description of the problem at hand.

2.1 Problem statement

This project aims at modeling the Sudoku puzzle game as an executable specification that covers structure, text representation, graphical representation, constraints, run and transform aspects of a meta-model/language.

2.2 Requirements

- ✓ The result of this project must be an executable Sudoku model/specification that includes essential solving strategies allowing one to solve simple Sudokus.
- ✓ The specification must cover all aspects of a meta-model/language: structure, constraints, textual representation, graphical representation, run and transform.
- ✓ The report should in addition provide some smaller tutorials/user guides to tools installation and usage.

2.3 Delimitations

- ✓ This project aims for a model/specification of Sudoku, not an implementation.
- ✓ The specification does not need to allow Sudoku's that are not of size 9x9 or solve on other terms than integers from 1 to 9.
- ✓ The specification need only solve simple Sudoku's that have one single solution and can be solved by logic.

2.4 Research questions

- ✓ What tools are available to model the Sudoku specification?
  Which tool(s) will be the better alternative(s) for this task if several alternatives exist? The essential point here is to find a tool that can provide the needed functionality while also fulfilling other demands from SMILE, if any.
- ✓ How can Sudoku be described by all the aspects of a meta-model/language? The specification must cover all these aspects; however some might be more thoroughly covered than others.

2.5 Project contribution

This project wants to contribute to the SMILE project by providing an example specification created following the SMILE framework and methodology. SMILE is

interested in this ad hoc solution as it can be used to show what can be achieved by applying the methodology to a concrete example. Therefore it is important that all aspects of a meta-model/language are described.

2.6 Solution strategy

I intend to solve this problem by trying out several different software solutions in my quest to cover all aspects of a meta-model/language for Sudoku. When all modeling is finished I will compare the different tools to each other. In addition I plan to form this report in such a way that a part of it can be used as a light guide to working with some of the tools I have investigated.

3 Background

In order to describe Sudoku by the mentioned aspects, I need to look into meta-modeling and tools that can help me model Sudoku the way I am looking for. This chapter will introduce you to Sudoku and meta-modeling as well as several tools, technologies and standards I will use in the development of this project.

3.1 Sudoku

As mentioned, a Sudoku usually consists of a 9 x 9 celled grid divided into rows, columns and boxes. Some cells might already contain numbers, known as "givens" (see Figure 1). The goal of the game is to fill in the empty cells, one number in each cell, so that each row, column and box contains the numbers 1 to 9 exactly once. The puzzles come in several difficulties depending on the number and/or layout of the givens. In this project I will use field as a common term for row, column and box.

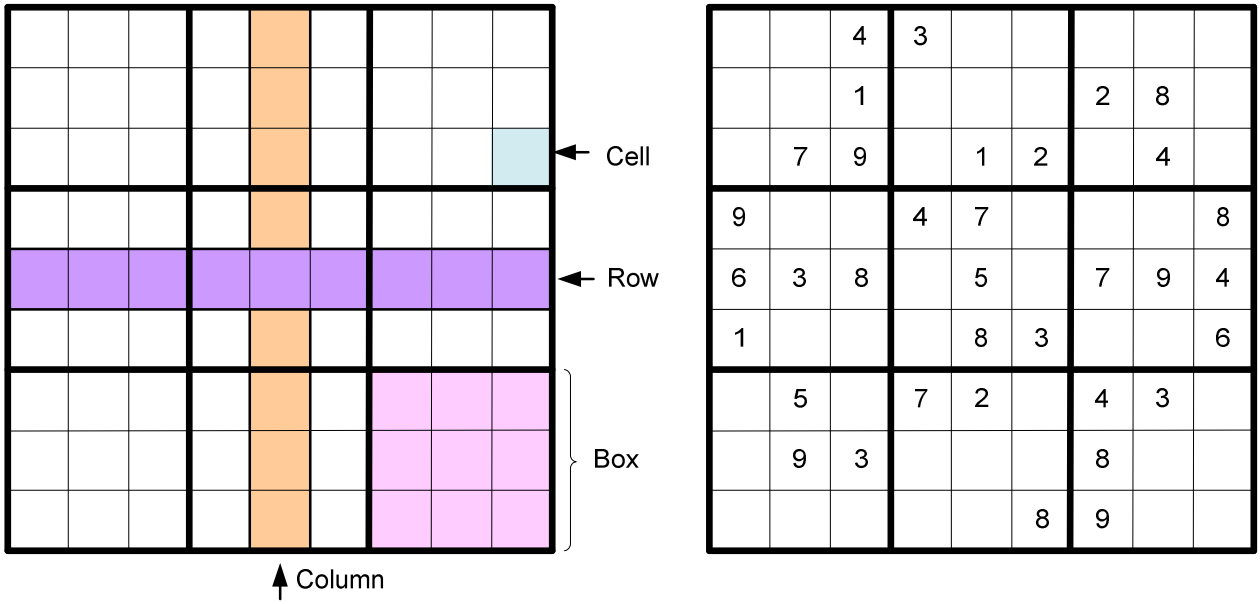

Figure 1 An empty Sudoku (left) and a Sudoku with" givens". Colors only for illustrational purposes.

According to [1] Sudoku has existed for many years, and is based on the magic Latin square of Leonard Euler from 1783. From [7] we learn that the first Sudoku or Number Place (see Figure 2) as it was called (and often still is) in the USA, appeared in May 1979 in an issue of *Dell Pencil Puzzles & Word Games*. The creator was anonymous but later revealed to be Howard Garns. In April 1984 the Number Place puzzle was discovered by Nikoli (see [12]), a Japanese puzzle group. They presented the puzzle to their readers of their puzzle paper Monthly Nikolist and named the puzzle Suuji Wa Dokushin Ni Kagiru, translated to "the number is limited to only a single (unmarried) one".

The name was later abbreviated to Sudoku where Su translates to “number” and Doku stands for “single” by Maki Kaji, the president of Nikoli. He also trademarked the name Sudoku in Japan. This has resulted in many Japanese competing companies calling the puzzle Number Place instead of Sudoku, while in many other countries, e.g. Norway, it is mostly called Sudoku. However, it was after the introduction of Sudoku in the Times newspaper of London in November 2004 that the puzzle became world known and popular. In 1986 Nikoli decided on a rule for making a Sudoku: that the givens must be arranged in a symmetrical pattern. This rule does not apply to the original Number Place.

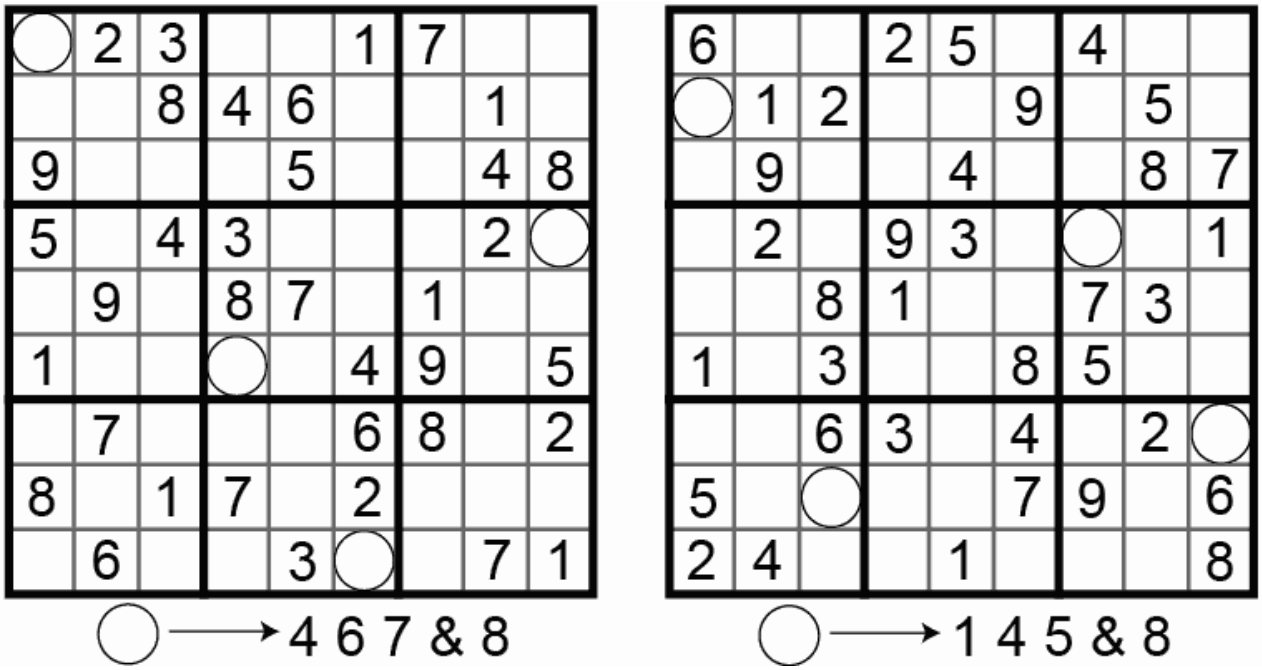

Figure 2 The first number place (Dell Pencil Puzzles & Word Games #16, page 6, 1979-05) [9]

See [8] for a website that provides a large number of Sudoku in a variety of sizes as well as different patterns and color involvement. At [9] several varieties of Sudoku are presented and explained.

3.1.1 The rule and solving strategies of Sudoku

There is essentially just one rule of solving Sudoku; every row, column and box must include the numbers from 1 to 9 exactly once. There are several solving strategies one can use when trying to solve a Sudoku but I’ll only mention a few of them here. I will not implement all of these rules in this project.

Elimination:

Before trying to solve the Sudoku by using the strategies that follow, fill in all possible candidates in each cell. It is very important when a number has been assigned to a cell, that this number is excluded as a candidate from all other cells sharing the same field (see [1]). Performing this elimination will make the process of solving the Sudoku much easier as it is likely to narrow the options for possible values in a cell (see [7]).

Single candidate:

If any number is the only candidate in a cell, this number must be the correct solution for this cell.

Hidden single candidate:

If any number is a candidate in only one cell in a field, this number must be the correct solution for this cell (see [1]).

Naked/matching pairs:

If two cells in the same Field both contain only the numbers x and y, and a third vacant cell in the same row contains the numbers x, y and z, you know that z must be the solution to the last cell. Similar strategies can be applied for triplets, quadruplets and so on (see [1]).

Locked candidates:

Sometimes a candidate within a box is restricted to one row or column. Since one of these cells has to contain that specific candidate, the candidate can be excluded from the remaining cells in that row or column outside of the box [1].

Sometimes a candidate within a row or column is restricted to one box. Since one of these cells has to contain that specific candidate, the candidate can be excluded from the remaining cells in the box (see [1]).

Disjoint subsets:

If some subset of values is constrained to a set of cells of the same size, any other candidates can be eliminated from these cells (see [7]).

3.2 Models and meta-models

Before I look into the different tools and technologies that will be used in this project, I will introduce models and meta-modeling. According to [15] a model is a representation of some subject while Clark, Evans, Sammut and Willans state in [13] that a meta-model is a model of a modeling language and that the meta-model describes a modeling language at a higher level of abstraction than the modeling language itself. A meta-model defines the syntax and semantics of the language it models. For more about meta-levels see 3.4.3.

A domain-specific language (DSL) is a programming language designed to fit a special domain/purpose, in contrast to a general-purpose language like e.g. Java or C# (see [45]). A DSL can be textual or graphical (visual), both types are required in this project. Well known languages that are domain-specific are SQL for database queries and BNF (Backus Naur Form) for syntax specification.

Model Driven Development (MDD) is about using models in the entire software development life cycle.

3.3 SMILE

According to [14] information technology is reaching higher levels of complexity for software, services, and data. In order to handle this complexity we have to use higher levels of abstraction, also called models. SMILE attacks the problem of language development with two novel ideas:

- ✓ To apply MDD to the process of language design and tool development.
- ✓ To extend structural meta-modeling with description facilities for all kinds of language semantics to create a methodology allowing complete descriptions of languages in both structure and semantics.

The overall goal of SMILE is to provide the methodology and tools for model driven engineering of languages and their semantics, and to exploit SMILE in different ICT domains. The main purpose for this project is to have Sudoku modeled using the SMILE framework and methodology by describing Sudoku by the aspects of a meta-model/language (see Figure 3). This is a very vide view of a meta-model, covering more aspects than what we see in e.g. UML.

In [13] the meta-modeling process is described using five steps:

1. Define abstract syntax.
2. Define well-formedness rules and meta-operations.
3. Define concrete syntax.
4. Define semantics.
5. Construct mappings to other languages.

These steps are to a great extent covered by working through the aspects of a meta-model/language as described by SMILE. The abstract syntax (1.) is defined by the structural aspect while well-formedness rules (2.) are defined using constraints. The concrete syntax (3.) is dealt with by both the graphical and textual representation. Constructing mappings (5.) to other languages can be done with transformations. The dynamic semantics (4.) are defined by run/execution (operational semantics) or transformations (denotational semantics).

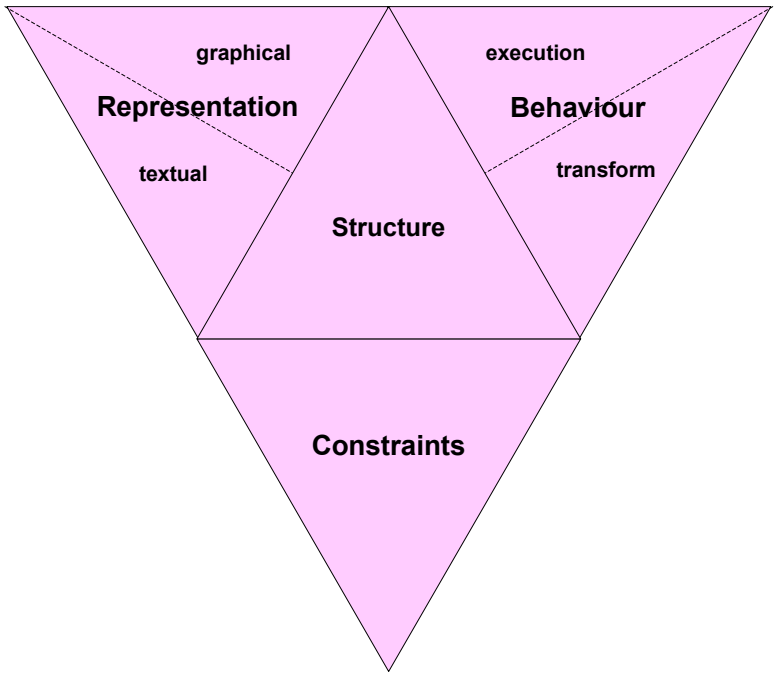

Figure 3 The aspects of a meta-model/language from SMILE

3.3.1 The aspects of a meta-model/language

This section presents and describes the different aspects of a meta-model or language as shown in Figure 3. I aim at describing Sudoku by all of these aspects, but some might be more thoroughly described than others.

Structure: This aspect aims at describing the model by its structure, using e.g. a class diagram. This part includes fairly simple structural properties. More advanced structural issues can often be handled by constraints (see [25]).

Constraints: According to [25] descriptions of constraints can be provided by using e.g. OCL. Constraints can be thought of as additional information to the structural properties. In meta-modeling we can also call these well-formedness rules.

Textual representation: According to [25] the textual grammars are well understood in terms of compiler theory.

Graphical representation: A graphical representation of the model can be presented in several ways, for example by an image or graph.

Run: The run can be a state machine based on the model structure, describing what happens when the model is executed (see [25]). (Obviously the model/specification must be executable for this aspect.)

Transform: This aspect describes how the model can be transformed into some other model type, e.g. from C# to Java or a PIM (Platform Independent Model) to a PSM (Platform Specific Model). The transformation can also have the same source and target model or meta-model depending on the transformation purpose.

3.3.2 SMILE methodology and framework

For the modeling of languages, the SMILE methodology takes three steps (see [24]):

1. The description of the language's structure and semantics.
2. Automatic generation of specific repositories and tools.
3. The use of the generated repositories and tools for concrete models.

The SMILE methodology will be supported by a domain-independent framework that provides language support for information structure and semantic descriptions (see [24]).

3.4 Tools and standards

There are many tools available for use in the modeling tasks of this project and they all have some advantages or drawbacks compared to each other. I will work with Eclipse and relevant plug-ins as well as Microsoft Visual Studio. One issue in general might be that some of these tools do not have many users, especially newer plug-ins for Eclipse. This can make it harder to find online help as well as solid documentation and related work. The following subchapters will introduce the tools and standards I will use in this project.

3.4.1 Object Management Group (OMG)

According to [16] the OMG works with developing standards for a selection of technologies. These standards include e.g. MOF, MDA, UML and OCL that will be described later in this chapter. The OMG's modeling standards enable powerful visual design, execution and maintenance of software. Martin Fowler says in [15] that the OMG was formed to build standards supporting interoperability of object-oriented systems. The OMG has been an international non-profit computer industry consortium since 1989. In this project I will use several of these standards when modeling the different meta-model/language aspects of Sudoku.

3.4.2 Model Driven Architecture (MDA) standard by OMG

OMG's MDA stresses the use of Platform Independent Models (PIM) for specification and transforming these models into Platform Specific Models (PSM). The PIM can be retargeted to different platforms, e.g. Java and .NET. This allows the separation of a system from the way that a system uses the capabilities of its platform (see [21]). The main goals of the MDA are portability, interoperability and reusability.

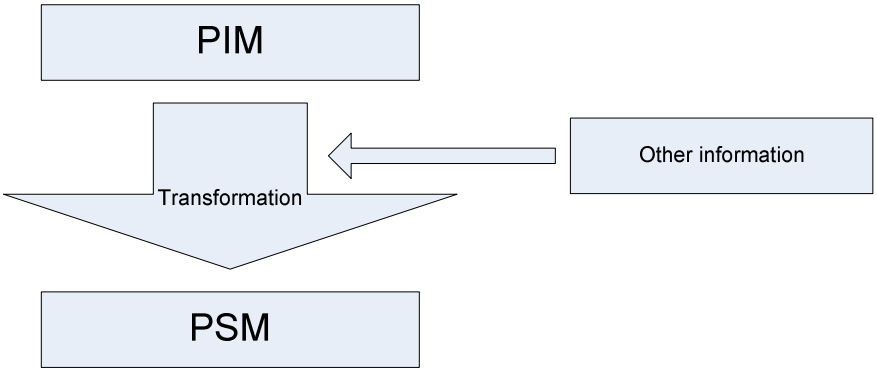

Figure 4 The MDA pattern

Figure 4 shows the MDA pattern. The PIM and other information are combined by the transformation to produce a PSM.

MDA provides an approach for, and enables tools to be provided for:

- ✓ Platform independent system specification.
- ✓ Platform specification.
- ✓ Choosing a particular platform for a system.
- ✓ Transforming the system specification into one that targets a particular platform.

Many OMG standards/technologies enable MDA and these include MOF and UML. An MDA overview is shown in Figure 5.

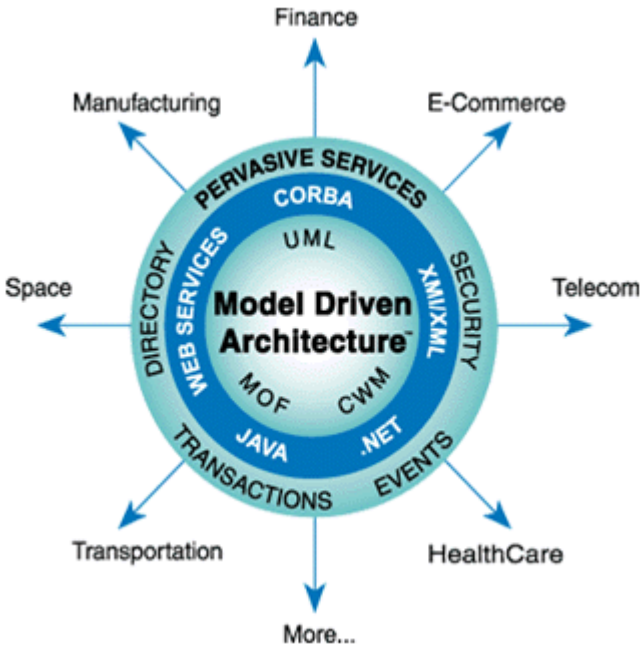

Figure 5 OMG Model Driven Architecture from [22]

3.4.3 Meta Object Facility (MOF) standard by OMG

In a sense MOF can be viewed as a standard for writing meta-models in a more narrow view than I do in this project. According to [13] the traditional meta-model architecture, proposed by the original OMG MOF 1.X standards is based on 4 meta-levels (see Figure 6):

- ✓ **M0** contains the data of the application (user data).
- ✓ **M1** contains the application (model instance/user model).
- ✓ **M2** contains the meta-model that captures the language (meta-model, e.g. UML).
- ✓ **M3** contains the meta-meta-model (MOF).

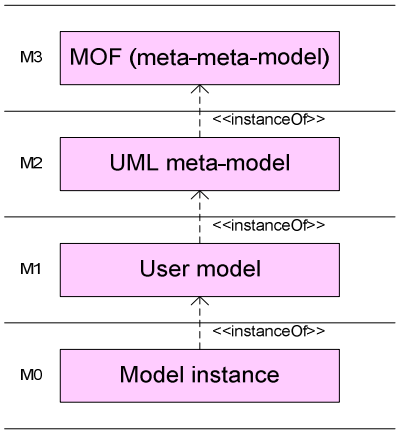

Figure 6 The 4-layer meta-model architecture, MOF 1.4

The concepts used on one level have corresponding descriptions on a next level (the level above, also called meta-level). Stated differently, a level is a model and the level below is an instance of this model. Hence M0 is an instance of M1 that is an instance of M2 and so on. In MOF 2.0 (see [19]) this perceived rigidness of a four-layered architecture is addressed, and we learn that the key modeling concepts are Classifier and Instance / Class and Object and the necessary ability to navigate from an instance (e.g. a model) to its classifier (e.g. meta-model). This means that the MOF2 architecture allows any number of layers greater than two, but please note that a minimum of two layers is mandatory as we must be able to represent and navigate from a model to its meta-model and vice versa (see [19]).

3.4.4 Object Constraint Language (OCL) standard by OMG

The Object Constraint Language (OCL) is a formal language for describing expressions on UML models (see [14]). These expressions typically specify invariants, pre- and post-conditions that must hold for the system being modeled or queries over objects

described in a model. When OCL expressions are evaluated they do not have side effects. This means that their evaluation cannot change the state of the executed system.

3.4.5 Unified Modeling Language (UML) standard by OMG

From [15] we learn that the UML is a visual language for modeling systems through the use of diagrams and supporting text. The UML helps developers specify, visualize, and document models of software systems, including their structure and design (see [17]). The UML 2.0 defines thirteen types of diagrams, divided into three categories:

- ✓ Structure Diagrams (includes the Class Diagram).
- ✓ Behavior Diagrams (includes the Use Case Diagram, Activity Diagram and State Machine Diagram).
- ✓ Interaction Diagrams.

3.4.6 Query/View/Transformation (QVT) standard by OMG

QVT is the OMG standard for model transformations. Three model transformation languages are defined: the Relations and Core languages and Operational Mappings. The QVT specification has a hybrid declarative/imperative nature. The declarative part is separated in a two-leveled architecture (see [20]). The two layers are:

> Relations: a user-friendly meta-model/language which supports complex object pattern matching and object template creation. Traces between model elements being transformed are created implicitly (see [20]).
>
> Core: a meta-model/language that is defined using minimal extensions to EMOF and OCL. It is a small model/language that only supports pattern matching over a flat set of variables by evaluating these variables against a set of models. The Core language is as powerful as the Relations language, but its semantics can be defined in simpler ways. Transformation descriptions in the Core language are more verbose than transformations described by Relations (see [20]) and the trace models must be defined explicitly.

The Relations and Core languages are declarative languages at different levels of abstraction while Operational Mappings is an imperative language that extends Core and Relation. Operational mappings provide OCL extensions with side effects, allowing a more procedural style. In addition to these three languages, it is possible to insert black-

box implementations via MOF operations. These operations may be derived from Relations making it possible to “plug-in” any implementation of such an operation with the same signature. This possibility has several benefits but also a downside. The beneficial features include the possibility of using complex algorithms to be coded in any programming language with a MOF binding, the possibility to use domain-specific libraries and the possibility to allow implementations of some parts of transformations to be opaque. On the downside the plug-in might do arbitrary things to model objects as it has access to object references in the models (see [20]). According to [20] black-box implementations do no have an implicit relationship to Relations, and each black-box must explicitly implement a Relation. The Relation is responsible for keeping traces between model elements related by the Operation implementation. The relationship between the QVT meta-models is displayed in Figure 7.

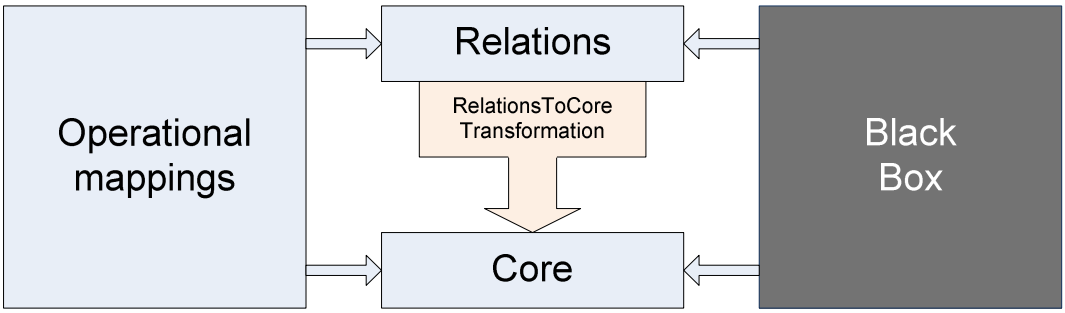

Figure 7 Relationship between QVT meta-models from (see [20])

3.4.7 Eclipse

The Eclipse platform is designed from the ground up for building integrated web and application development tooling. The platform design does not provide a great deal of end user functionality by itself. The value of the platform is what it encourages: rapid development of integrated features based on a plug-in model (see [29]). A plug-in in Eclipse is a component that provides some type of service and/or additional features to the Eclipse workbench.

In this project I will use Eclipse as the main developing environment. Several plug-ins will be used as well, that provide functionality needed for the tasks of this project. These plug-ins will be described in the following subchapters.

3.4.8 Eclipse Modeling Project

The Eclipse Modeling Project focuses on model-based development technologies within the Eclipse community by providing a unified set of modeling frameworks, tooling, and standards implementations. Some components that are part of the Eclipse Modeling Project and will be presented in this chapter are EMF, MDT and GMF.

3.4.9 Eclipse Modeling Framework (EMF)

According to [31] the EMF is a modeling framework and code generation facility for building applications based on structure data models. From a model specification described using XMI (see [30] and [31]), EMF provides both tools and runtime support to produce Java classes for the model, adapter classes that enable viewing and editing of the model as well as a basic editor. Models can be specified using annotated Java, XML documents, or selected modeling tools and then imported into EMF. EMF provides the foundation of interoperability with other EMF-based tools and applications. The EMF uses the Ecore meta-meta-model. An Ecore model can be created in EMF directly using Java or an Ecore editor, or generated from e.g. annotated Java or Rational Rose models. The Ecore model can be used in/by/with oAW/TEF, GEF/GMF, MDT OCL and medini QVT that I will work with later in this project.

3.4.10 MDT OCL (Model Development Tools Object Constraint Language)

According to [35] the MDT project focuses on modeling within the Eclipse Modeling Project (see[38]). Its purpose is to provide an implementation of industry standard meta-models as well as providing exemplary tools for developing models based on those meta-models. The OCL component of the Eclipse MDT project is an implementation of the OCL OMG standard (see [14]) for EMF-based models and provides the following capabilities to support OCL integration:

- ✓ APIs for parsing and evaluating OCL constraints and queries on EMF models.
- ✓ Defines an Ecore implementation of the OCL abstract syntax model, including support for serialization of parsed OCL expressions.
- ✓ A visitor API for analyzing/transforming the AST model of OCL expressions.
- ✓ An extensibility API for clients to customize the parsing and evaluation environments used by the parser.

3.4.11 Graphical Editing Framework (GEF)

According to [32] the Graphical Editing Framework is an open source framework dedicated to providing a rich, consistent graphical editing environment for applications on

the Eclipse Platform. It allows developers to create graphical editors from existing application models. GEF includes and depends on the org.eclipse.draw2d plug-in that provides a layout and rendering toolkit for displaying graphics. The developers can take advantage of the many common operations provided in GEF and extend them for the specific domain if necessary. GEF employs the MVC (model-view-controller) architecture, thus enabling simple changes to be applied to the model from the view. GEF does not help the developer with code generation, thus the amount of work for e.g. a rather small editor can be quite large and demanding.

3.4.12 Graphical Modeling Framework (GMF)

GMF is a part of the Eclipse Modeling Project. According to [34] the GMF provides a generative component and runtime infrastructure allowing the development of graphical editors based on EMF and GEF. GMF employs the Model View Controller (MVC) architecture, thus the model and diagram data are separated allowing simple changes to be applied to the model in the designer (view). The GMF helps the developer by providing wizards as well as generated code, thus even developers who are not very experienced programmers can create simple models.

3.4.13 Omondo EclipseUML

EclipseUML 2007 Europa Studio Edition is an advanced UML solution for Java modelers and developers. There are two available editions of EclipseUML, a Free Edition and the Studio Edition. A table comparing the features of the two editions is available at [39]. The Free Edition includes most features necessary for smaller models, but only allows one developer per project and does not support e.g. CVS, MDA or project documentation. The Studio Edition includes the ability to import EclipseUML Free Edition diagrams, team work support and advanced reverse engineering as well as project documentation.

3.4.14 Textual Editing Framework (TEF)

The TEF (see [40]) allows the developer to create text based editors to her languages. These editors provide an extensive set of modern text editor features such as syntax highlighting, code completion, intelligent navigation, or visualization of occurrences. TEF editors are created by describing your model notation as a set of templates. Each template describes how a model element is to be represented in text. An editor written in TEF is based on a meta-model (e.g. an ecore model) and therefore allows editing of instances for this meta-model. TEF is available as an Eclipse plug-in. The TEF project is still in an experimental phase.

Some features that are or will be part of TEF, from [40]:

- ✓ TEF is based on a template language that provides you with the concepts needed to create a textual notation based on your meta-model. Each template defines how the instances of a meta-model element are to be represented as text.
- ✓ Syntax highlighting meaning you decide which elements are displayed in which color, font, or style.
- ✓ The TEF allows you to define completions that can be applied to certain user defined syntactical constructs. It is possible to provide different completions based on the model element the completion is requested on.
- ✓ You can add constraints to your model and TEF will generate error annotations where these constraints are violated.

3.4.15 medini QVT

medini QVT is a product from ikv++ technologies ag (see [42]) in Berlin. The medini QVT tool implements OMG's QVT Relations specification (but not Core language and Operational Mappings) in a powerful QVT engine. medini QVT is available as a separate Eclipse installation that includes medini QVT along with two examples as well as a plug-in that can be used with your existing Eclipse.

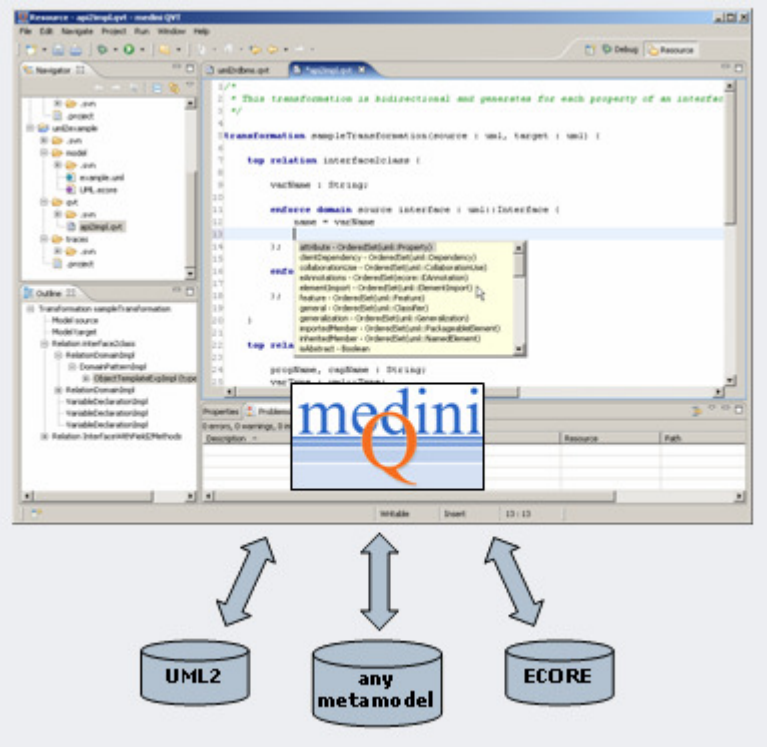

Figure 8 medini QVT screenshot/figure [43]

Some supported features of medini QVT according to [43] are:

- ✓ Execution of QVT transformations expressed in the textual concrete syntax of the OMG MOF 2.0 Relations language.
- ✓ Editor with code assistant.
- ✓ Debugger.
- ✓ Bidirectional transformations.

3.4.16 Microsoft Visual Studio 2005 with VS SDK including DSL Tools

In addition to the Eclipse plug-ins I will also use Visual Studio from Microsoft to develop a DSL for Sudoku. Working with Visual Studio is in many ways very different from working with Eclipse. Visual Studio is a complete set of development tools for building ASP.NET Web applications, XML Web Services, desktop applications, and mobile applications. Several Microsoft programming languages all use the same integrated development environment (IDE), which allows them to share tools and facilitates in the creation of mixed-language solutions (see [48]). I will use the C# language in this project.

In addition, the Visual Studio 2005 SDK (separate download) provides a framework that can be used to extend the functionality of Visual Studio, much like plug-ins do for Eclipse. The SDK includes Domain-Specific Language Tools (DSL Tools) (see [49]), a component that lets the developer to generate graphical designers that are customized for a specific problem. Textual languages are not supported.

3.5 Related work

I was hoping to find similar solutions to what I am about to start in this project. However most of the available work with Sudoku seems to be mathematical articles or pure generators/solvers. Therefore I do unfortunately not have a concrete solution to compare my work with. There are many relevant tools that could have been interesting in the context of this project, and I will present a couple of them in the following subsections.

3.5.1 openArchitectureWare (oAW) 4.2

openArchitectureWare won the 3rd price in the JAX Innovation Award 2007, and oAW 4.2 was released 2007-09-18. According to [53] oAW is an open-source tool platform for model-driven development. oAW supports parsing of arbitrary models, and a language family to check and to transform models as well as to generate code based on models. It has strong support for EMF based models but can work with other models as well. A number of pre-built workflow components can be used for reading and instantiating

models, checking them for constraint violations, transforming them into other models and generating code.

Some Core Features of oAW (see [53])

- ✓ With suitable instantiation, oAW can read any model. Currently oAW provides support for EMF, Eclipse's UML2, several UML tools, textual models, XML and Visio as well as pure::variants variant configuration models.
- ✓ **Check** is an OCL-like language that supports declarative definition of constraints.
- ✓ **Xtend** is a functional model transformation language.
- ✓ **xText** is a framework for creating textual domain specific languages.

3.5.2 Kermeta

The Kermeta workbench is a meta-programming environment that is based on an object oriented DSL optimized for meta-model engineering and is fully integrated with Eclipse. Its features include for example (see [56]):

- ✓ Abstract syntax specification.
- ✓ Static- and dynamic semantics.
- ✓ Model transformation.

Kermeta is built as an extension to EMOF (part of the MOF2).

3.6 Available resources

I expect internet discussion forums to be an important resource during this project. Some of the tools I plan to investigate are still on experimental levels, thus the user groups are probably very small, hence also available documentation and tutorials. I know forums exist for Visual Studio and oAW as well as many Eclipse projects. Forums with active users are very useful as users often encounter similar problems and solutions to these problems are often found in forums. Forums are also useful for issues not covered by documentation. In addition to forums, I plan to use documentation and tutorials that are mostly available on the web as well as scientific articles in my area of work.

3.7 Solution outline

In the process of working on this project I will investigate several tools for working with a language/meta-model for Sudoku. I expect this work to be challenging as there are relatively few people working with some of the problems and software that I will cover in this project. Not all aspect of a meta-model/ language can be covered by all tools; hence this will be different for each tool. I plan to develop the structure using EMF (Ecore) and

UML for the Eclipse-based solution while Microsoft has its own type of structure implementation in Visual Studio. Constraints will be implemented using OCL in Eclipse and C# in Visual Studio. For the textual and graphical representation I will create graphical and textual editors using GMF and TEF. A graphical editor will be created in Visual Studio as well. Transformation will be handled by medini QVT in Eclipse while transformation and execution will be written in C# for the Visual Studio based solution. For execution in the Eclipse based solution I will try to manage this with QVT as well. When my work with all aspects and all tools is complete, I plan to compare the tools to each other, and to provide a discussion for and against each tool.

4 Solution: Sudoku described by the aspects of a meta-model/language

In this chapter I will present the solutions I have worked with during this project by applying the SMILE methodology to create a meta-model of Sudoku. Each aspect is first presented by the expected solution of what we want to express. Then the results achieved using the different tools are presented.

4.1 Structure

In this project the structure consists of `Puzzle`, `Field`, `Row`, `Column`, `Box` and `Cell`. `Puzzle` is the root model class and it contains several `Fields`. A `Field` must have references to the `Cells` that are associated with it. `Row`, `Column` and `Box` extend `Field`. A `Cell` must refer to exactly one referencing `Row`, `Column` and `Box`, while a `Row`, `Column` or `Box` must refer to `iDimension Cells`. A `Cell` must have an integer `iCellValue` that contains its value, while `Puzzle` has an integer `iDimension` keeping its dimension. It is important to have a well formed structure model as this model will be used during the rest of the project. I will also try to keep the models as similar as possible. The ideal Sudoku structure for this project is shown in Figure 9.

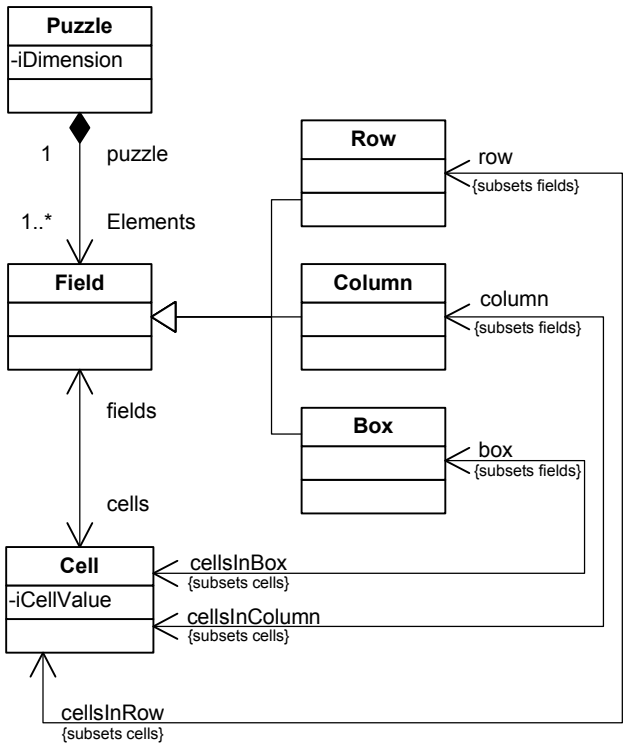

Figure 9 Ideal Sudoku meta-model structure, MOF compliant

4.1.1 EclipseUML

Figure 10 presents the Sudoku structure in a UML Class Diagram. This diagram was created using the Omondo EclipseUML plug-in. It differs from the ideal structure in the sense that operations are automatically added. As you can see from the figure, get and set operations for `iCellValue` are added to the operations list. In addition to this the inheritance of relationships is also missing. The reference between `Row` and `Cell` is different as well; it is a composition type of relationship. This is done to better fit the development in the tools we are to use.

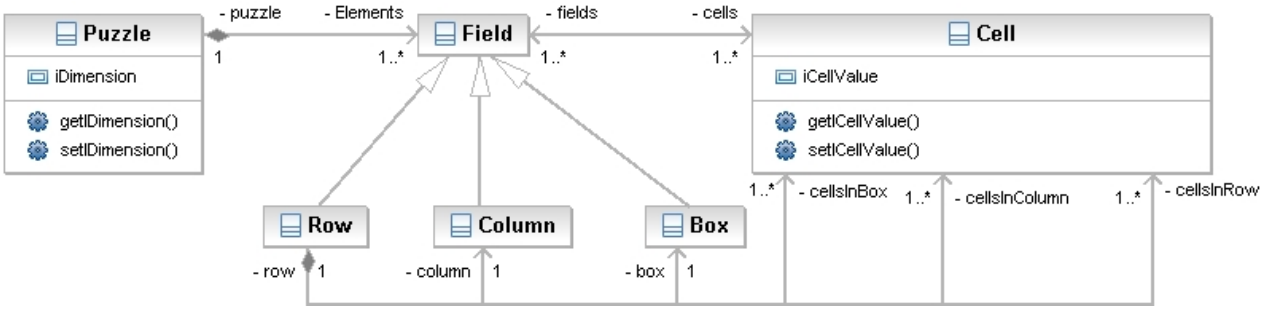

Figure 10 Eclipse: Sudoku structure as a UML Class Diagram created in Eclipse with EclipseUML

4.1.2 Visual Studio 2005 with SDK/DSL Tools

Visual Studio has its own designer for setting up the DSL structure that is somewhat similar to a class diagram, called DomainModel. A DomainModel has DomainClasses and DomainRelationships similar to classes and relationships in UML. Figure 11 and Figure 12 displays the structure diagram from Visual Studio (separated into two images for practical reasons). The lines that are visible from `Row` and `Cell` DomainClasses are relations to these classes' graphical elements and do not influence the structure of the model. The relationships look a bit different from what we are used to from e.g. UML. The DomainRelationship `PuzzleHasElements` is an embedding relationship, much like the UML composition relationship. Attributes can be added to the DomainRelationship. This is automatically added to the DomainModel. The reference between `Row` and `Cell` is also different as it is an embedding type of relationship. This is done to better fit the development in Visual Studio with DSL Tools.

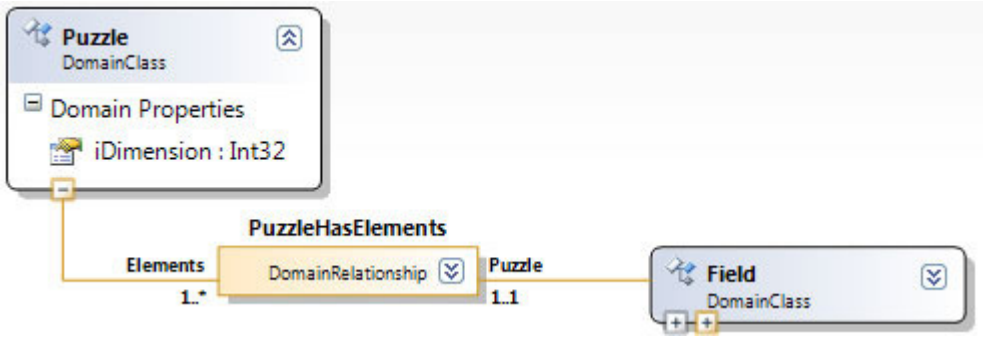

Figure 11 Visual Studio: Sudoku structure part 1

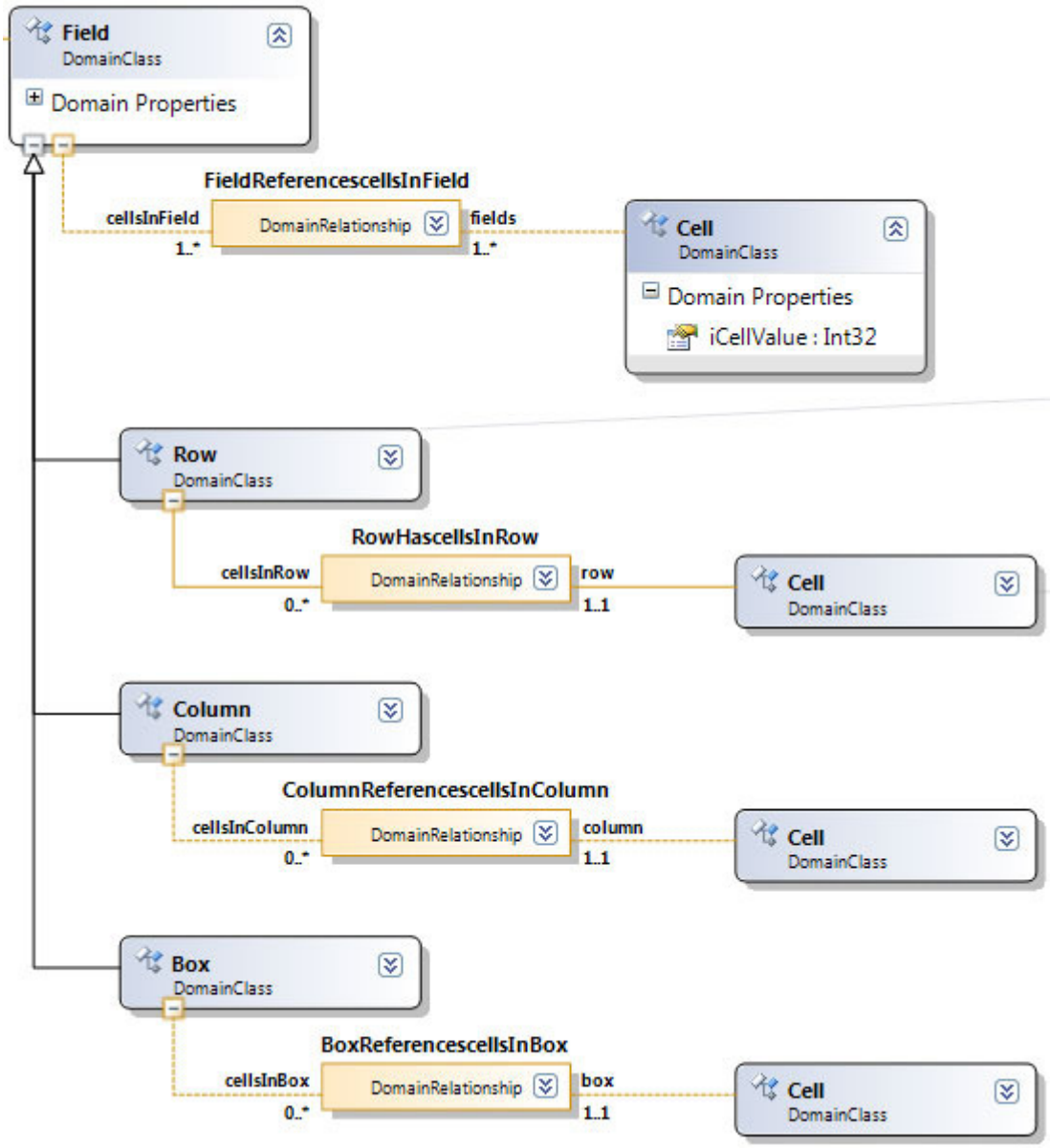

Figure 12 Visual Studio: Sudoku structure part 2

4.2 Constraints

Constraints are very valuable help when it comes to making sure the model is valid at any point. To use the structure as an example, it does not provide all necessary limitations that this project demands. For example I need to make sure that a `Row` can have only exactly `iDimension cells`. OCL can help with this as well as with many other constraints this model needs. In this project, the constraints will be some rules of Sudoku formalized using code. The code language will vary for the different tools. For the Eclipse based solution, OCL is the natural choice for constraints. There is no available function for square root in OCL, a function I need for a couple of the constraints. For illustrational purposes I will use the mathematical symbol for square root, but this will not work in a real program. Visual Studio demands some Microsoft programming language, thus the constraints in Visual Studio will be implemented using C#. The following constraints are valid for a 9x9 Sudoku.

Informal constraints:

Puzzle constraints

- ✓ PuzzleDimension:
  The Puzzle dimension must be 9.
- ✓ PuzzleRow:
  A Puzzle must have as many rows as the puzzle iDimension.
- ✓ PuzzleColumn:
  A Puzzle must have as many columns as the Puzzle iDimension.
- ✓ PuzzleBox:
  A Puzzle must have as many boxes as the Puzzle iDimension.

Field constraints

- ✓ CellsInField:
  A Field must have as many cells as the Puzzle dimension.
- ✓ UniqueValues:
  All Cell.iCellValues in one field must be unique.
- ✓ RowColumnCommonCells:
  A Row and a Column can only have exactly one Cell in common.
- ✓ RowBoxCommonCells:
  A Row and a Box must have either 0 or exactly √iDimension cells in common.
- ✓ ColumnBoxCommonCells:
  A Column and a Box must have either 0 or exactly √iDimension cells in common.

Cell constraints

- ✓ ValueiCellValue:
  The Cell.iCellValue must have a value from 1 to iDimension, or 0 meaning the Cell is empty.
- ✓ CellRowReference:
  Each Cell must only have reference to exactly one single Row.
- ✓ CellColumnReference:
  Each Cell must only have reference to exactly one single Column.
- ✓ CellBoxReference:
  Each Cell must only have reference to exactly one single Box.
- ✓ CommonRowColumn
  Two cells can only have row or column in common.

Logic constraints

Rows(p) ≡ Field(p) ∩ Row

Cols(p) ≡ Field(p) ∩ Column

Boxes(p) ≡ Field(p) ∩ Box

Puzzle constraints

- ✓ PuzzleDimension :
  $\forall p \in Puzzle \bullet iDimension(p) = 9$
- ✓ PuzzleRow:
  $\forall p \in Puzzle \bullet \#Rows(p) = iDimension\ (p)$
- ✓ PuzzleColumn:
  $\forall p \in Puzzle \bullet \#Cols(p) = iDimension\ (p)$
- ✓ PuzzleBox:
  $\forall p \in Puzzle \bullet \#Boxes(p) = iDimension\ (p)$

Field constraints

- ✓ CellsInField:

  $\forall p \in Puzzle \bullet \forall f \in Field(p) \bullet \#cells(f) = iDimension(p)$

- ✓ UniqueValues:

  $\forall p \in Puzzle \bullet \forall f \in Field(p) \bullet \forall cell_1, cell_2 \in f \bullet cell_1 \neq cell_2 \Rightarrow$

  $cell_1.iCellValue \neq cell_2.iCellValue$

- ✓ RowColumnCommonCells:

  $\forall p \in Puzzle \bullet \forall r \in Rows(p) \bullet \forall c \in Cols(p) \bullet \#(cells(r) \cap cells(c)) = 1$

- ✓ RowBoxCommonCells:

  $\forall p \in Puzzle \bullet \forall r \in Rows(p) \bullet \forall b \in Boxes(p) \bullet$

  $\#(cells(r) \cap cells(b)) = \sqrt{iDimension(p)} \vee \#(cells(r) \cap cells(b)) = 0$

- ✓ ColumnBoxCommonCells:

  $\forall p \in Puzzle \bullet \forall c \in Cols(p) \bullet \forall b \in Boxes(p) \bullet$

  $\#(cells(c) \cap cells(b)) = \sqrt{iDimension(p)} \vee \#(cells(c) \cap cells(b)) = 0$

Cell constraints

- ✓ ValueICellValue:

  $\forall cell \in Cell \bullet 0 \leq cell.iCellValue \leq 9$

- ✓ CellRowReference:

  $\forall cell \in Cell \bullet \#row(cell) = 1$

- ✓ CellColumnReference:

  $\forall cell \in Cell \bullet \#column(cell) = 1$

- ✓ CellBoxReference:

  $\forall cell \in Cell \bullet \#box(cell) = 1$

- ✓ CommonRowColumn:

  $\forall p \in Puzzle \bullet \forall f \in Field(p) \bullet \forall cell_1, cell_2 \in f \bullet [cell_1 \neq cell_2 \Rightarrow$

  $row(cell_1) \neq row(cell_2) \vee col(cell_1) \neq col(cell_2)$

OCL constraints:

Constraints on `Puzzle`:

```
context Puzzle inv PuzzleiDimension:
iDimension = 9

context Puzzle inv PuzzleRow:
self.Elements->select(f : Field |  f.oclIsTypeOf(Row))-> size()=iDimension

context Puzzle inv PuzzleColumn:
self.Elements->select(f : Field |  f.oclIsTypeOf(Column))-> size()=iDimension

context Puzzle inv PuzzleBox:
self.Elements->select(f : Field |  f.oclIsTypeOf(Box))-> size()=iDimension
```

Constraints on `Field`:

```
context Field inv UniqueValues:
self.cells->self.cells->isUnique(cell : Cell | cell.iCellValue)

context Field inv CellsInField:
self.cells -> size()= iDimension

context Row inv RowColumnCommonCells:
self.cells -> isUnique(c : Cell | Column.allInstances()->
any(col | col.cells->includes(c)))

context cell inv RowBoxCommonCells:
self.row.cells->intersection(self.box.cells)->size() = 0 or
self.row.cells->intersection(self.box.cells)->size() = √iDimension

context cell inv ColumnBoxCommonCells:
self.column.cells->intersection(self.box.cells)->size() = 0 or
self.row.cells->intersection(self.box.cells)->size() = √iDimension
```

Constraints on `Cell`:

```
context Cell inv ValueiCellValue:
self -> forAll(iCellValue <= iDimension and iCellValue >= 0)

context Cell inv CellRowReference:
self.row -> size()=1

context Cell inv CellColumnReference:
self.column -> size()=1

context Cell inv CellBoxReference:
self.box -> size()=1

context Cell inv CommonRowColumn:
self -> forAll(c: Cell | self<>c implies
(self.row = c.row implies self.column!= c.column))
```

4.2.1 EMF with MDT OCL

The EMF and MDT OCL makes it rather easy to implement OCL constraints in a Java application. It enables you to set the context for your queries and use OCL statements to query your model.

```
// create an OCL instance for Ecore
OCL ocl;
ocl = OCL.newInstance(EcoreEnvironmentFactory.INSTANCE);
// create an OCL helper object
OCLHelper<EClassifier, ?, ?, Constraint> helper = ocl.createOCLHelper();
// set the OCL context classifier
helper.setContext(newstructure.NewstructurePackage.Literals.FIELD);
Constraint uniqueValuesInv =
helper.createInvariant("self.cells->isUnique(cell : Cell | cell.iCellValue)");
Query<EClassifier, EClass, EObject> evaluateuniqueValuesInv =
                                        ocl.createQuery(uniqueValuesInv);
Query<EClassifier, EClass, EObject> evaluatenumberOfCellsInv =
                                        ocl.createQuery(numberOfCellsInv);
if(evaluateuniqueValuesInv.check(field))
{
}
else errormessage += "Field values are not unique \r\n";
```

Code example 1 Eclipse: Constraint code for unique cell values in field

Code example 1 shows an example constraint code for checking that a `Row` has exactly 9 `Cells` and that all `iCellValues` in a `Row` are unique. The constraints are checked for validity by a simple if-else-loop, providing an error message if the query evaluates to false.

MDT OCL differs from the expected solution as it demands a large amount of code in addition to the OCL statements, thus knowing OCL is not enough to actually implement the constraints. This is for example code for setting the context of the query and the query itself. Another problem is how we decide when the constraints are checked, this must also be specified in the code. There is no way to specify the severity of constraint violation, e.g. separating errors, warnings and messages.

4.2.2 Visual Studio 2005 with SDK/DSL Tools

In Visual Studio, constraints are added as validation code in your preferred VS programming language. In this project the constraint code is written in C#. To avoid interfering with the generated code the validation methods are written in separate files that define partial classes (see [46]) where the constraint code is put. This also prevents that custom code is deleted if code is regenerated. I created three different code files for constraints, separating the `Puzzle`, `Row`/`Field` and `Cell` constraints. DSLTools allows returning both messages (informative only), warnings and error messages as well as assigning an error message number to each error. One can also specify when a constraint shall be evaluated. In the code example in Code example 2 the code is excerpted from the `Field` partial class, thus this in the code refers to the `Field` that is being evaluated.

```
///Check that all iCellValues in field are unique
[ValidationMethod(ValidationCategories.Open | ValidationCategories.Save |
ValidationCategories.Menu)]
public void uniqueiCellValues(ValidationContext context){
    try{
        foreach (Cell c in this.cellsInField){
            foreach (Cell cc in this.cellsInField){
                if ((c.Id != cc.Id) && (c.iCellValue == cc.iCellValue)
                                                    && (c.iCellValue != 0))
                context.LogError("Duplicate values in field", "", this,c,cc);
            }
        }
    }
    catch (Exception e){
        context.LogError(e.ToString(), "", this);
    }
}
```

Code example 2 Visual Studio: Constraint code for unique cell values in field

All constraint violations results in the error messages provided in the code. Figure 13 shows an invalid Sudoku in Visual Studio, where `Row` 1 contains two 7's. The error messages are listed below the Sudoku puzzle in the Error List available in Visual Studio. In the same list one can receive messages and warnings. To see what `Cell(s)` violated the constraint, just double click the error message. The `Cells` in question will now appear marked as shown in Figure 13. This is specified in the code like this: context.LogError("Error message here", "Error code here", violatingelement1,violatingelement2);

The Visual Studio constraints solution differs from the expected constraints solution as it can not use OCL or logic constraints, but rely on C# code.

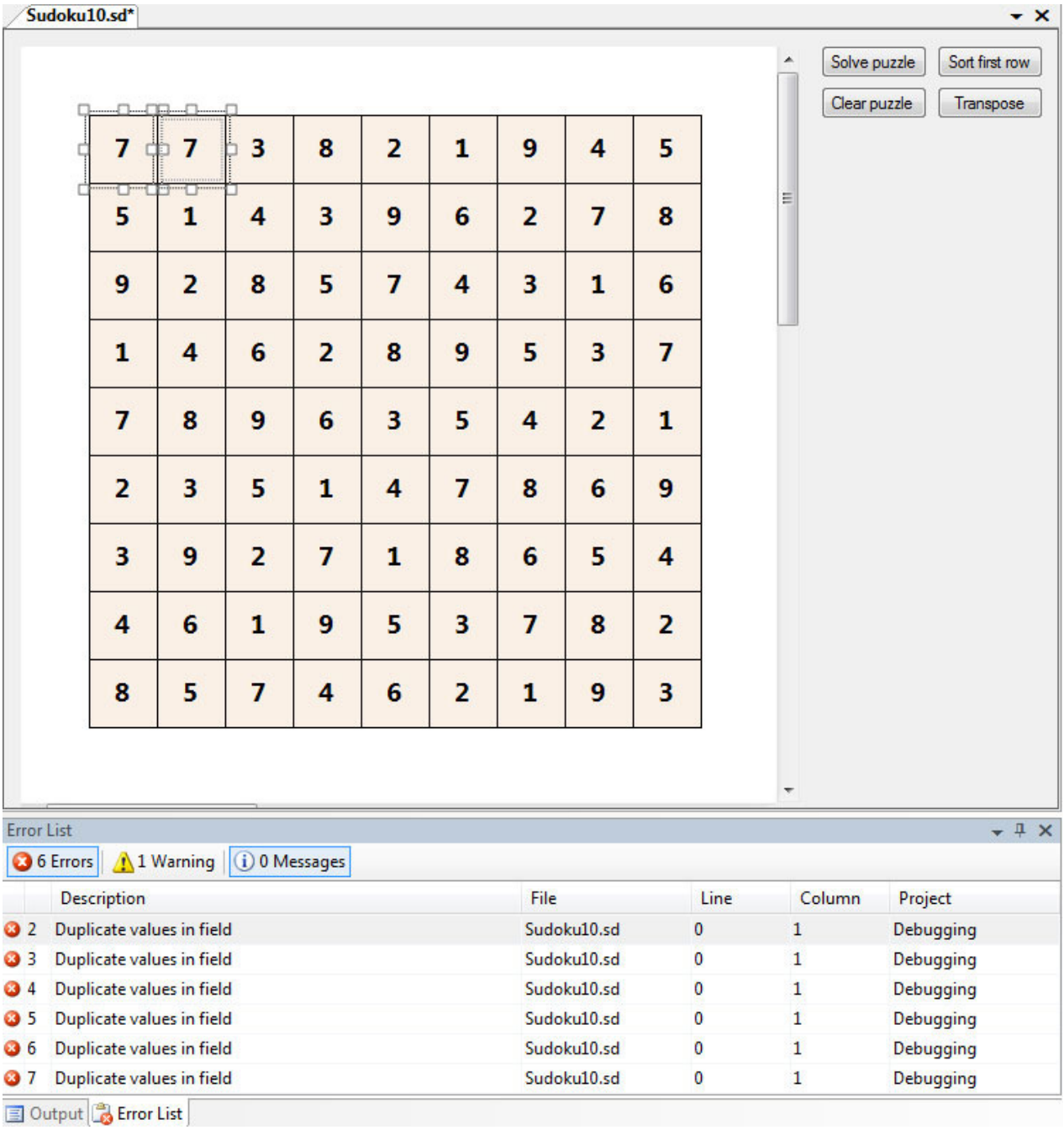

Figure 13 Visual Studio: Invalid Sudoku

4.3 Textual representation

The grammar for the textual editor should be quite simple and straight forward, letting the user create a Sudoku by filling `Rows` with `Cells` simply by writing text. The user should not be concerned with creating `Columns` and `Boxes` or adding `Cells` to these, so this should be handled automatically in the background and is therefore not represented in the grammar. Thus most of the references between elements must be handled automatically. The elements in the grammar represent the elements from the Sudoku structure presented in chapter 4.1.

The ideal grammar for the textual representation:

Puzzle::= "Puzzle (" iDimension ") = " Field.
Field ::= Row.
Row ::= "Row (" Cell ")"*.
Cell::= iCellValue ",".
iDimension ::= INT.
iCellValue ::= INT.

4.3.1 Grammar in oAW xText

For the textual editor I started using xText, a part of openArchitectureWare (see [53]). xText lets the developer write grammars for domain-specific languages using the xText editor. However xText turned out not to work very well, and when errors occurred the error messages were not that informative. I spent about a month working on this without satisfying results. After discussing the problems with using oAW with my supervisors we decided that I should try out TEF (see [40]) instead.

Grammar created in oAW:

```
Puzzle:
      "Puzzle (" iDimension=INT ")" "="
      (fields+=Row)*;

Cell:
      iCellValue=INT;

Row:
      "("
            (cells+=Cell)*
      ")";
```

4.3.2 TEF v. 0.5.0

With TEF it is relatively easy to construct a grammar. First you provide the path to your Ecore meta-model. You are then free to write your grammar specifying the necessary elements and creating your own syntax based on the provided model.

The Sudoku grammar created with TEF:

```
syntax toplevel PuzzleTpl,
          ecorepath "platform:/resource/Sudoku/resources/newstructure.ecore" {

      element CellTpl for Cell{
            single for iCellValue, with INTEGER;
      }
      element RowTpl for Row{
            "Row"; "(";
            sequence for cellsInRow, with @CellTpl, seperator ",", last false;
            ")";
      }
      element PuzzleTpl for Puzzle{
            "Puzzle"; "(";
            single for iDimension, with INTEGER;
            ")";
            "=";
            sequence for Elements, with @FieldTpl, seperator ",", last false;
      }
      choice FieldTpl for Field{
            @RowTpl
      }
}
```

The grammar specifies the elements/element templates for `Puzzle`, `Field`/`Row` and `Cell`. A `Cell` has a single slot for the `Cell` integer attribute `iCellValue`, while a `Row` has a sequence of slots for `Cell`. A `Puzzle` has a sequence of slots for `Field` and the choice for `Field` is `Row`. If necessary, `Column` and `Box` could be added as choices for `Field` as well as they also inherit `Field`. In addition, `Puzzle` has a single slot for the `Puzzle iDimension` as an integer. To create a new `Puzzle` one can now write:

```
Puzzle (9)=
Row (6,7,3,8,2,1,9,4,5),
Row (5,1,4,3,9,6,2,7,8),
Row (9,2,8,5,7,4,3,1,6),
Row (1,4,6,2,8,9,5,3,7),
Row (7,8,9,6,3,5,4,2,1),
Row (2,3,5,1,4,7,8,6,9),
Row (3,9,2,7,1,8,6,5,4),
Row (4,6,1,9,5,3,7,8,2),
Row (8,5,7,4,6,2,1,9,3)
```

This creates a new `Puzzle` with `iDimension` 9 containing nine `Rows` all containing 9 `Cells` with an `iCellValue`. `Columns`, `Boxes` and references are created as well but

this is not visible to the user. The grammar differs somewhat from the ideal solution as TEF provides the opportunity to specify that a separator is not necessary for the last element. For example, in the Sudoku in the written Sudoku shown, there is no "," after the last `Row`, and there is no "," after the last `Cell` of each `Row`.

Automatic handling of `Columns` and `Boxes` must also be done by overriding the *check* method and using EMF. This is in some ways misuse of the *check* method, but it is currently the only option for creating these elements. When doing this it is important to make sure this only happens when there are not 9 `Columns` and `Boxes` in the `Puzzle` already as this would result in a wrong number of `Boxes` and `Columns` in a `Puzzle`. `Boxes` and `Columns` are easily added to the `Puzzle` as shown in Code example 3. `Cell` and its references are created in the same way. rX represents `Row` X in the `Puzzle` and the method `getCellsInRow` retrieves the `cells` contained by that `Row`.

```
NewstructureFactory factory = NewstructureFactory.eINSTANCE;
Puzzle puzzle = (Puzzle)((EMFModelElement)modelElement).getEMFObject();
EList<Field> puzzleList = puzzle.getElements();
EList<Field> elements = puzzle.getElements();

//Get all rows from modell instance
r1 = (Row)elements.get(0);
r2 = (Row)elements.get(1);
r3 = (Row)elements.get(2);
r4 = (Row)elements.get(3);
r5 = (Row)elements.get(4);
r6 = (Row)elements.get(5);
r7 = (Row)elements.get(6);
r8 = (Row)elements.get(7);
r9 = (Row)elements.get(8);
//Create a new Box and add cells to the box. These are retrieved from the rows
Box b1 = factory.createBox();
b1.getCells().add(r1.getCellsInRow().get(0));
b1.getCells().add(r1.getCellsInRow().get(1));
b1.getCells().add(r1.getCellsInRow().get(2));
b1.getCells().add(r2.getCellsInRow().get(0));
b1.getCells().add(r2.getCellsInRow().get(1));
b1.getCells().add(r2.getCellsInRow().get(2));
b1.getCells().add(r3.getCellsInRow().get(0));
b1.getCells().add(r3.getCellsInRow().get(1));
b1.getCells().add(r3.getCellsInRow().get(2));
//Add a box to puzzle
puzzleList.add(b1);
//Set box.puzzle to this puzzle.
b1.setPuzzle(puzzle);
//Code omitted, same procedure for all boxes

//Create new column and add cells to the it. These are retrieved from the rows
Column c1 = factory.createColumn();
//Add cell references to columns
c1.getCells().add(r1.getCellsInRow().get(0));
c1.getCells().add(r2.getCellsInRow().get(0));
c1.getCells().add(r3.getCellsInRow().get(0));
```

```
c1.getCells().add(r4.getCellsInRow().get(0));
c1.getCells().add(r5.getCellsInRow().get(0));
c1.getCells().add(r6.getCellsInRow().get(0));
c1.getCells().add(r7.getCellsInRow().get(0));
c1.getCells().add(r8.getCellsInRow().get(0));
c1.getCells().add(r9.getCellsInRow().get(0));
//Add column to puzzle
puzzleList.add(c1);
//Set column.puzzle to this puzzle
c1.setPuzzle(puzzle);
//Code omitted, same procedure for all columns
```

Code example 3 Eclipse: Code for creating Box and Column in EMF used with TEF

Please note that a new version of TEF has been released after I covered TEF in this project.

Implementing constraints

For now constraints in TEF must be implemented in the template classes by overriding the *check* method for each class you want to add constraints to. This method returns a string and if all constraints are validated the returned string is null. The method is run every time a corresponding model element is changed. If any constraint is violated, an error message (string) must be returned. The constraints can be written in pure Java code, or one can take advantage of using OCL. In this project the constraints used are written in OCL. I wrote most constraints in Java as well, but for the SMILE project OCL is preferred.

One minor issue with this solution for TEF constraints is that there is currently no way to fire constraints for `Column` and `Box` (or `Field`) directly as they are not directly represented in the grammar, thus they do not have their own template classes. To work around this problem I added the constraint code for `Box` and `Column` in the `Cell` template file's check method. This is run every time a `Cell` is changed, but when a `Cell` changes so does a `Row`, `Column` and a `Box`, so there is really no significant difference. In order to get the correct `Column` and `Box` elements, I retrieve these from the `Cell` that is changed. As mentioned, a `Cell` refers to one `Row`, `Column` and `Box` so they are easily retrieved as shown in Code example 4 on the next page. The IModelElement modelElement provides the `Cell` element that has been edited.

```
@Override
public String check(IModelElement modelElement, SemanticsContext context){
    //Code omitted
    //Retrieve this cells box and column
    Cell cell = (Cell)((EMFModelElement)modelElement).getEMFObject();
    box = cell.getBox();
    column = cell.getColumn();
    //Code omitted
}
```

Code example 4 Eclipse: Code snippet to retrieve Box and Column objects from Cell

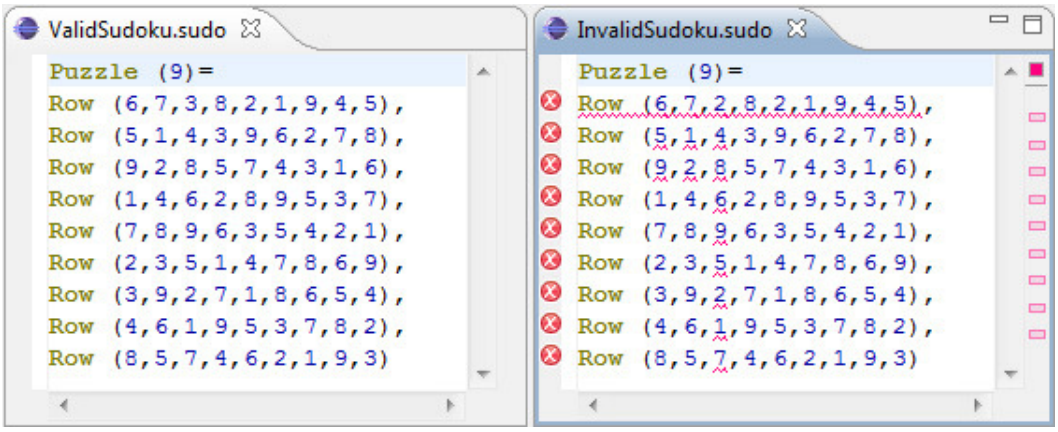

Figure 14 Eclipse: A valid (left) and invalid (right) Sudoku in TEF

Figure 14 shows an invalid Sudoku in TEF where constraints are violated. `Cell` 3 in `Row` 1 has value 2, but it should be 3. This results in error markings for all `Cells` in the `Cells` `Row`, `Box` and `Column`. To find the `Cell` with an error, one simply finds where the error marked `Row` and `Column` intersects. To see the error message from a violated constraint one can simply hover the mouse pointer over the red x's on the left side.

4.3.3 Visual Studio 2005 with SDK/DSL Tools

By using Visual Studio 2005 with DSL Tools, you can create custom graphical designers that use your domain-specific diagram notation (see [47]). DSL Tools does however not give you the possibility of creating textual DSLs, and it does not seem to be something that Microsoft plans to implement in the future as of now as DSL Tools is meant for creating graphical DSLs.

4.4 Graphical representation

For the graphical representation we want editors that let a user edit a graphically presentable Sudoku. Ideally when the user starts a new Sudoku editor, she should be presented with a ready-to-use Sudoku that contains the correct number of `Rows` containing `Cells`, `Columns` and `Boxes`. All references must be handled automatically. Only `Row` and `Cell` need graphical elements, the rest should be handled in the background.

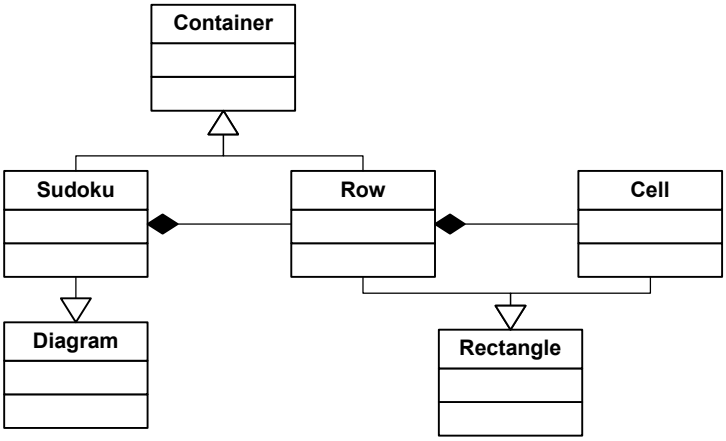

Figure 15 Ideal graphical representation model

4.4.1 GMF

GMF allows the creation of graphical elements and the mapping of these graphical elements to model elements. This lets the developer create graphical editors for their models in a relatively simple way. Using GMF is more user friendly than GEF as wizards lead the developer through large parts of the development and generated necessary code. Custom code can be used by overriding the generated methods. As a bridge between EMF and GEF, the GMF takes an Ecore model as input and combines this model with graphic- and tool definitions to create a mapping definition and then a generator model that generates an editor/diagram plug-in based on the provided model and definitions.

The GMF solution differs from the expected solution as is provides more functionality for for example element location and size. The graphic definition is held in the*.gmfgraph file. By adding graphical elements and editing properties of these elements in this file, one can customize the graphical elements for all model elements. Shape, color, image,

size and layout are among the possible properties for these elements as well as e.g. lines for displaying relationships (not necessary in this project). In addition one can create tools for creating new elements by drag-and-drop and mappings between tools, graphical elements and model elements. `Cell` values are edited simply by selecting a `cell` and entering the new value.

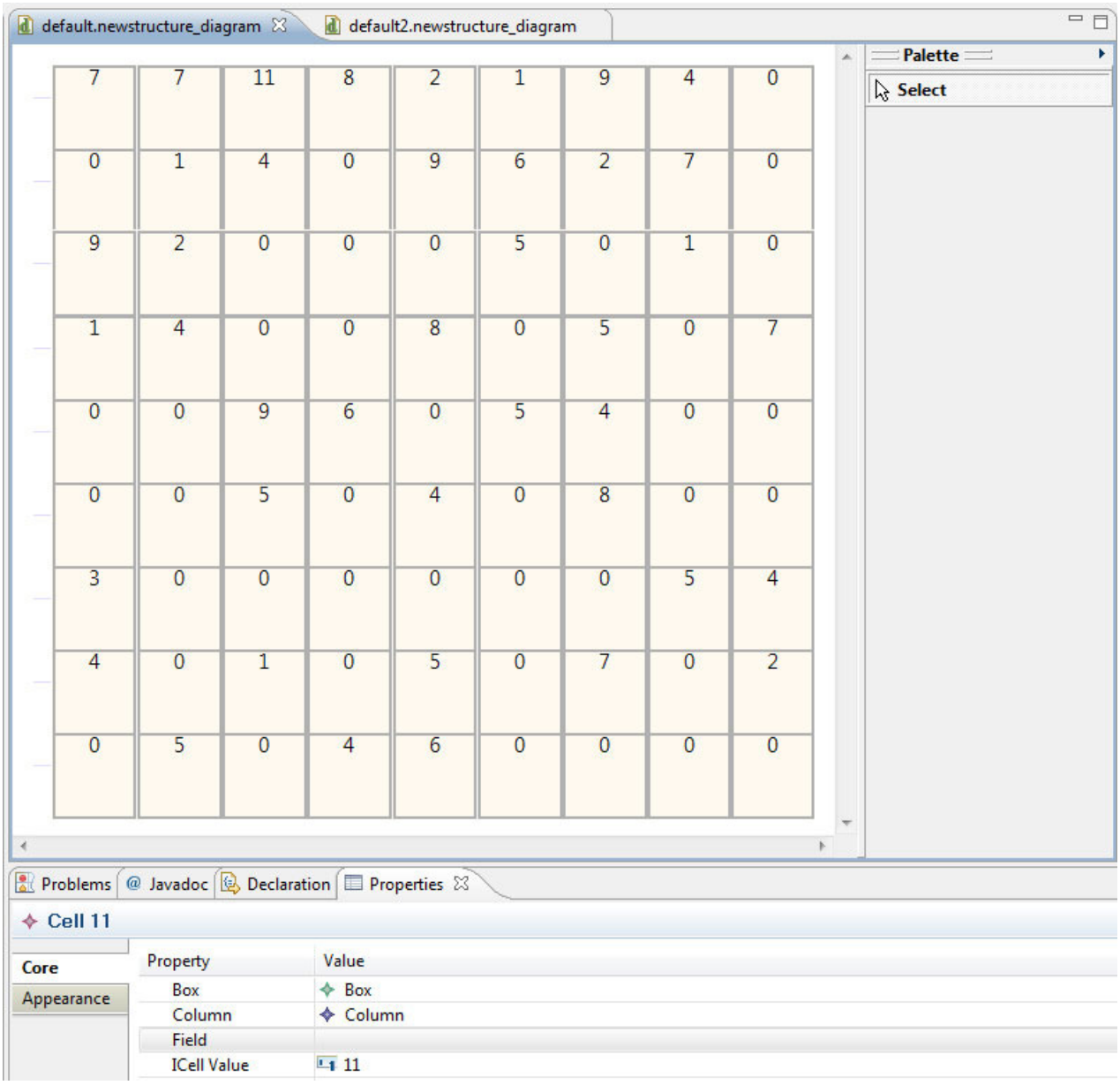

Figure 16 Eclipse: graphical editor created in GMF

Figure 16 shows the editor created with the GMF. Unfortunately I have not succeeded in the correct placement of `Rows`, so when a new diagram is created the rows appear on a horizontal line and must be placed in a `Column` manually. The `Rows` are filled with 9

`Cells` each and these cannot be moved. The code for this custom layout must be inserted in the generated code.

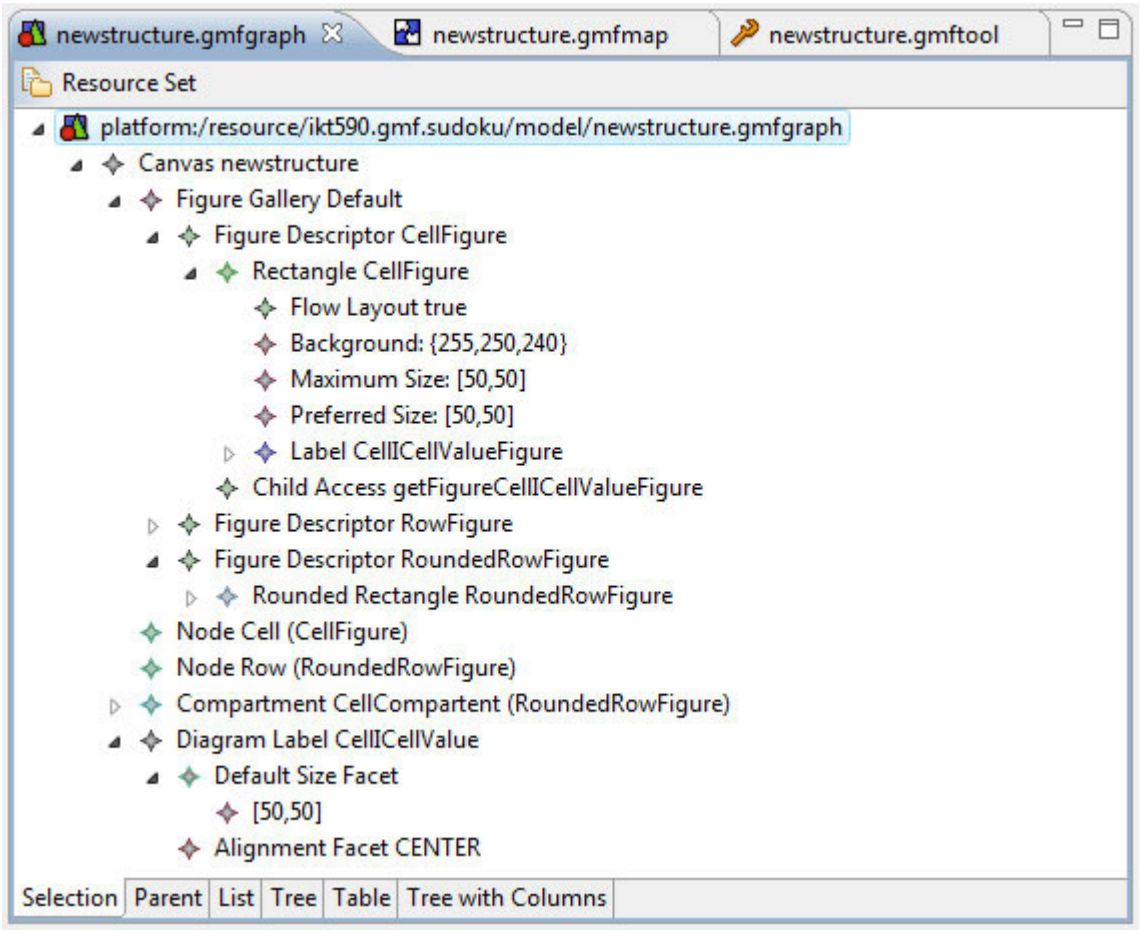

Figure 17 Eclipse GMF: Graphical definition

Creating the initial model

To provide the users of the editor with a ready-to use Sudoku when starting a new diagram, the elements must be created manually by adding custom model code into the generated code. GMF generates a lot of code, so it is not always easy to find out where to put your custom code in order for it to work properly. The elements and all necessary relationships are created using the EMF. The procedure for creating the initial model is explained in chapter 5.3.10 of this report. For a code example see Code example 5.

```
private static Puzzle createInitialModel() {
      NewstructureFactory factory = NewstructureFactory.eINSTANCE;
      //Create puzzle in the modell instance
      Puzzle puzzle = factory.createPuzzle();
      puzzle.setIDimension(9);
      //Create rows
      Row r1 = factory.createRow();
      //Set row.puzzle to this puzzle
      r1.setPuzzle(puzzle);
      //Create cells
```

```
        Cell c11 = factory.createCell();
        Cell c12 = factory.createCell();
        Cell c13 = factory.createCell();
        Cell c14 = factory.createCell();
        Cell c15 = factory.createCell();
        Cell c16 = factory.createCell();
        Cell c17 = factory.createCell();
        Cell c18 = factory.createCell();
        Cell c19 = factory.createCell();

        //Add cells to row containment
        r1.getCellsInRow().add(c11);
        r1.getCellsInRow().add(c12);
        r1.getCellsInRow().add(c13);
        r1.getCellsInRow().add(c14);
        r1.getCellsInRow().add(c15);
        r1.getCellsInRow().add(c16);
        r1.getCellsInRow().add(c17);
        r1.getCellsInRow().add(c18);
        r1.getCellsInRow().add(c19);

        //Add cells to row
        r1.getCells().add(c11);
        r1.getCells().add(c12);
        r1.getCells().add(c13);
        r1.getCells().add(c14);
        r1.getCells().add(c15);
        r1.getCells().add(c16);
        r1.getCells().add(c17);
        r1.getCells().add(c18);
        r1.getCells().add(c19);

        //set row reference from each cell to the row that contains/refers to it

        EList<Cell> r1cells = r1.getCells();
        for (Cell c : r1cells) {
              c.setRow(r1);

        puzzle.getElements().add(r1);

        //Code omitted, same procedure for all elements that must be created in

        //code
}
```

Code example 5 Eclipse GMF/EMF: Create initial model (code excerpt)

Constraints in GMF with OCL

Adding constraints to GMF is relatively easy and the procedure is described in [33] part 2 and chapter 5.3.10 of this report. The constraints are added as pure OCL statements.

To validate the Sudoku model, there is a Diagram menu item, next to the Edit menu item. Simply click diagram and select Validate. This will run the given constraints, and if any constraints are violated, the error message(s) will be displayed in the problems list of Eclipse. The OCL constraints used are presented in the beginning of chapter 4.2.

4.4.2 GEF

Unfortunately, there is no working solution/editor for Sudoku created in GEF in this project. This is due to the lack of complete tutorials combined with the available time in this project. More about this can be found in chapter 5.3.9.

4.4.3 Visual Studio 2005 with SDK/DSL Tools

Visual Studio is built specifically for creating graphical designers for domain-specific languages. The Sudoku designer lets the user edit her own `Puzzle` by assigning values to `cells`. Each `Row` and `Cell` is represented by graphical elements, a `Row` containing 9 `Cells`.

It is fairly simple to create these graphical elements if you are familiar with using Visual Studio. This is no surprise as this is the purpose of DSL Tools – to create graphical designers. There are several starting templates, which get you started with some elements that you can change to fit your need and of course add new elements. Using the same model as in the structure part, you can create a diagram element, e.g. shapes (see Figure 18) for each class for which you need graphical elements. These shapes are easily mapped to the corresponding class in the model. In the properties view you can change the appearance of the shape, like color, line thickness, geometry, size and so on.

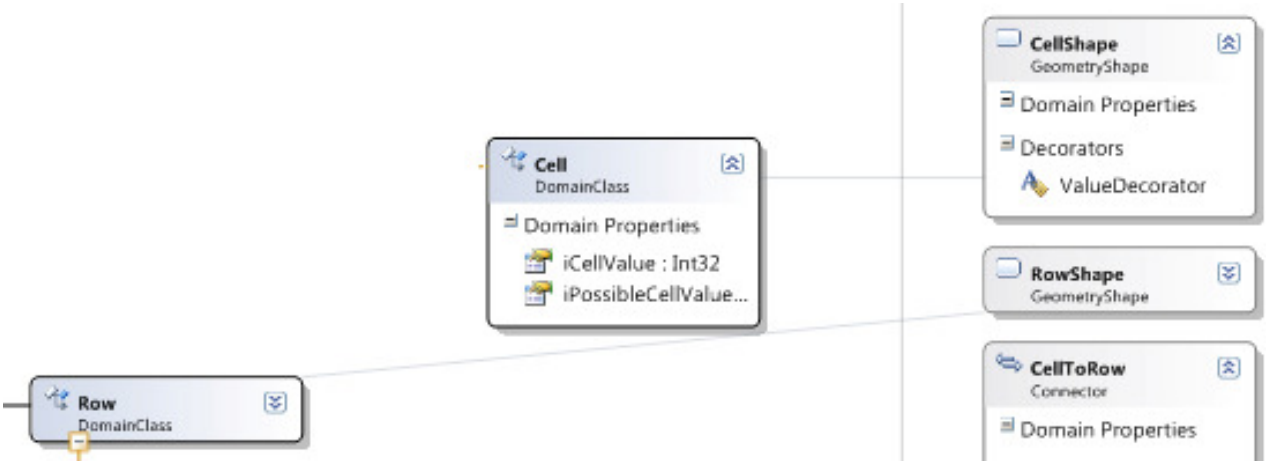

Figure 18 Visual Studio: Cell and Row DomainClass with mapping to graphical shape elements

For my graphical editor, a `Cell` and a `Row` have the same height, but the `Row` width is 9x a `Cells` width.

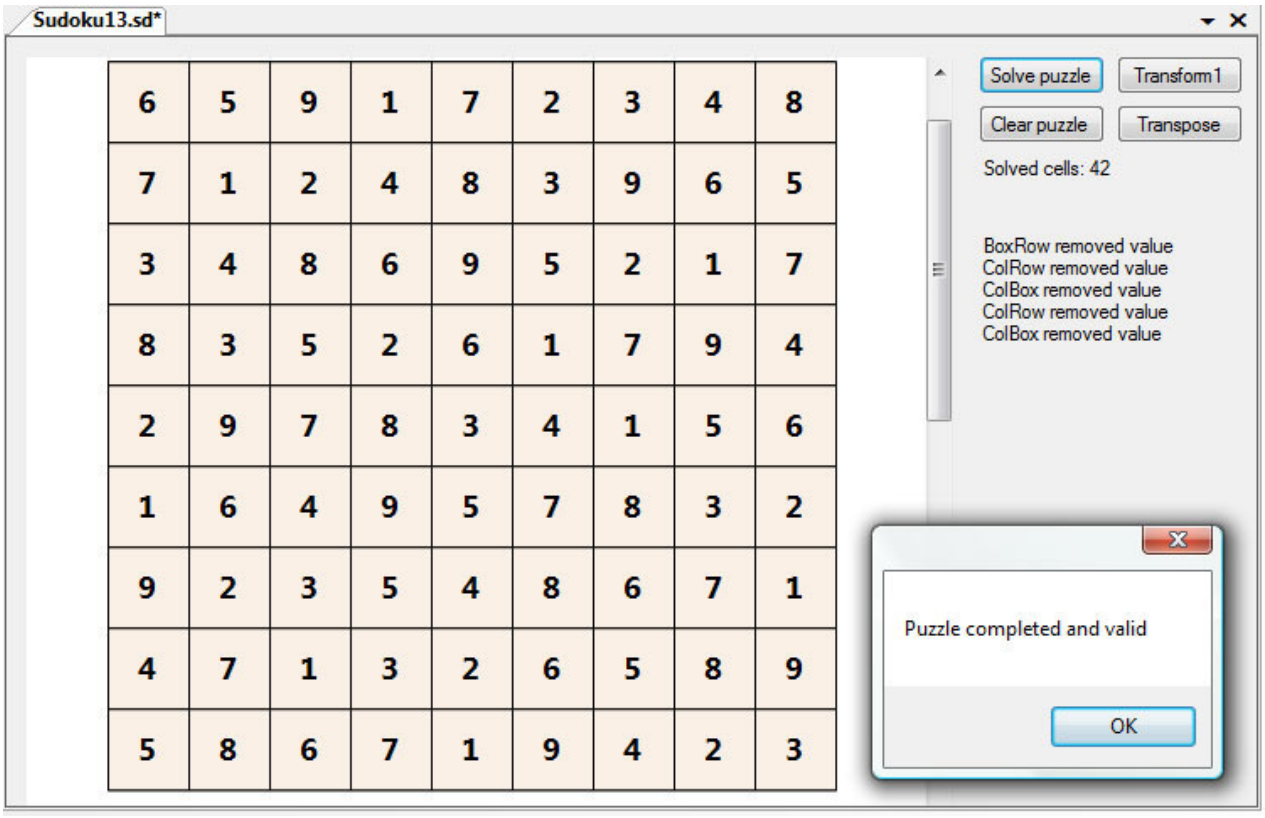

Figure 19 Visual Studio: debugging mode, a valid and solved Sudoku

Custom rules

Custom rules can be added to your DSL to fire on specific events. I created custom rules for the events when a `Row`, `Column`, `Box` or `Cell` is added to the model. When a `Cell` is added to the diagram, its `iPossibleCellValues List<int>` is filled with the numbers (integers) from 1 to 9. The code for this must be given as a custom rule.

When a new Sudoku is created, it is already filled with 9 `Rows` containing 9 `Cells` that are visible to the user. In addition to this, there are also 9 `Boxes` and `Columns` that have the necessary references to `cells`. This is achieved by creating the whole `Puzzle` structure in XML in the diagram files (sd.diagram and sd.sd) in the DslPackage of the project.

`Cell` references from `Row`, `Column` and `Box` are automatically handled in the diagram files. However, the `Cells` references to `Box` and `Column` are added easiest in a custom rule like this:

```
[RuleOn(typeof(Column), FireTime = TimeToFire.TopLevelCommit)]
 public class AddColToCellRule : AddRule{
    public override void ElementAdded(ElementAddedEventArgs e) {
       Column col = e.ModelElement as Column;
```

```
        Store store = col.Store;
        using (Transaction txCreateElem =
             store.TransactionManager.BeginTransaction("Create references")){
                if (col != null) {
                    foreach (Cell c in col.cellsInField) {
                        c.column = col;
                    }
                }
                txCreateElem.Commit();
            }
        }
    }
```

Code example 6 Visual Studio: Add column reference to cell

For each `Column` that is added to the model, all the `cells` that the `Column` refers to by its `cellsInField` will again refer to the `Column` as its `column`. The same is done for `Box`.

The framework must be notified about all custom rules in order for it to fire. This is done by overriding the `GetCustomDomainModelTypes` method of the domain model in a new partial class as shown in Code example 7.

```
    public partial class SudokuDomainModel{
        protected override System.Type[] GetCustomDomainModelTypes(){
            return new System.Type[] { typeof(RowPositionAddRule),
                                       typeof(AddBoxToCellRule),
                                       typeof(AddColToCellRule),
                                       typeof(CellPositionAddRule) };
        }
    }
```

Code example 7 Visual Studio: Register custom rules

I have also used custom rules to handle the layout and restrict the possibility to resize the graphical elements. Note that these custom rules also apply to serialized elements unless disabled by code. Therefore these custom rules will apply to the elements that are created in the XML file.

```
// Rule invoked when the user is changing a shape's outline.
public class RowBoundsRule : BoundsRules{
    public override RectangleD GetCompliantBounds(ShapeElement shape,
                                                   RectangleD proposedBounds{
        return new RectangleD(proposedBounds.Location, new SizeD(4.5, 0.5));
    }
}
public class CellBoundsRule : BoundsRules{
    public override RectangleD GetCompliantBounds(ShapeElement shape,
                                                  RectangleD proposedBounds){
        return new RectangleD(proposedBounds.Location, new SizeD(0.5, 0.5));
    }
}
```

Code example 8 Visual Studio: Restrict resize of Row and Cell Shape elements

```
[RuleOn(typeof(ParentShapeContainsNestedChildShapes), FireTime =
                                                    TimeToFire.TopLevelCommit)]
public class CellPositionAddRule : AddRule{
    private static int counterX = 0;
    private static int counterY = 0;
    private double offsetX = 0.5;
    private double offsetY = 0;
    private PointD location = new PointD(0, 0.5);
    public override void ElementAdded(ElementAddedEventArgs e){
        CellShape shape = null;
        ParentShapeContainsNestedChildShapes nestedLink = e.ModelElement as
                                            ParentShapeContainsNestedChildShapes;
        if (nestedLink != null){
            shape = nestedLink.NestedChildShapes as CellShape;
        }
        if (shape != null && shape.Diagram != null){
            shape.IsExpanded = true;
            shape.Location =new PointD(location.X+offsetX,location.Y+offsetY);
            counterX++;
            if (counterX == 1){
                offsetX += shape.Size.Width;
                counterX = 0;
                counterY++;
            }
            if (counterY == 9){
                offsetY += shape.Size.Height;
                counterY = 0;
                offsetX = 0.5;
            }
        }
    }
}
```

Code example 9 Visual Studio: code for CellShape location

Code example 8 shows code for restricting resizing of `CellShape` and `RowShape` while Code example 9 presents code for the location of a `CellShape` when it is added to the diagram. Similar code is added for `RowShape`.

To learn the procedure for adding a form with buttons see [51].

4.5 Run/execution

The execution part in this project consists of some solving strategies of Sudoku formalized code that will be run when the model is executed. The main objective in this project is not to solve very difficult Sudoku, so I will not implement the more advanced solving strategies.

The strategies I will implement are:

- ✓ Elimination.
- ✓ Single Cell candidate.
- ✓ Hidden single cell candidate.
- ✓ Locked candidates.

These strategies are described in chapter 3.1.1.

In order to perform these solving strategies I need a new attribute for `Cell`, preferably a `List<int>`. This list will be named `iPossibleCellValues` and I will use it to keep track of the values that are available in a `Cell` at any given time during the solving process.

Ideal execution of Sudoku:

```
Run(s:Sudoku) =def
   forall f in s.field do RunElimination (f)
   forall f in s.field do RunSingleCell (f)
   forall f in s.field do RunHiddenCell (f)
   forall f in s.field do RunLockedCandidates (f)

RunElimination(f:Field) =def
   forall c in f.cells with c.iCellValue<>null do
      forall cc in f.cells do
         delete c. iCellValue from cc. iPossibleCellValues

RunSingleCell(f:Field) =def
   forall c in f.cells with c. iCellValue = null and c. iPossibleCellValues.size = 1
      choose v in c.iPossibleCellValues do c. iCellValue= v
```

```
RunHiddenCell(f:Field) =def
   forall v in [1..iDimension] do
      let possibleCells = { c in f.cells with v in c. iPossibleCellValues}
         if possibleCells.size = 1 then
            choose c in possibleCells do
               c. iCellValue= v

RunLockedCandidates(f:Field) =def
forall otherF in puzzle.field with (f.cell ∩ otherF.cell).size > 1 do
   forall v in [1..iDimension] do
      let possibleCells = { c in f.cells with v in c.iPossibleCellValues}
         if possibleCells intersect otherF.cells
            forall cc in otherF.cells with cc not in f.cells
               delete v from cc.iPossibleValues
```

4.5.1 medini QVT

Due to the lack of more suitable solutions, I will try to create the execution part of this projects Eclipse based solution using medini QVT. I consider medini QVT to be sufficient as a run is really transitions between states and not too different from some transformations.

Solve puzzle

I did not succeed in implementing any solving strategies using medini QVT, but one of my test solutions are presented in Code example 10. There were several problems and limitations that I did not expect. For instance there is a compatibility problem between OCL and EMF as OCL does not support the `EEList` data type. This prevented me from retrieving the value `iPossibleValues->at(0)` which is necessary in order to solve a `Cell`. This function is actually necessary for all solving strategies. The code below shows how I expected to be able to solve a `Cell` by enforcing the `iCellValue = iPossibleValues->at(0)` of all `Cells` that has only 1 entry in the `iPossibleValues` list.

```
transformation Solve(source : newstructure, target: newstructure){
      top relation solveCell{
            checkonly domain source cell:Cell{
                  iCellValue = 0};
            enforce domain target cell:Cell{
                  iCellValue = iPossibleValues->at(0)};
            where{
                  iPossibleValues->size()=1;}
}}
```

Code example 10 Eclipse: Expected Single Cell Candidate solving strategy using medini QVT

I also tried to implement the Elimination strategy, but failed to do so as I have yet to find out how to delete elements from an `EEList` in QVT.

4.5.2 Visual Studio 2005 with SDK/DSL Tools

In Visual Studio, execution is as all other aspects implemented in your preferred language for VS, in my case C#.

As mentioned I need a List type Domain Property in the `Cell` Domain Class. This type is strangely enough not supported in the DSL Tools structure; thus I had to create the type I needed myself. This was relatively easy when after some searching I found someone who has had the same problem. The answer was found in an MSDN forum (see [50]) for DSL Tools users.

Solve puzzle

A `Cell` is solved when its `List iPossibleCellValues` only contains one value. If a `Cell` only has one possible value, this must be the correct value for this `Cell` and its `iCellValue` is set to this value. When the “Solve puzzle”e button is clicked, values are deleted from the `iPossibleCellValues` by the rules of the solving strategies mentioned in the beginning of this chapter. A `Puzzle` is solved when all `Cells` are filled (not empty) and the `Puzzle` is valid. In addition, a label displays how many `Cells` have been solved and what solving strategies are used. Note that this only lists the more advanced strategies used. In the following code examples only the essential excerpts of the Hidden single cell candidate and Locked candidates strategies are shown.

Hidden single cell candidate

```
foreach (Field element in puzzle.Elements){
    int iCount = 0;
    for (int i = 1; i <= 9; i++){
    //Check how many cells contain value i
        foreach (Cell c in element.cellsInField){
            if (c.iPossibleCellValues.Contains(i))
                iCount++;
        }
        //If only one cell contains value i this must be the correct value
        if (iCount == 1){
            foreach (Cell c in element.cellsInField){
                if (c.iCellValue == 0){
                    using (Transaction txCreateElem =
                                store.TransactionManager.BeginTransaction("")){
                        if (c.iPossibleCellValues.Contains(i)) {
                            c.iCellValue = i;
                            iSolvedCells++;
                            txCreateElem.Commit();
                            lblSolved.Text = "Solved cells: " + iSolvedCells;
                            lblRule.Text += "Hidden single candidate\r\n";
}   }   }   }   }   }   }
```

Code example 11 Visual Studio execution: Hidden single cell candidate strategy

Locked candidates (one out of four)

```
List<Cell> sameValue = new List<Cell>();
foreach (Field element in puzzle.Elements){
    if (element is Box){
        Boolean sameRow = true;
        for (int i = 1; i <= puzzle.iDimension; i++){
            //Collect all cells that contain i
            foreach (Cell c in element.cellsInField){
                if (c.iPossibleCellValues.Contains(i)){
                    sameValue.Add(c);
                }
            }
            //Check if they all belong to the same row
            foreach (Cell c1 in sameValue) {
                foreach (Cell c2 in sameValue){
                    if (!c1.Equals(c2) && !c1.row.Equals(c2.row))
                        sameRow = false;
                }
               //If they all belong to the same row, delete i fom the cells in
                                                    this row but different box
                if (sameRow == true){
                    foreach (Cell c3 in c1.row.cellsInField){
                        if (!c1.Equals(c3) && !c3.box.Equals(c1.box) &&
                                        c3.iPossibleCellValues.Contains(i)){
                            using (Transaction txCreateElem =
                               store.TransactionManager.BeginTransaction("")){
                                c3.iPossibleCellValues.Remove(i);
                                iRemovedCells++;
                                setRemovesCellsLabel();
                                lblRule.Text += "BoxRow removed value\r\n";
                                txCreateElem.Commit();
}   }   }   }   }   }   }
```

Code example 12 Visual Studio execution: Locked candidates strategy

The Visual Studio solution is very different from the ideal solution presented at the beginning of this chapter. As the execution aspect is integrated with the rest of the Visual Studio solution, I have the possibility to use buttons so that the user can execute the run by clicking these buttons. In addition there is a lot of extra code, e.g. for transactions, writing to labels for user information and handling exceptions. All of this extra code is not necessary, but still included to make the solver more user-friendly.

Clear Puzzle

Clicking the Clear puzzle button simply sets all `iCellValues` to 0 and populates the `iPossibleValues` again.

4.6 Transformation

The specification must present some model to model transformation. There are some transformations that can be performed on a Sudoku without altering the logic [11][10]:

- ✓ Permutations of rows and columns within blocks.
- ✓ Permutations of block rows and columns.
- ✓ Permutations of the symbols used in the board.
- ✓ Transposing the Sudoku.

A block refers to a row or column of boxes, e.g. Row 1, 2 and 3 is a block and so is Column 4, 5 and 6.

I will try two of these transformations in this project:

- ✓ **Permutations of the symbols used in the board**

  The symbols are in this project the numbers from 1 to 9. By permutation we can arrange the Sudoku in such a way that the cells in the first row are ordered ascending from 1 to 9. In order to achieve this, the `iCellValues` from the first `Row` must be retrieved, and then used to switch all the `iCellValues` in the `Puzzle`. E.g. if the first cell in the first `Row` has `iCellValue = 6`, then all `Cells` where `iCellValue = 6` must be changed to 1.
- ✓ **Transpose**

  The matrix transpose, most commonly written M^T, is the matrix obtained by exchanging M's rows and columns [10]. Stated differently, given an *m*x*n* matrix M, the transpose of M is the *n*x*m* matrix denoted by M^T whose columns are formed from the corresponding rows of M [6]: $M^T_{ij} = M_{ji}$ for $1 \leq i \leq n$, $1 \leq j \leq m$

In this project I will use in-place transformations, meaning that the source and target models are the same.

Ideal Sudoku transformation: Sort the first row:

```
Run(s:Sudoku) =def
   forall f in s.field do RunSort (f)
   forall f in s.field do RunElimination (f)

RunSort(f:Field) =def
forall c:Cell do
```

c.iCellValue = rows[1].cell[i].iCellValue where c.iCellValue = i

Ideal Sudoku transformation: Transpose

RunTranspose(f:Field) =def

N: [1..9]

forall i ∈ N do

forall j ∈ N do

puzzle.rows[i].cell[j].iCellValue = puzzle.rows[j].cell[i].iCellValue

4.6.1 medini QVT

Finding tutorials on QVT turned out to be quite difficult. I used the QVT specification from OMG and the built in model-to-model examples in the medini QVT installation to figure out how to build my own transformations.

To use medini QVT for model to model transformations one must provide meta-models for the models to be transformed. In this case this will be the ecore file describing the Sudoku structure which will act as both source and target meta-model.

Sort the first row

To sort the first `Row` by permuting `Cell` values, all `Cell` values of the first `Row` must be retrieved. The `Cell` values throughout the `Puzzle` must be changed to the same value as the same value's original position in the first `Row`. This means that if the `iCellValue` in the first position of the first `Row` was 6, then all 6 in the Sudoku must be changed to 1. If the value in the second `Cell` was 3, then all 3's must be set to 2 and so on. This turns out to be problematic as there might already be e.g. a number of 2 in the `Puzzle`. Then, if the value in `Cell` 3 is 2, all 2's must be changed to 3. However there are really two kinds of 2, the kind that is already correct and the ones that are not. To avoid this problem, the `Cell iCellValues` are set to the value they should have + 10 meaning 6 is set to 16. To get the correct values a new transformation must be run to remove the extra 10 from each `iCellValue`. I tried to implement this second transformation by simply subtracting 10 from each `Cell` in the `Puzzle` for which `iCellValue > 10` but the > operator was not accepted in a relation. If the condition was placed in a where-clause, the transformation went into an infinite loop. Code example 13 shows some of the code for sorting the first `Row` with medini QVT.

```
transformation SortFirstRow(source : newstructure, target: newstructure){
top relation getNinthValue {
      row1:Field;
      checkonly domain source puzzle:Puzzle
      {};
      checkonly domain source cell:Cell
      {iCellValue = row1.cells->at(9).iCellValue};
      enforce domain target cell:Cell
      {iCellValue = 19};
      when
      {row1 = puzzle.Elements->select(f:Field|f.oclIsTypeOf(Row))->first();}
}
top relation getEightValue {
      row1:Field;
      checkonly domain source puzzle:Puzzle
      {};
      checkonly domain source cell:Cell
      {iCellValue = row1.cells->at(8).iCellValue};
      enforce domain target cell:Cell
      {iCellValue = 18};
      when
      {row1 = puzzle.Elements->select(f:Field|f.oclIsTypeOf(Row))->first();}
}
--Same procedure followed for all values, code omitted

}}
transformation minusTen(source : newstructure, target: newstructure){
--Code omitted, same procedure as the two following relations for all values
top relation change18 {
      checkonly domain source newstructure:Cell {
      iCellValue = 18};
      enforce domain target newstructure:Cell {
      iCellValue = 8};
}
top relation change19 {
      checkonly domain source newstructure:Cell {
      iCellValue = 19};
      enforce domain target newstructure:Cell {
      iCellValue = 9};
}
top relation noChange {
      value:Integer;
      checkonly domain source newstructure:Cell {
      iCellValue = value};
      enforce domain target newstructure:Cell {
      iCellValue = value};
}}
```

Code example 13 Eclipse: Transformation for sorting the first row

Transpose

I failed to implement the Transpose transformation in medini QVT. The problem as I see it is that a QVT Relation on e.g. a `Cell` is run on all `Cells` but one by one. That means that during a transpose, the second `Cell's` value in the first `Row` is moved to the second `Cell` in the first `Column`. This is fine until we come to the second `Row` and want to move the value from the first `Cell` (that is the first `Column`). This value has already been set to its correct value and does not represent the original value we need.

4.6.2 Visual Studio 2005 with SDK/DSL Tools

The built-in transformation tool for DSL Tools is the text templates [52]. In DSL Tools a text template is a file that can contain both text blocks and control logic. When a text template is transformed, the control logic combines the text blocks with the data in the model(s) in question, to produce some output file. This file can be a code file like C#, Java, HTML or just pure text controlled by the file extension you decide and our course the text itself. The other option is to perform transformations the same way that `Cell iCellValues` are set in the first place, by editing the model directly. As it is model to model transformation we are interested in, the second option is our main interest. The model to model transformation is handled in the same way as the Solve/execution part described in chapter 4.5.2.

Sort the first row

To sort the first `Row`, all `iCellValues` from this `Row` are collected. If the `iCellValue` of the first `Cell` has `iCellValue` 9, then all `iCellValues` in the `Puzzle` that have this value will get the value 1 instead. By performing this change for all `Cells` in the first `Row`, changing values, the result will be that the first `Row` has the values from 1 to 9 and as all values are changed accordingly, thus the Sudoku is still valid. This solution is not as abstract as the expected solution presented in the beginning of this chapter, and also contains a lot of extra code besides the actual sorting. See Code example 14 for the sorting code. All `iCellValues` from the first `Row` are stored in `int iCellValueX` where `X` refers to the `iCellValues` original position in `Row 1`. Then for all the `Cells` in each `Row` the new `iCellValue` is set so that if `iCellValue = iCellValueX`, then `iCellValue = X`.

```
//Code omitted
foreach (Field field in puzzle.Elements){
      if (field is Row){
            countrow++;
            if (countrow == 1){
                  foreach (Cell c in field.cellsInField){
                  //Retrieve all iCellValues from row 1
                        countcell++;
                        if (countcell == 1)
                        {iCellValue1 = c.iCellValue;}
                        if (countcell == 2)
                        {iCellValue2 = c.iCellValue;}
                        if (countcell == 3)
                        {iCellValue3 = c.iCellValue;}
                        if (countcell == 4)
                        {iCellValue4 = c.iCellValue;}
                        if (countcell == 5)
                        {iCellValue5 = c.iCellValue;}
```

```
                        if (countcell == 6)
                        {iCellValue6 = c.iCellValue;}
                        if (countcell == 7)
                        {iCellValue7 = c.iCellValue;}
                        if (countcell == 8)
                        {iCellValue8 = c.iCellValue;}
                        if (countcell == 9)
                        {iCellValue9 = c.iCellValue;}
                  }
            }
            foreach (Cell c in field.cellsInField){
                  using (Transaction t =
                              store.TransactionManager.BeginTransaction("")){
                        //Set the correct new iCellValues
                        if (c.iCellValue == iCellValue1)
                              c.iCellValue = 1;
                        else if (c.iCellValue == iCellValue2)
                              c.iCellValue = 2;
                        else if (c.iCellValue == iCellValue3)
                              c.iCellValue = 3;
                        else if (c.iCellValue == iCellValue4)
                              c.iCellValue = 4;
                        else if (c.iCellValue == iCellValue5)
                              c.iCellValue = 5;
                        else if (c.iCellValue == iCellValue6)
                              c.iCellValue = 6;
                        else if (c.iCellValue == iCellValue7)
                              c.iCellValue = 7;
                        else if (c.iCellValue == iCellValue8)
                              c.iCellValue = 8;
                        else if (c.iCellValue == iCellValue9)
                              c.iCellValue = 9;
                        t.Commit();
}     }     }     }
//Code omitted
```

Code example 14 Visual Studio: Sort the first row

Transpose

One problem with transposition in Visual Studio is that a displayed model element is both the model element and its corresponding shape. To perform a correct transposition, the model elements corresponding shape should be moved while the `Cell` model elements must be “moved” to other `Rows`, `Columns` and `Boxes`. Because these issues make transposition rather complex, I will implement transpose in the same way as the permutations mentioned above, by just changing the `iCellValue` of all `Cells` in the `Puzzle`. All `iCellValues` from the `Cells` in all `Rows` are first stored as List<int> `RowX` where `X` denotes the `Row` number. Then all `iCellValues` are changed in such a manner that the `iCellValue` in the `Cell` at `RowX[Y]` now hold the `iCellValues` that were in `RowY[X]` originally. This means that the value from `Row` 2 position 1 is now in `Row` 1 position 2. Code example 15 on the next pages is from the transpose code in the Visual Studio solution.

```
//Action for Transpose button
private void btnTranspose_Click(object sender, EventArgs e){
    int countrow = 0;
    int countcell = 0;
    List<int> Row1 = new List<int>();
    //Code omitted… 9 rows
    List<int> Row9 = new List<int>();
    //Retrieve puzzle
    Puzzle puzzle = this.docView.CurrentDiagram.ModelElement as Puzzle;
    Store store = puzzle.Store;
    //Retrieve all cell values
    foreach (Field element in puzzle.Elements){
        if (element is Row){
            countrow++;
            foreach (Cell c in element.cellsInField){
                if(countrow == 1)
                    Row1.Add(c.iCellValue);
                else if(countrow == 2)
                    Row2.Add(c.iCellValue);
                else if(countrow == 3)
                    Row3.Add(c.iCellValue);
                else if(countrow == 4)
                    Row4.Add(c.iCellValue);
                else if(countrow == 5)
                    Row5.Add(c.iCellValue);
                else if(countrow == 6)
                    Row6.Add(c.iCellValue);
                else if(countrow == 7)
                    Row7.Add(c.iCellValue);
                else if(countrow == 8)
                    Row8.Add(c.iCellValue);
                else if(countrow == 9)
                    Row9.Add(c.iCellValue);
     }  }  }
    countrow = 0;
    //transpose by switching cell values
    foreach (Field element in puzzle.Elements){
        if (element is Row){
            foreach (Cell c in element.cellsInField){
                using (Transaction txCreateElem =
                store.TransactionManager.BeginTransaction("Create elements")){
                    countcell++;
                    if (countcell == 1)
                        c.iCellValue = Row1[countrow];
                    if (countcell == 2)
                        c.iCellValue = Row2[countrow];
                    if (countcell == 3)
                        c.iCellValue = Row3[countrow];
                    if (countcell == 4)
                        c.iCellValue = Row4[countrow];
                    if (countcell == 5)
                        c.iCellValue = Row5[countrow];
                    if (countcell == 6)
                        c.iCellValue = Row6[countrow];
                    if (countcell == 7)
                        c.iCellValue = Row7[countrow];
                    if (countcell == 8)
                        c.iCellValue = Row8[countrow];
                    if (countcell == 9)
                        c.iCellValue = Row9[countrow];
```

```
                    txCreateElem.Commit();
                }
            }
            countcell = 0;
            countrow++;
}   }   }
```

Code example 15 Visual Studio: Transposition

4.7 Results summary

The results in this project are unavoidably software dependent. I have successfully covered the structure, constraints, graphical editor, transformation and run aspects of a meta-model/language in the Visual Studio based solution, while the Eclipse based solution has covered structure, constraints, textual representation, graphical representation and transformation.

5 Discussion

The discussion covers the results achieved in this project as well as my experiences with the different tools used. In addition, small “user guides” are provided covering issues not well documented elsewhere as well as installation procedures. There is also some discussion on my personal experiences, problems and what I would have done differently were I to start a project like this again. I could not find any earlier solutions of modeling Sudoku in the way that I have done in this project, so there is no real comparison to earlier work in this area.

I find it somewhat difficult to place this Sudoku meta-model at a specific meta-level in the MOF sense. Some parts of the solution can definitely be called meta-model, but others on the other hand might “just” be a model. This is perhaps hard to avoid in a project like this as the example is so very concrete. An example is OCL, that is used both to describe the constraints as well as being used directly in the code. A Sudoku puzzle created in one of the editors is really a model instance, and then one can argue that the level above is the model, see 3.4.3. As mentioned earlier in this report, the Sudoku meta-model is a meta-model in a wider sense than e.g. MOF or UML. In addition to structure many other aspects are covered as shown in this report.

The tool specific solutions are sometimes very different from the ideal solutions presented at the beginning of each of the solution chapters. It is fair to say that the tools force the developer(s) to do things the way that fits each specific tool. We have seen this even for OCL, even though this is an OMG standard one often need something more, e.g. Java code, to implement it.

Some of my code could definitely be more efficient, especially the code from in Visual Studio. It could be more dynamic thus also making it easier to implement support for Sudokus of different sizes or that solves on other terms. In some situations I decided to run constraints on Row/Column/Box where Field could be used thus reducing the amount code. I chose to use Row/Column/Box in these situations in order to be able to provide the users with more specific error messages. I also wish I never started to use the term Puzzle, but used Sudoku instead. Initially I did this to avoid confusion between this Sudoku meta-model and a real Sudoku, but I think I should have used Sudoku after

all as it is the more obvious choice. In the end it is Sudokus that are created with the editors.

5.1 Results discussion: Visual Studio

It is very easy and straightforward to create the structure in Visual Studio. However it differs in some way from the MOF compliant model in the sense that DomainRelationships are created automatically. Implementing constraints was very easy with Visual Studio, and all constraints are successfully implemented and functional. The execution part was also relatively easy to implement in the Visual Studio solution. Visual Studio even provides the opportunity to create buttons and add functionality to them resulting in buttons for Solve, Clear and transformations. The Visual Studio solution provides everything except a textual editor in one single solution.

5.2 Results discussion: Eclipse

The Sudoku structure created using the EclipseUML plug-in is very similar to the MOF compliant structure. The textual editor created in TEF works very well. It allows the user to write a Sudoku constrained by OCL statements. If there is an error, the user is informed via error messages. I considered using a grammar that only allowed exactly 9 `Rows` with exactly 9 `Cells`. However I decided that for future work this would be a waste as the current grammar allows Sudoku of different sizes; the size is constrained by OCL. The editor created in GMF has very limited functions, it simply lets the user edit the `Cell iCellValues` and informs of violated constraints. I have not succeeded in assigning the correct location for Row elements when added to the model. I had trouble implementing the necessary execution behavior in the Eclipse based solution. The problem was concerning compatibility between OCL and the Ecore feature of EMF as the OCL does not directly support the EEList attribute. This was not a problem with MDT OCL as this is automatically taken care of as MDT OCL maps Ecore's EEList to OCL's Sequence.

5.3 Tool experiences

Working through as many different tools as I have done during this project, is bound to leave some preferences and opinions on advantages and disadvantages of the different tools. In this chapter I will describe as best as I can how I experienced working with the different tools, and in some cases how I solved problems that I found hard or impossible to solve or even find solutions to.

5.3.1 Installing and using Visual Studio with SDK

Installing Visual Studio is a pretty straight forward windows install, but it takes some time as it is very large. Since I am using Windows Vista I needed an update but this was handled semi-automatically by Vista and Visual Studio and should not present any problems.

Building the structure (DomainModel) in Visual Studio is very easy for the experienced VS-user. One can simply drag and drop classes (DomainClass) onto the design surface and easily create relationships (DomainRelationship) between these classes. All necessary tools are available in a toolbox. The walkthroughs available at the msdn [49] are very informative and easy to follow.

The experience of working with Visual Studio is getting better for every time I try. I used VS with DSL Tools in a project in 2006/2007, and struggled a bit. I did not accomplish much more than creating a structure and adding constraints to it. This time around the whole process was easier. It is a very comprehensive tool, and sometimes it is easy to get lost in all the available possibilities. One advantage with VS is that it in general has more users than e.g. many of the open source plug-ins available for Eclipse. This makes it easier to find related work as well as forums with useful information on usage and possibilities as well as limitations. Another advantage of VS is the possibility to have several aspects of the language in one application. Structure, graphical editor, constraints and execution and transformations are neatly packed in a single project.

Adding a List<int> DomainProperty to a DomainClass

As mentioned in chapter 4.5.2 DSL Tools does not support List attributes. However, there is a nice workaround that I found at [50] that describes how to add a new external type to the model, allowing types that are not supported by default. I have not found any other description of how to achieve the possibility of other attributes (domain properties) than the ones provided by default. A short description of the procedure follows:

1. In your DSL explorer window, right click the top node and select Add New External Type.
2. In the properties window for your new external type, set the “Name” property to List<int> and the “Namespace” property to System.Collections.Generic.
3. Provide custom storage for your new type as shown in Code example 16.

```
partial class Cell
    {
        List<int> iPossibleCellValuesPropertyStorage = new List<int>();
        public List<int> GetiPossibleCellValuesValue(){
            return iPossibleCellValuesPropertyStorage;
        }
        public void SetiPossibleCellValuesValue(List<int> value){
            if (null != value)
            {
                iPossibleCellValuesPropertyStorage = value;
            }
        }
  }
```

Code example 16 Visual Studio: Custom storage of custom external type

You are now ready to use your new External Type. The same procedure can be used to create other types, e.g. if you need a list if Strings or such. I assume also other types, like Array can be added this way but I have not tried this as it is not necessary for this project.

Editing diagram template files

In order to have the DSL present the user with a “ready to use” Sudoku puzzle at the creation of a new diagram, I had to edit the diagram files shown in Figure 20. The files “sd.diagram” and “sd.sd” are the two files that determine what a new diagram (Sudoku) looks like when it is created. These are XML files. In order to set up these files to suit my needs, I first created the Sudoku with `Rows` and `Cells` in the debugging mode. I then copied the two files from the debugging mode solution to these template files and finished them by setting the correct relationships manually in the code. Some problems are handled by custom rules as explained in a previous chapter. Thus when a new diagram is created, it is already filled with 9 `Rows`, `Columns` and `Boxes`, 81 `Cells` and relations between `Cells` and the necessary `Fields`.

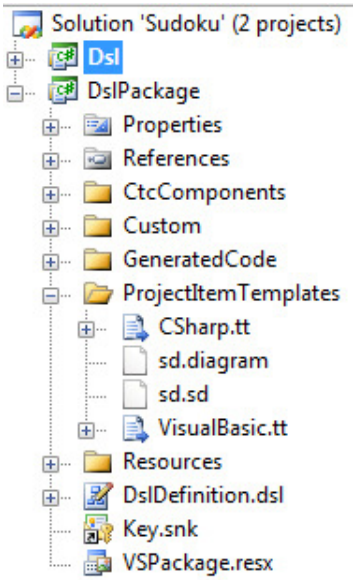

Figure 20 Template diagram files sd.diagram and sd.sd

5.3.2 Installing and using Eclipse

I have used Eclipse on some smaller projects before, but most of the eclipse plug-ins used in this project were new to me. They are all very different from each other in both use and function. I learned quite a few new and very handy tricks working with Eclipse this time, and I am sorry to say that if I had know about them before I would have saved a lot of time.

One side of Eclipse that has caused a bit of frustration for me has been the processes of installing all of the plug-ins. I am now working with four different Eclipse installations and workspaces, as some installations are very unstable and crash several times per hour. This could be due to unstable plug-in combinations.

Using the Update Manager in Eclipse

In Eclipse it is very easy to update software / download new plug-ins using the update manager. Please follow these steps, provided you have an URL for an update site for the desired plug-in.

1. In Eclipse, click the “Help” menu item.
2. Select “Software Updates”.
3. Select “Find and Install”.
4. Select “Search for new features to install” (assuming you are installing a new feature).
5. Select “New Remote Site” and enter the update site URL and a name (any name).

6. Make sure that the entry you created is checked and press finish.
7. Accept licenses and finish.
8. Remember to restart Eclipse for your new plug-in to function.

5.3.3 Installing and using Omondo EclipseUML

The EclipseUML plug-in from Omondo is very easy to install and use. However, the free edition seems to be somewhat unstable even though the only apparent difference between the Studio edition and Free edition is the support for CVS. A very good tutorial for EclipseUML is available at [39]. The editor works as most other UML editors I have tried and if you are familiar with Eclipse and UML this should be quite simple.

My only negative thoughts on EclipseUML are that my Eclipse installation with EclipseUML and oAW became very unstable independent of which plug-in I was working with. The most common problem I had with EclipseUML was Eclipse crashing and the only error message I got was the one in Figure 21. The log did not help.

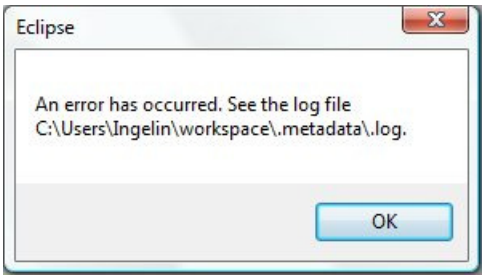

Figure 21 EclipseUML error

5.3.4 Installing EMF, GEF, MDT OCL, GMF, UML

The EMF, GEF, OCL, GMF and UML plug-ins can be installed using the procedure described in chapter 5.3.2. However, instead of selecting “New Remote Site” in step 5, check the “Europa Discovery Site” and press finish. If a question about a mirror site pops up, press yes/OK. Expand the “Europa Discovery Site” tree, and select the features you want to install. This is the part where things often get tricky due to dependencies between some packages. After trying (and failing) many times, this is the best solution I came up with regarding which packages to install:

Enabling features:
Select any entry with EMF and/or OCL
Graphical Editors and Frameworks:

Check this node for GEF

Models and model development:

Eclipse Modeling Framework Runtime

Eclipse Modeling Framework Runtime Extender SDK

EMF Data Integrity Frameworks (for GMF)

Graphical Modeling Framework (Europa Edition)

Object Constraint Language 2.0 End-User Features

Object Constraint Language Extender SDK

UML2 End-User Features

UML2 Extender SDK

UML2 Tools

UML2 Tools SDK

This download sometimes takes a whole day.

5.3.5 Installing and using EMF

See 5.3.4 for installation procedure of EMF. I used EMF for the modeling in the Eclipse based solution in this project. The Ecore model can be created with EMF using Java, or the built in Ecore diagram editor. There are several tutorials available that are very helpful.

EMF to XMI/XML

In order to use the model from e.g. TEF in a medini QVT transformation, I needed to serialize the EMF model to XML. Code example 17 shows code for storing a `Puzzle` (or any Ecore model) as an XMI file.

```
//Store the ecore model as an xmi file
      public void serializeEcore(Puzzle puzzleToSerialize){
            // create resource set and resource
            ResourceSet resourceSet = new ResourceSetImpl();
            // Register XML resource factory
            resourceSet.getResourceFactoryRegistry().
                                    getExtensionToFactoryMap().
                                    put("xmi", new XMIResourceFactoryImpl());
            Resource resource =
           resourceSet.createResource(URI.createFileURI("c:/temp/sudoku.xmi"));
            // add the root object to the resource
            resource.getContents().add(puzzleToSerialize);
            try {
                  resource.save(null);
            } catch (IOException e) {e.printStackTrace();}
      }
```

Code example 17 Eclipse: Store Ecore model as XMI

5.3.6 Installing and using oAW

Installing oAW always presented some problems. However, the problem was not always the same. My somewhat strange recommendation is to use the oAW Update Manager site (see [55]) together with the Europa Discovery Site. When you select to install oAW, you can use the Europa Discovery Site to add elements if necessary. openArchitectureWare 4.2 was built for use with Java 5/6 and depends on Eclipse 3.3, EMF 2.3, UML2 2.1 and GMF 2.0 (Europa releases). Do not use older versions. Get EMF, UML2 and GMF from the Eclipse Europa Discovery Site (this is preconfigured in Eclipse 3.3's update manager) as explained in chapter 5.3.2 and 5.3.4.

On paper, oAw and xText looks great. My Sudoku grammar compiled without errors and all was well. However I had serious problems when it came to testing the editor and I could never actually run the editor at all, thus preventing me doing the necessary tests and completing my work with oAW. In addition to this I had some strange issues if there was an Eclipse crash (which happened quite often). In one case I had 65 errors when Eclipse crashed, and when I started Eclipse again without changing any code, the number of errors was 27. Another problem was a bundle issue and this is one of the problems that really made us walk away from oAW in this project. After looking for help on the web (see [54]) I discovered that I was not the only one with this problem. I also experienced that my Eclipse environment became very unstable and I had to do several new Eclipse installations. After my supervisor Terje Gjøsæter encountered the same problems as I did, we decided to leave oAW and try out another framework for textual editor development. I spent about a month working with openArchitectureWare.

5.3.7 Installing and using TEF

The TEF download is fairly simple and you can use the procedure described in chapter 5.3.2. The procedure is also described in the download section on the TEF website [40] where the URL for the update site also can be found. A tutorial is available at [41]. Completing this tutorial results in an example solution one can use as a starting point for a new editor.

The elements that are represented in the grammar will result in model elements being created when the language is used. In this project this means that `Puzzle`, `Row` and `Cell` elements are created when a Sudoku is created. `Column` and `Box` elements on the other hand must be created manually as they are not represented in the grammar. This

is done using EMF and some code examples are presented in chapter 4.3.2. TEF is rather easy to use and the creator is very helpful if asked for help. A downside with TEF is that it is quite new (first release in August 2007) and thus the available documentation and resources are minimal. The user group seems small and there are no available discussion forums as of now (March 2008).

To add custom code to your TEF project, this code can be added to the classes check method in template files as explained in the tutorial. This means code for constraints, model elements and relationships and so on.

A new TEF version was released on 20.03.2008, very late in the time span of this project. The syntax definition has changes as well as libraries, file extensions and so on. Unfortunately I was not able to cover the new version in this project.

5.3.8 Installing and using MDT OCL

MDT OCL is installed as described in 5.3.2, see also [36] for the update sites. For a very good tutorial on working with OCL along with EMF see [37]. With this tutorial MDT OCL was very easy to work with, however you must be familiar with the OCL.

5.3.9 Installing and using GEF

To install GEF use the update manager in Eclipse and select the Europa Discovery site. See chapter 5.3.2 for further instructions.

I found it very hard even getting started with GEF. The example solutions, even if they are “simple” ones, are very complex and there is a lot of information to handle. I have yet to find a complete tutorial explaining all sides of an example. Even a small example requires an enormous amount of code and files, resulting in the process being very time consuming. I tried to follow several tutorials, and all of them got me to a certain point and then they stopped, assuming the reader could complete on its own. After conferring with Merete Skjelten Tveit, a Ph. D student at UiA, I decided to try out GMF first as it seems like a GEF solution will be too time consuming to be a part of this project

5.3.10 Installing and using GMF

To install GMF, use the update manager in Eclipse and select the Europa Discovery site. See chapter 5.3.2 for further instructions.

For first time users of GMF I strongly advice following the mindmap tutorial available at [33]. This tutorial covers many topics and explains the process as well. This tutorial is also available as a “Cheat Sheet” in Eclipse (Menu item Help -> Cheat Sheets) if you used the update manager for your installation. I started trying out GMF without this tutorial the first time, and after a while I realized it was better to spend some time on the tutorial first as I was getting near nowhere on my own. After the tutorial I had a much better understanding of GMF and it was easier to work with as I now had the basic knowledge to start working on the Sudoku graphical editor. For this new editor I could even follow the mindmap tutorial for a while with minor adjustments to fit my model.

A frustration with GMF is the lack of useful and user-friendly documentation beyond the tutorial. As an example, for the initial model I needed for the graphical editor, this process proved to be very easy and straight forward. The problem was just finding out where to put the custom code to create the model. For newcomers to GMF the generated code is not always easy to understand, thus this process might be very time consuming. For experienced users that are familiar with GMF and its architecture, this should not be very problematic.

Creating the initial model in GMF

To create an initial model that will be present in a new Sudoku editor, you must add your EMF model code to the code files GMF created. In the somename.diagram.part package in the diagram part of the project files, there is a file called somenameDiagramEditorUtil.java. Open this file and find the createInitialMode()l method. Before you add your code make sure to set the method *@ generated NOT* so that your custom code will not be overwritten the next time the code is regenerated. You are now ready to put in your custom code for the initial model you want. Note that this only adds model elements, not graphic elements. The graphic elements are created and mapped to model elements as explained in the mindmap tutorial.

Adding constraints in GMF

The generated code for constraints has a dependency on the EMF Data Integrity Frameworks from the Europa update site. It will not be selected automatically when installing GMF, so you will need to explicitly install it. For the installation procedure, please take a look at chapter 5.3.2.

To add constraints to your GMF editor, please follow these steps (see [33]):

1. Open the mapping definition (somename.gmfmap) and right-click the Mapping node.
2. Select New Child -> Audit Container and give it a name, ID and description.
3. Select the container you just added, and add to it a new Audit Rule. Give the new rule an appropriate name.
4. To add a target for your Audit Rule, right click the rule and select Domain Element Target. As Element, select the class you want to be the target of your rule.
5. Add a new child Constraint to the Audit Rule and enter some OCL statement for the Body, leaving the Language set to OCL.

After regenerating somename.gmfgen model, you must set the Validation Enabled property of the Gen Diagram element to true in order for your new audit rule to be executed.

5.3.11 Installing and using medini QVT

medini QVT is provided as a complete Eclipse Installation that includes medini QVT. Currently it is not provided as a separate plug-in. The install is very easy and the installation also includes two model-to-model transformations in QVT. A video tutorial is available on the medini website (see [44]). Please note that this video shows only how to use the medini QVT tool, it is not a QVT tutorial.

Finding useful and relevant QVT tutorials proved to be more difficult than expected. I ended up using the OMG QVT specification even though it is a bit advanced for someone who has no experience with QVT at all. Using medini QVT was mostly quite easy, even though at some times I had problems understanding how medini QVT processed the transformation files. The only available documentation seems to be the video mentioned above, but I expect more detailed tutorials to be available at a later time. At the medini QVT website, there is a discussion forum; however the activity is somewhat low.

My first attempt (see Code example 18) on writing a transformation in QVT ended in deletion of `Cell` values from the `Cells` I wanted to transform.

```
transformation valueTransformation(source : newstructure, target:
newstructure){
top relation change1to2{
            checkonly domain source newstructure:Cell { iCellValue = 1 };
            enforce domain target newstructure:Cell { iCellValue = 2 };
      } }
```

Code example 18 Eclipse: First medini QVT attempt

After many attempts on transformations I finally came to some answers. I initially tried to perform the transformation in fewer steps, e.g. transforming x to m, y to x and m to n. However this resulted in an infinite loop and Vista crash, so in the end I ended up with 4 “steps” after all.

From April 2008 the medini QVT is available as a separate plug-in that can be installed into your existing Eclipse.

5.4 Software discussion and evaluation – Eclipse vs. Visual Studio

We can also call it free vs. expensive, open source vs. commercial closed source. Eclipse and Visual Studio are definitely very different environments to work with. Apart from the obvious differences as using different programming languages, there are many other differences as well.

One definite advantage of using Visual Studio in my experience is that everything is available to the user/developer in one single solution. E.g. for this project: structure, constraints, graphical editor, execution and transformation are all available in one single project/solution. Using Eclipse, most of these features depend on some plug-in, thus to cover all these aspects several plug-ins must be installed. Each plug-in might require a specific kind of project or setup, and one project actually ends up with e.g. 4 projects: the main project, the designer project, the editor project and so on. Things get messy and confusing.

As Eclipse plug-ins are mostly open source and also usually have different developers, they do not always function well together. However the idea and purpose of Eclipse is to use it with plug-ins and this leaves the developer with the privilege of choosing the

preferred plug-ins that fit her project best. For example when several tools for creating textual editors are available, one can choose to work with the one that fits the current need or seems to function better in every project. Using Visual Studio, this is not an option. The amount of plug-ins available for Eclipse are increasing every day, making even more functionality available to developers. As one can decide which plug-ins one wants to use, it is possible to keep Eclipse installations very clean and free from unnecessary elements. This is more restricted in Visual Studio, and it does for example not even provide a framework for creating textual editors. The amount of available features in Visual Studio is not increasing at the same speed as it is for Eclipse. I assume this is because Visual Studio is a large system without available source code for any developer to play with. This limits additions mostly to the ones that come from Microsoft, after a long period of decision-making, meetings and testing.

Another advantage of the nature of Eclipse and its plug-ins is that one stands more free to choose from several languages. For example in TEF, one can implement constraints using both Java by itself or Java and OCL. When using Microsoft Visual Studio one is more bound to using Microsoft programming languages (e.g. C, Visual Basic, C#). On the other hand, it can be an advantage for both Visual Studio and Eclipse that one can create a complete DSL using only one programming language like C# or Java. An advantage of Visual Studio is that one can do so without demanding the installation of many plug-ins. This can be an advantage especially for beginners in the world of domain-specific languages who are familiar with Visual Studio.

Both Eclipse and Visual Studio tend to be somewhat unstable. Often when I want to create a new item in Visual Studio, e.g. a Class, Visual Studio crashes and I have to start it all over. The same problems happen with Eclipse; I had a lot of trouble when I was working with EclipseUML and oAW in the same installation. My general impression is that not all plug-ins work well together, while others interact fine.

When it comes to documentation and tutorials, there is a big difference between Eclipse and Visual Studio. The support for Visual Studio is available at the Microsoft developer network (msdn) where all documentation and tutorials is available. The network is easy to use and has a very user friendly structure. The tutorials are explanatory and use a step-by-step pattern that is easy to follow. In addition the forums are good sources of help from many other users. These forums are particularly useful when one needs to

solve a problem not directly supported by Visual Studio/DSL Tools. When using Eclipse, the quality of tutorials and documentation vary. As mentioned, I failed to find a GEF tutorial that guides you through a complete example. GMF on the other hand provides a very good tutorial. For TEF, the available documentation was rather scarce. The forums available at the Eclipse website are pretty good but mainly for plug-ins provided by/through the Eclipse project. I assume these differences are partly because for Visual Studio, people get paid to create these tutorials and such things while the Eclipse plug-ins are in many cases created by someone with a passion for the area of work but who does this on her spare time. Some plug-ins have one single creator while others have large teams. However, I also had some trouble with the DSL Tools documentation for Visual Studio. From its first release, DSL Tools has changed. This is no surprise; development software is under continuous updating. The problem is when an update removes features or changes the way they are used, as this information is not always available on places it should be, resulting in frustration and build errors.

	TEF	GMF	GEF	EMF	EclipseUML	medini QVT	Visual Studio
Structure	Ecore/EMF	Ecore/EMF		5	5	Ecore	5
Constraints	5	5		5			5
Textual Rep	5						NA
Graphical Rep		4					4
Run/Execution							5
Transformation						5	3
Tutorials	3	4	2	5	5	4	5
User friendliness	4	4	2	4	3	4	5
Installation	5	6	6	6	6	6	6

Table 1 Tool evaluation form

I find it very difficult to make a fair comparison of Eclipse and Visual Studio as they both have advantages and disadvantages. Their value must be considered for each project individually. I have to say that Visual Studio was easier to work with, especially as everything is available in a single project. On the other hand Eclipse has so many possibilities because of its nature with plug-ins. I have put a small comparison summary in Table 1, where marks are given from 1 to 6 where 6 is best.

5.5 What I have learned

I think most students have many expectations about how their experience working on a master thesis will be. I can definitely say that my experience is far from what I expected. I

have always been a “good” student, delivering my homework on time and getting good grades. I expected this to automatically follow me during my master thesis as well, but I was wrong. For the first time in my life I really hated working alone. After one year of maternity leave I started working on my meta-modeling master thesis all by myself. Normally this would not be a problem. But after being ill for longer periods myself as well as taking care of a baby who was ill as well, my motivation came to an all-time low. As the good student I had been before I never needed a project plan, so I didn’t have one when I started working on this project. This proved to be a big mistake when things became tough. Luckily for me, I had a very good supervisor who several times put be back on track, and had me create a project plan of progress. I have now learned that having such a plan is very important the second one falls behind, and from now on I will always have a detailed plan for my projects. I have learned the value of having teammates, and I think I have learned a lot about myself, handling obstacles and not to give up.

When I started this project I had limited knowledge about meta-modeling, and I had definitely never heard of e.g. QVT. My only experience in this area of work when I started this project was a smaller previous project on Visual Studio as a tool for domain-specific language development. It is fair to say that I have gained an enormous amount of new knowledge during my work on this project. In my opinion, the area of meta-modeling is much about achieving a general understanding. The concept is very abstract and the way of thinking must be learned and this takes a lot of time. This understanding is important when it comes to seeing the possibilities that are available in the different tools and standards. After my work on the Sudoku meta-model I feel that I have achieved a much better understanding of meta-modeling even though some concepts still seem a bit hard to comprehend. I have created textual and graphical editors, transformations and constraints using both new and familiar tools. I have learned the importance of project plans as well as how great it is to have a project partner, and how frustrating it can be not having one.

Many times I have asked myself if this was the right project for me. My answer is usually yes, although I really wish I had a partner. Working alone on this type of project without someone to discuss with, especially in the development processes has been challenging.

5.6 Problems

The problems I encountered during my project period were mostly tool related. Working with new tools that have small user groups can be quite frustrating. Some of the plug-ins did not work very well together, for example oAW and EclipseUML. In addition the plug-ins are not always stable and have some bugs. Combined with rather poor documentation this can be quite hard to work with.

5.7 What I would have done differently

If I were to start a project like this again, I would do a better job with literature from the beginning. I had for example never heard of QVT before, and deeply regret that I did not introduce QVT to myself at an earlier stage of the project, allowing it to sink in and achieve an early understanding of the concepts of QVT. In addition this would have helped in getting a general overview for creating a realistic and more complete project plan earlier. I dare to say this is a generally good idea starting on any project that introduces one to new and unfamiliar subjects. In addition, as mentioned, I would create a detailed plan of progress/Gantt schema at the very beginning. Falling behind without knowing how far behind is never a good thing and is likely to cause problems at some time in the process. The sooner the better, I say, so let's have a plan. I could also have done a better job documenting my findings during the entire process. I am not sure if this is a very good solo project (at least not after one year's leave). The area of work is very abstract in many ways and I often missed a partner to discuss problems and possible solutions with.

5.8 Future work

There is definitely room for improvement in this solution. It could be more dynamic in the sense that it could support Sudoku of different sizes as well as ones solving on other terms than integers. Examples are symbols, colors and so on. Non-square solutions are also possible. It could be interesting to try more tools, e.g. successfully use oAW, or give Kermeta a try. When it comes to the transformations, I suspect a solution that supports the Operational Mappings language of QVT could be very interesting. Finally it would be great to have more solving strategies implemented, or any solving strategy would be good in the Eclipse based solution.

6 Conclusion

This project aimed at modeling Sudoku by the aspects of a meta-model/language, hoping to provide the SMILE project with Sudoku as an executable specification as well as evaluating the different tools used to benefit others working in the same area.

I solved this problem by working on one Eclipse based solution and one solution in Visual Studio. Both environments have their advantages and disadvantages, and are very different from each other. I did my best to cover all aspects of a meta-model/language in both environments.

The outcome of this project does not completely fulfill all requirements as I failed to implement a solving strategy in the Eclipse based solution and the textual editor was not supported in Visual Studio. I am, however, very happy with my results, as this was a complicated project with many areas, many of them new to me, to cover. I successfully implemented 5 of 6 aspects for each of the two solutions, and the problems I encountered were mainly tool related. I have learned a lot in the area of work as well as about working on a larger project and I have a much better understanding of meta-modeling than I had a few months ago. I can provide the SMILE project with insight into several tools, and an example of what one can accomplish by applying the SMILE methodology. The results of this project are several plug-ins to support the creation of Sudoku by textual and graphical editors, as well as some tutorial material.

When it comes to choosing software for projects like this one, my conclusion is that one should look into and consider several tools, and select the one that seems to meet the demands of each project and fits the developer(s) better.

Appendices

Appendix 1 Glossary & Abbreviations

BNF - Backus Naur Form
DSL - Domain Specific Language
EMF - Eclipse Modeling Framework
GEF - Graphical Editing Framework
GMF - Graphical Modeling Framework
MDA - Model Driven Architecture
MDD - Model Driven Development
MDT - Modeling Development Tools
MOF - Meta Object Facility
msdn - Microsoft developer network
oAW - openArchitectureWare
OCL - Object Constraint Language
OMG - Open Management Group
PIM - Platform Independent Model
PSM - Platform Specific Model
QVT - Query View Transformation
SDK - Software Development Kit
SQL - Structured Query Language
TEF - Textual Editing Framework
UML - Unified Modeling Language
UiA - University of Agder
XMI - XML Metadata Interchange
XML - Extensible Markup Language

Appendix 2 References

[1] Hanne D. McBride, “Løs og lær Sudoku”, Arneberg forl., 2005

[2] Sudoku Central, “What is Sudoku: rules, history and terminology”, see also http://www.sudokucentral.com/what-is-sudoku Last accessed 2008-04-29

[3] “A Su Doku solver”, see also http://www.act365.com/sudoku/ Accessed 2007-01-03

[4] Angus Johnson , “Solving Sudoku”, see also http://www.angusj.com/sudoku/hints.php Last accessed 2008-04-29

[5] “Sudoku”, see also http://www.intosudoku.com/ Last accessed 2008-04-29

[6] David C. Lay, “Linear Algebra and its applications”, Addison-Wesley, third edition, 2003 p 114-115

[7] Jean-Paul Delahaye, “The Science behind SUDOKU”, Scientific American, June 2006 p. 80-87

[8] Vegard Hanssen, ”Sudoku oppgaver”, see also http://www.menneske.no/sudoku/ Last accessed 2008-04-29

[9] Ed Pegg Jr , “Math Games, Sudoku Variations”, The Mathematical Association of America, editorial, September 6 2005, see also http://www.maa.org/editorial/mathgames/mathgames_09_05_05.html

[10] Wolfram MathWorld, ”Transpose”, see also http://mathworld.wolfram.com/Transpose.html Last accessed 2008-04-29

[11] Agnes M. Herzberg and M. Ram Murty, “Sudoku Squares and Chromatic Polynomials”, Notices of the AMS, Volume 54, Number 6, June/July 2006, see also http://www.ams.org/notices/200706/tx070600708p.pdf

[12] NIKOLI Co.,Ltd., “WebNikoli”, see also http://www.nikoli.co.jp/en/ Last accessed 2008-04-29

[13] Tony Clark, Andy Evans, Paul Sammut and James Willans, “Applied Meta-modelling”, Xactium, 2004

[14] OMG, “UML2.0 OCL Specification”, 2003, see also http://www.omg.org/docs/ptc/03-10-14.pdf

[15] Martin Fowler, “UML Distilled: A Brief Guide to the Standard Object Modeling Language”, Addison-Wesley Professional, third edition, 2003

[16] OMG, “The Object Management Group (OMG) “, see also http://www.omg.org/

[17] OMG, “Introduction to OMG's Unified Modeling Language™ (UML®)”, see also http://www.omg.org/gettingstarted/what_is_uml.htm

[18] OMG, “OMG's Meta Object Facility”, see also http://www.omg.org/mof/

[19] OMG, “Meta Object Facility (MOF) Core Specification”, Version 2.0, 2006

[20] OMG, “Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification”, 2007

[21] OMG, “MDA Guide version 1.0.1”, June 2003

[22] OMG, “The Architecture of Choice for a Changing World”, see also http://www.omg.org/mda/

[23] SMILE, “The SMILE project”, see also http://osys.grm.hia.no/osys/projects/smile

[24] SMILE, “Specific Targeted Research Project Proposal (STREP)”, see also http://gullfisk.agder-ikt.hia.no/osys/smile/Documents/SMILE_Call5_V10.doc

[25] Jan P. Nytun, Andreas Prinz, and Merete S. Tveit , Automatic generation of modelling tools, Springer Berlin / Heidelberg, 2006.

[26] Andreas Prinz , Markus Scheidgen and Merete S. Tveit , “A Model-Based Standard for SDL”, Springer Berlin / Heidelberg, 2007.

[27] Guido Wachsmuth , “Modelling the Operational Semantics of Domain-Specific Modelling Languages”

[28] Xactium, see also http://xactium.myzen.co.uk/

[29] Eclipse, “What is Eclipse”, see also http://help.eclipse.org/help33/index.jsp?topic=/org.eclipse.platform.doc.isv/guide/int_eclipse.htm Last accessed 2008-04-29

[30] Eclipse, ” The Eclipse Modeling Framework (EMF) Overview”, see also http://dev.eclipse.org/viewcvs/index.cgi/org.eclipse.emf/org.eclipse.emf/doc/org.eclipse.emf.doc/references/overview/EMF.html?root=Modeling_Project&view=co

[31] Eclipse, “ Eclipse Modeling Framework Project (EMF)”, see also http://www.eclipse.org/modeling/emf/

[32] Eclipse, ”What is GEF?”, see also http://www.eclipse.org/gef/overview.html

[33] Eclipse, “GMF mindmap tutorial”, Eclipsepedia, see also http://wiki.eclipse.org/index.php/GMF_Tutorial Last accessed 2008-04-29

[34] Eclipse, ”Graphical Modeling Framework”, see also http://www.eclipse.org/gmf/ Last accessed 2008-04-29

[35] Eclipse, “ Model Development Tools (MDT)”, see also http://www.eclipse.org/modeling/mdt/ Last accessed 2008-04-29

[36] Eclipse, “MDT Update Manager site” http://www.eclipse.org/modeling/mdt/updates/ Last accessed 2008-04-29

[37] Eclipse, “OCL developer guide”, see also http://help.eclipse.org/help33/nav/35 Last accessed 2008-04-29

[38] Eclipse, “Eclipse Modeling Project”, see also http://www.eclipse.org/modeling/ Last accessed 2008-04-29

[39] Omondo, “EclipseUML Features”, http://www.eclipsedownload.com/eclipseUML_features.html Last accessed 2008-04-29

[40] Markus Scheidgen , “TEF”, see also http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tool.html Last accessed 2008-04-29

[41] Markus Scheidgen , “Tutorial”, see also http://www2.informatik.hu-berlin.de/sam/meta-tools/tef/tutorial.html Last accessed 2008-04-29

[42] ikv++ technologies ag http://www.ikv.de/index.php

[43] ikv++ technologies ag, medini QVT website, see alsp http://www.ikv.de/index.php?option=com_content&task=view&id=75&Itemid=77

[44] ikv++ technologies ag, medini QVT demo video, see also http://www.ikv.de/ikv_movies/mediniQVT Last accessed 2008-04-29

[45] Mernik, M., Heering, J., Sloane, Anthony M., "When and how to develop domain-specific languages", ACM Computing Surveys, Vol. 37, No. 4, December 2005, pp. 316–344

[46] Microsoft, “Visual Studio SDK”, see also http://msdn2.microsoft.com/en-us/library/bb166441(VS.80).aspx, Last accessed 2008-04-02

[47] Microsoft, “Getting Started with Domain-Specific Languages”, see also http://msdn2.microsoft.com/en-us/library/bb126278.aspx Last accessed 2008-04-02

[48] Microsoft, “Introducing Visual Studio”, see also http://msdn2.microsoft.com/en-us/library/fx6bk1f4(VS.80).aspx Last accessed 2008-04-29

[49] Microsoft, “Domain-Specific Language Tools”, see also http://msdn2.microsoft.com/en-us/library/bb126235(VS.80).aspx Last accessed 2008-04-01

[50] MSDN Forum ” Problem with List type value property”, see also http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=348389&SiteID=1 Last accessed 2008-04-29

[51] MSDN Forum, “How to wrap a DSL diagram inside a Windows Form” , see also http://forums.microsoft.com/MSDN/ShowPost.aspx?PostID=815803&SiteID=1&mode=1 Last accessed 2008-04-29

[52] Micrisift, “Generating Artifacts Using Text Templates”, see also http://msdn2.microsoft.com/en-us/library/bb126445(VS.80).aspx Last accessed 2008-04-29

[53] openArchitectureWare, see also http://www.openarchitectureware.org/ Last accessed 2008-04-02

[54] oAW forum, “Problem with dependencies to editor”, see also http://www.openarchitectureware.org/forum/viewtopic.php?showtopic=5697 Last accessed 2008-04-29

[55] oAW update site http://www.openarchitectureware.org/updatesite/milestone/site.xml Last accessed 2008-04-29

[56] Kermeta, see also http://www.kermeta.org/