Modeling Constraint Programs with Software Technology Standards*

Student name: Matthias Hoche
Supervisor name: Prof. Dr. Ing. Stefan Jähnichen

Fraunhofer FIRST, Kekuléstr.7, 12489 Berlin, Germany mathoc@first.fhg.de

Abstract. There is no unified modeling standard available to the CP community, so constraint programs can not be developed independently from the used CP library. Without such standard, CP applications are difficult to develop and to maintain as also no substantial tool support can be enabled to help here. This hinders a wider use in business.
This paper targets platform independent modeling of constraint programs in an object-oriented way. It will be shown how models of constraint programs can be given using software technology standards and further how this standards will enable automated transformations of models into executable source code. Using existing well supported modeling languages can help to increase the acceptance of CP in business applications.

1 Introduction

In [6] Puget argued for a "model-and-run" paradigm for constraint programming. He proposed to develop a standard file format to express CP models. Such a standard will bring abstraction and independence from the used CP library. This increases maintainability and transparency of constraint programs and can support the integration of CP in business applications. Currently no such standard exists to the CP community.

This work proposes some initial ideas how a modeling standard for CP can be achieved.

- Chapter 2 gives an introduction to Model Driven Architecture (MDA) as a software technology standard for the definition of platform independent models and also their transformation to a specific platform. It will be shown how the Unified Modeling Language (UML) [5] can be extended with profiles to define domain specific semantics.
- UML Profiles can be used to create "CP modeling profiles" for platform independent modeling of constraint programs (chapter 3). The use of CP modeling profiles enables the application of MDA transformation techniques.

This paper refines ideas presented in [9] already. There, Wolf et al. have formally shown how standards like UML and OCL [8] can be used to create well formed models of a constraint problem called Constraint Network Schemata (CNS). This paper considers CNS modeling techniques in the context of MDA.

* This work is funded by the EU (EFRE) and the state Berlin, grant no. 10023515.

2 Model Driven Architecture - MDA

After the introduction of compilers making high-level programming languages available, software technology targets modeling in a general approach. This leads to a process of compiling models to programs, the same way programs can be compiled to machine code.

The OMG had a great influence on this development with the proposal of the Model Driven Architecture in 2001. This proposal addresses a full life-cycle application development as an industry architecture standard, see [3]. The young MDA process accelerated with the support of leading industry partners.

This chapter gives a short introduction to the MDA standard. The first part summarizes what makes out a model in this context. The second part will focus on the transformation of these models.

Meta-Models and UML Profiles In MDA, every model is defined by a meta-model. All meta-models are specified conform to the Meta Object Facility standard [2]. Models are further separated into Platform Independent Models (PIM) and Platform Specific Models (PSM). The preferred modeling standard for a PIM is the Unified Modeling Language (a meta-model for UML models).

UML is an approach for object oriented modeling. Its generality is a problem, because it is hard to give precise models for software in all its facets. One way to solve this is to add domain specific semantics. UML provides three extensibility mechanisms to do this: *stereotypes*, *tagged values* and *constraints*.

Stereotypes and *tagged values* can be defined as new modeling elements. They do not extend the structure but add semantics to existing model elements. While stereotypes can be used as a meta-class to mark elements, a tagged value gives an explicit definition of a name-value pair. A set of stereotypes and tagged values will produce a UML Profile. A profile defines the semantics and usage of these elements. Model *constraints* in form of a textual language enrich model elements with semantic conditions or restrictions. The used language is arbitrary. At the moment OCL is often used as a model constraint language.

These extensions help to create models with rich semantics for a considered domain. Their usage in the CP context is explained in chapter 3.

Transformations and Code Generation Besides defining models MDA is also a standard for model to model transformation. Several transformations can be combined to a transformation-chain, see figure 1. After all, the process can end up in the most platform specific model - the source code.

There is a growing number of tools providing an MDA framework (overview in [4]). Example 1 summarizes how these tools handle code generation to illustrate MDA transformations.

Example 1. First the meta-model elements are covered by patterns, see figure 2. These patterns define what activity has to be invoked when a model element of

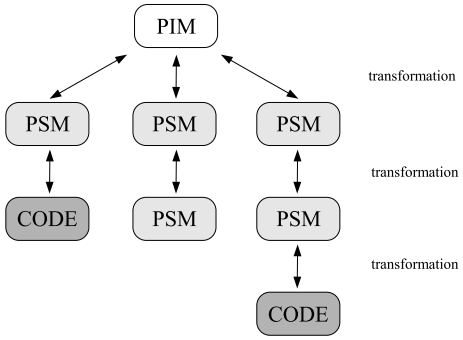

Fig. 1. MDA transformation

the meta-model's type is encountered. Dependent from the used tool, this can be the application of a script, a template or an abstract template.

These procedures differ in behavior and generation power. Where a template language will produce textual output directly, scripting languages (like Basic) and abstract template languages can produce an in-memory-representation as a model object at first. The in-memory-representation is more flexible because all properties can be edited before textual output is produced.

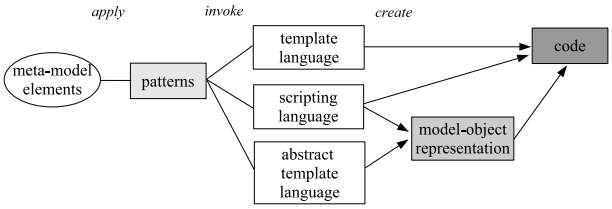

Fig. 2. code generation procedures

3 CP Modeling Profiles

The last chapter gave a sketch of the MDA standard. Its application to the domain of constraint programming is discussed now. Therefore the next example shows, how constraint models can be given with UML and its extensibility mechanisms.

Example 2. Figure 3 shows simple models of a factory (left) and a CP-library (right). The factory model contains the structure elements of the problem representing real-world objects. Here we have available machine times (`MachineTime`)

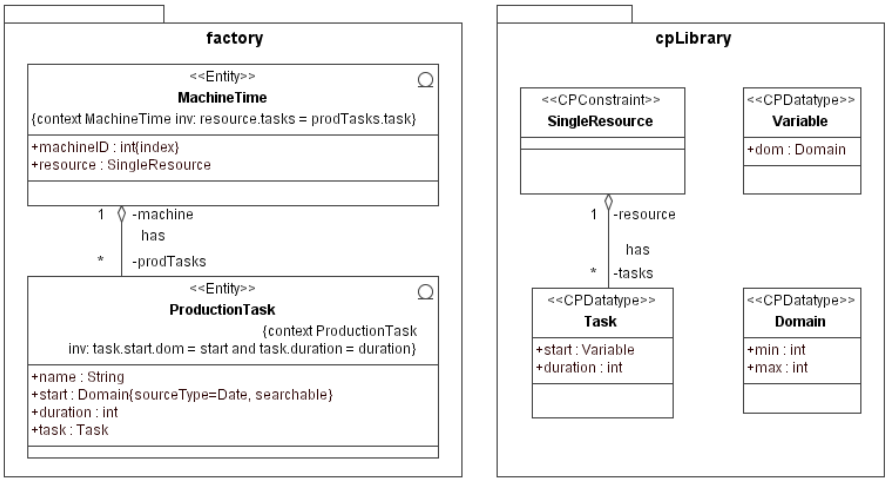

Fig. 3. Models of a Factory using a CP - library

and production tasks (`ProductionTask`) to be scheduled in them. The CP-library model is a template for available CP-elements. These CP-elements are added as attributes to structure elements giving the specification of the constraint problem. In this case `SingleResource`[1], `Task` and `Domain` are taken from the CP-library template and are added to `MachineTime` and `ProductionTask`.

Now, the UML extensions are used as follows: Stereotypes separate the CP-elements from the structure elements, expressed with `<<CPConstraint>>` and `<<Entity>>` for example. The stereotype `<<Entity>>` is used to handle persistent data storage[2] for this classes what specifies the input and output of the constraint program. The relations between CP-elements and such input is established by model constraints with OCL, used here to define invariants for equalities between objects. The invariant: `resource.tasks=prodTasks.task` in `MachineTime` expresses which tasks have to be constrained by `resource`. Further the invariant: `task.start.dom=start and task.duration=duration` in `ProductionTask` defines which elements of this class become the task property and create the CP element `task`.

Tagged Values are added for further specifications. For example, the tag `index` for `machineID` specifies that this attribute is an index for a database table storing `MachineTime`-objects. Also the tag `searchable` for the attribute `start` could mark this element for inclusion in an underlying search strategy. In addition the tag `sourceType` can lead to automated transformation of input values, here from type `Date` to a CP data type `Domain`.

[1] *SingleResource* is a global constraint, which aligns tasks on a single resource without overlapping

[2] for example in databases or XML-files

A constraint model as defined in example 2 is very similar to a Constraint Network Schema introduced in [9]. As an extension to CNS the explicitly given CP-library, stereotypes and tagged values were added.

Obviously, this simple example provided some ideas only. Neither does it cover all aspects of modeling constraint programs nor are transformations for all of the given semantics available. The main problem here is the use of an existing code base - the CP library. This requires the dynamic creation of library elements during runtime. Modeling of such behavior is not fully supported in the current UML standard and available transformation techniques. Anyhow, the explicit knowledge of an underlying CP-library and the restriction to the domain of CP can help to create expressive models of constraint programs. Furthermore, MDA automation processes for parts of the model in example 2 are present already, i.e. persistent data storage.

However, before transformation rules can be developed to generate executable constraint programs a concise standardized model specification has to be defined for the domain of CP. The author proposes to adopt UML profiling techniques to create such model specification, see figure 4. Therefore a set of stereotypes, tagged values and model constraints must be defined (as seen in figure 3). Together with the model of a CP-library this can be used as a model template - a **CP modeling profile**.

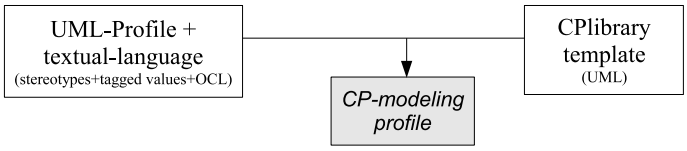

Fig. 4. modeling profile for CP

But having one such template to model all aspects of CP will complicate the development process also. To simplify standardization CP modeling profiles can be developed successively for specialized purposes. Take scheduling problems for example. A CP modeling profile for scheduling will contain special global constraints and search-strategies well suited to this application field, like the `SingleResource`-constraint in figure 3. This does not restrict the model because CP modeling profiles could be combined the same way UML profiles can be combined already. Available MDA tools can then be used for the generation of executable constraint programs out of models using these profiles.

It depends on the expressiveness of the model as also on the quality of the used transformation rules, whether such improvements bring a "model-and-run" functionality. If this can not be achieved in a first attempt, code generation will bring at least a "model-generate-code-and-run" approach, what is a flexible solution for the beginning. As a great benefit of using available software technology standards the process can participate from their advances in future.

4 Related Work and Conclusion

In [7] Schlenker and Ringwelski presented POOC as an object oriented programming syntax for finite domain constraint solver. POOC can be used as a Java package to develop constraint programs independently from a CP-library. Libraries can be easily integrated by implementing POOC's interface and will then be bound during runtime.

Frank et al presented META-S in [1] as a unifying framework to combine different solvers into one global solution strategy. This is very useful when problems require support from solvers of different CP-domains like linear equation solvers or finite domain solvers.

Both META-S and POOC achieve abstraction with meta-programming where META-S focuses more on strategy than platform independence. Although these approaches differ from the ideas discussed here they demonstrate that abstraction from the used CP-library is possible what encourages the author's hope for a successful application of MDA techniques to CP.

As a summary, Model Driven Architecture can be seen as a chance to develop a general modeling approach for Constraint Programming - making a "model-and-run" paradigm possible. If this will be achieved, CP applications can be developed with well supported and widely used modeling standards. Using such standards can increase acceptance and usability of constraint programs in business applications.

References

1. Stephan Frank, Petra Hofstedt, and Pierre R. Mai. Meta-S: A Strategy-oriented Meta-Solver Framework. In Ingrid Russell and Susan Haller, editors, *Proceedings of the 16th International Florida Artificial Intelligence Research Symposium Conference (FLAIRS)*. The AAAI Press, May 2003.
2. OMG Object Management Group. Meta Object Facility. http://www.omg.org/technology/documents/formal/mof.htm.
3. OMG Object Management Group. Model Driven Architecture. http://www.omg.org/mda/.
4. Bernhard Merkle. Designermodelle. *iX - Magazin für professionelle Informationstechnik*, page 102, May 2005.
5. Object Management Group, Inc. *OMG Unified Modelling Language Specification*, March 2003.
6. Jean-Francois Puget. Constraint Programming Next Challenge: Simplicity of Use. In Marc Wallace, editor, *Principles and Practice of Constraint Programming – CP 2004, 10th International Conference, Proceedings*, number 3258, pages 5–8, Toronto, Canada, September/October 2004. Springer-Verlag.
7. Hans Schlenker and Georg Ringwelski. POOC - A Platform for Object-Oriented Constraint Programming. In *ERCIM/CologNet Workshop on Constraint Solving and Constraint Logic Programming*. Springer LNCS 2627, 2002.
8. Jos B. Warmer and Anneke Kleppe. *The Object Constraint Language, Second Edition*. Object Technology Series. Addision-Wesley, 2003.
9. Armin Wolf, Henry Müller, and Matthias Hoche. Towards an Object-Oriented Modeling of Constraint Problems. In *W(C)LP*, pages 41–52, 2005.