Constraint Specification by Example in a Meta-CASE Tool

Hazem Qattous
Department of Computing Science
Sir Alwyn Williams Building
University of Glasgow, G12 8QQ, UK
+44 141 330 0916
hqattous@dcs.gla.ac.uk

ABSTRACT

CASE tools are very helpful to software engineers in different ways and in different phases of software development. However, they are not easy to specialise to meet the needs of particular application domains or particular software modelling requirements. Meta-CASE tools offer a way of providing such specialisation by enabling a designer to specify a tool which is then generated automatically. Constraints are often used in such meta-CASE tools as a technique for governing the syntax and semantics of model elements and the values of their attributes. However, although constraint definition is a difficult process it has attracted relatively little research attention. The PhD research described here presents an approach for improving the process of CASE tool constraint specification based on the notion of programming by example (or demonstration). The feasibility of the approach will be demonstrated via experiments with a prototype using the meta-CASE tool Diagram Editor Constraints System (DECS) as context.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques – *Computer Aided Software Engineering (CASE).*

General Terms

Design, Human Factors, Languages.

Keywords

Meta-CASE tools, Domain Specific Language, Programming By Example.

1. INTRODUCTION

CASE tools are helpful to software engineers in increasing the productivity, shortening development time, and improving software quality. One of the main advantages of CASE tools is their potential domain specificity. Meta-CASE tools can specify and generate domain specific CASE tools including design diagram editors, the experimental domain of this research. A meta-CASE tool defines a diagram editor by specifying the modelling language itself through a meta-modelling process. Meta-modelling techniques depend on two components to define the domain specific language syntax and semantics, a meta-model and constraints.

The meta-modelling process is complicated, needs time, and in the case of several tools, needs experts [4]. This is because the definition of syntax and semantics needs experience and knowledge of the meta-model, constraint definition, and the domain specific language to be modelled. In the field of meta-CASE tools, most research is directed towards enhancing the meta-models to improve the meta-modelling process [7]. However, constraints which play an important role in CASE tool configuration and the meta-modelling process attract very little research attention. It is believed that improving the meta-modelling process can be achieved through enhancing and simplifying the constraint definition process. However, constraint definition is not an easy task. Constraint definition, in the context of meta-CASE tools, is performed using a constraint language. “MetaBuilder” [4] is a meta-CASE tool that requires the user (CASE tool designer) to enter constraints via a constraint language. Consequently, the user must learn a constraint programming language to accomplish the job. Several studies have attempted to reduce the difficulty of constraint definition within meta-CASE tools. In place of direct entry, “MetaEdit+” [8] uses form-filling, with one form for each constraint type.

One other technique used to reduce the difficulty of constraint definition is visual programming [10]. The Pounamu meta-CASE tool includes a visual language to represent events and associated actions. They also have proposed using programming by example in the event handling as another technique to reduce the complexity of the process including constraint definition. These examples indicate that constraint definition is not easy and several approaches have been offered to solve its complexity. This also indicates the importance of constraints as a part of the meta-modelling process.

The next section of this abstract presents the aim of the research including a thesis statement. This is followed by an outline of the approach being taken. Constraints in meta-CASE tools and their importance are then introduced in section 4. DECS, the meta-CASE tool used for this work, is presented in section 5. Section 6 introduces programming by example and its use for specifying constraints. Section 7 outlines progress made so far, followed in Section 8 by plans for future work and a set of open questions.

2. Thesis Statement

The aim of this research is to improve the process of constraint definition as part of domain specific CASE tool specification in a meta-CASE tool. In particular, the work will focus on facilitating and simplifying constraint definition process using programming by example technique. Programming by example technique has not been used before in the context of meta-CASE tools which is considered as the main contribution of this research.

In summary, *this research sets out to establish that it is possible to improve the process of CASE tool specification by facilitating and simplifying the constraint definition process using a programming by example technique*.

3. Approach

To be able to accept or reject the hypothesis of the thesis, studies should be conducted to explore the feasibility, advantages, and disadvantages of the new technique, programming by example, in constraint definition process in a meta-CASE tool.

As a first step in the research, an experimental meta-CASE tool, Diagram Editor Constraints System (DECS), has been developed, improved and used as a platform and context for the experiments.

The next step involved creating an XML-based constraint description and expression language. Concurrently, a constraint manager that holds and handles constraints at runtime was developed as a separate component and attached to DECS.

The third step introduced a way to define constraints other than direct editing of the XML constraint descriptions. The first part of this step required developing a constraint definition wizard.

Recently, a constraint definition by example technique has been partially developed. An inference engine has been adopted and embedded within a newly developed component, the inference manager. Details of these steps are given in the following sections.

4. Constraints in Meta-CASE Tools

In the context of design diagram editors, constraints can be considered as signs to guide architects to a good design solution. They are rules that limit the available alternatives to achieve a task, and provide valuable support to designers by enforcing compliance with a specific development methodology [14].

In most reviewed meta-CASE tools, constraints play an important role in specifying diagram editors and composing an important part of the meta-modelling process. In meta-CASE tools, both, meta-model and constraints are used to define syntax and semantics of the domains specific language. Despite this importance of constraints in meta-modelling process, very little research conducted to enhance constrain definition process such as [10] and [5] compared to research conducted in the field of meta-models [7].

5. DECS

DECS (Diagram Editor Constraints System) is an Eclipse-based meta-CASE experimental prototype initially developed at the University of Glasgow prior to the start of this research. It generates constraint-based domain specific diagram editors extended from Graphical Editing Framework (GEF) Eclipse plug-in. The DECS meta-modelling process is composed of defining vertices, edges, and constraints. As shown in DECS architecture (figure 1), this process is done by the CASE tool designer (editor designer) using wizards. The definition is stored as XML files.

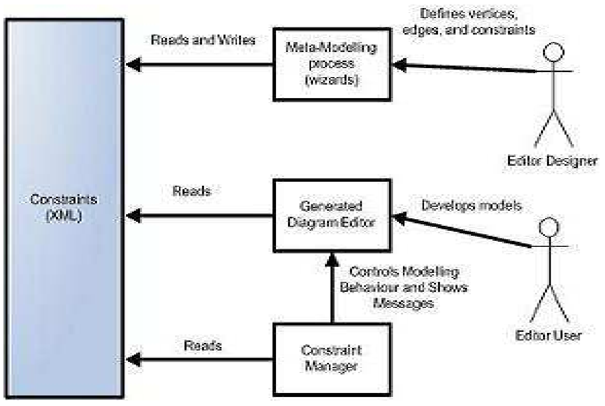

Figure 1. DECS architecture.

The CASE tool user (editor user) uses the generated environment to develop system diagrams and models. DECS depends mainly on XML files as a repository to allow communication between different levels (meta-level and modelling level).

5.1 The Constraint Manager

The constraint manager is implemented as a separate DECS component. It reads the constraint XML files and converts them into runtime objects using the wrapper designer pattern. The design depends on considering every property of the constraint as a separate wrapper layer which gives flexibility for any future extension of the properties. The constraint manager maintains the constraints in a list and uses them as assertions for the user's actions in the modelling environment. Every time the user updates (modifies) the diagram model, the constraint manager is consulted. If a violation is detected, the constraint manager either warns or prevents the user from violating the constraint, depending on the violated constraint type.

6. Constraint Definition by Example

According to [11], programming by example is the technique of presenting examples of data and values to the system, from which it can generalise these values to generate a program. The objective of programming by example is to make programming an easier task and available for non-programmers. The technique depends on providing examples of the required program to the system. The system then infers the program by generalising the examples. Cypher [3] introduces inferring the user intent from an example as the main challenge in applying such technique.

This technique has been applied in a number of different contexts such as DocWizards" [12] for generating documentation, and in mapping between a state and its associated actions in robotic applications [1]. The programming by example technique has been applied to constraint specification by several researchers. Myers [11] used the technique to infer constraints between different graphical user interface components. Using "Peridot", it is possible to build a GUI without programming. Kurlander & Feiner [9] used the same technique to infer geometric constraints between vertices in a system called "Chimera". Borning [2] defines graphical constraints by example that must be true all the time. The system always tries to keep the constraints hold while the user manipulates the graph.

However, from the reviewed literature, it has been noticed that the constraint by example technique has not been used yet in the context of meta-CASE tools. In this research, it was decided to study the feasibility of applying this technique for the purpose of making constraint definition in meta-CASE tools easier.

7. Progress Report

7.1 Constraint Description

To be able to express constraints, DECS adopted a property-value XML description as each constraint depends on a set of properties. Similar technique has been introduced by [14]. Some values in a DECS constraint description can be the URIs of other constraints. This ability to encapsulate constraints within each other makes it possible to construct complex constraints. The same feature has been introduced by [10] to express complex behaviour and they called that “packing”.

Within DECS, the user can define a constraint either using a wizard or by example. In the first case, the user goes through several forms to assign values to different constraint properties as required. The proposed technique, constraint definition by example, depends on allowing the user to introduce one or more examples of the required constraint. From the example(s), the system should be able to infer the intended constraint. The inference process in this context is fairly complicated because of constraints’ complicated nature and the wide range of constraint alternatives that an example could mean. A similar problem has been introduced by [9]. Their solution depended on limiting the number of constraints that can be inferred. Since DECS generates constraint-based CASE tools, it is not possible to adopt a similar solution as that may limit the number of constraints or their expressiveness.

7.2 Constraint Inference

The key challenge in this research is the inference engine and inference technique to be used. A general review of the available inference engines and inference techniques has been conducted. It has been decided to use forward chaining instead of backward since the user introduce example to the system and it should infer the target constraint not vice versa. An open source inference engine [13] has been adopted and modified for use in DECS. One key advantage of this system is the simplicity of its rule language, simply written as if-then text statements in natural language, lowering the overhead of writing rule sets.

A new component, the inference manager, has been developed and the inference engine has been adapted within it. The inference manager is only effective when the user (CASE tool designer) is in the constraint definition by example environment. Whenever the user performs an action (add, delete, connect, or modifies a property) on any element (vertex or edge), the manager reviews its knowledge base and performs an inference.

Another challenge in this research is the generalisation of objects (elements, including vertices and edges). This problem has been introduced by [15] in Kidsim; they overcame the problem by allowing the user to specify his/her intention. In DECS, this problem is handled using the rules in the inference engine. Inference rules have been classified into two types, “choice rules” and “action rules”, with different behaviour. Choice rule, if triggered, returns a choice to the user which represents an inference from the current example (graph). Action rule affects the example itself by generalising some elements of it.

The system takes the example(s) introduced by the user and collects all the inferences generated by choice rules. These inferences are introduced to the user to choose from. If the user cannot find the intended constraint, the inference manager looks for an action rule to apply. Applying an action rule leads to a modification of the example itself. This can be considered as rewriting the example as shown in (figure 2). Although this is different from graphical rule rewriting introduced in KidSim by [15], the graph rewriting or re-drawing process in DECS is important to achieve the required generalisation and to reduce the number of choices presented to the user.

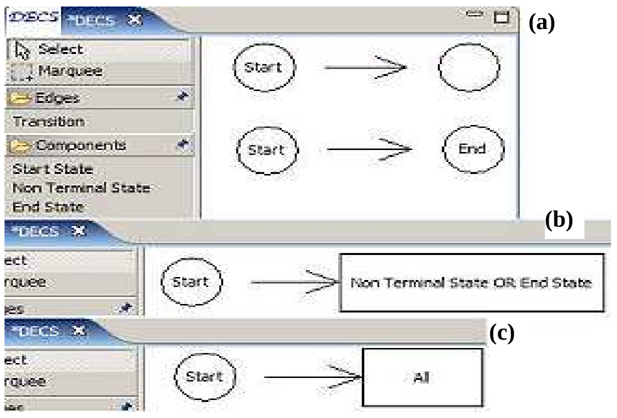

Figure 2: Rewriting graph in DECS. (a) example by the user, (b) first rewriting, (c) second rewriting.

Rewriting the example in DECS can be considered as a form of inference which leads to triggering new rules. As the example has been modified, new choice rules can be triggered. These rules return new choices to the user to choose from. This cycle is repeated until no more action or choice rules are triggered. In this manner, the generalisation task has been assigned to the rules at the knowledge base not to the user. In each cycle, the user is asked to choose from a small inference list instead of presenting all the inferences at once. This solves the problem of the large number of possible meanings of an example by showing them all. The process also involves the user in helping to determine the required constraint, which can be considered as synergistic or cooperative interaction between the user and the system. Such synergistic interaction has previously been used in a programming by example system to define the visual layout of a graph [6].

In contrast to most reviewed constraint programming by example tools, DECS depends on negative constraints which represent what must not be allowed. DECS negative constraints can be expressed using negative examples. “Peridot” depends on positive constraints and uses positive examples to express them. However, sometimes, it was important to introduce “negative examples” to express “what not to do” [11]. Similarly, positive examples are sometimes useful in DECS for expressing negative constraints. Although users may not be aware of the difference, they should always keep in mind that they are working in a negative example environment.

The knowledge base has been designed with high flexibility for extension. Each rule is composed of two parts, the IF and the THEN part. Each part is represented by a singleton object. The

IF object evaluates if the rule is triggered or not. The THEN object returns either a choice or a modified example (graph) depending on its type. Whenever there is a need to add a rule, then it is necessary to write the rule in the inference engine rule language, create an object for the IF part to evaluate the rule, and create an object to return either a choice or a modified graph.

7.3 Initial Experiment and Evaluation

As a first investigation of the thesis hypothesis, an experiment will be conducted to evaluate constraint definition by example compared to constraint definition by wizard. For this purpose, a set of constraints that specify state transition diagram syntax and semantics have been identified. A state transition diagram has been chosen because it is well known and relatively easy to understand. The diagram also contains all the general constraints (syntax and semantics) that may appear in most other diagrams. A number of subjects will be trained to define constraints using the two techniques. The determined constraints will be divided into two sets. The two sets will interchangeably defined using one of the techniques. Task performance will be measured in terms of accuracy and time to complete. Participants will also be interviewed to find out about perceived effort and performance as well as their subjective assessment of the techniques.

8. Future Work

The next step of the research will be trying to apply constraint definition by example to specify another diagram type. Some questions that can be asked here include: is there any effect of the modelling diagram itself? Is there any specific constraint type that is easier to define using a wizard than by the example-based technique?

This thesis is an initial investigation into the viability of the programming by example technique in the meta-CASE domain. It may be necessary to explore different inference techniques if our relatively simple rule-based method does not prove to be satisfactory in our first experiments. As a part of this, rule sequencing and prioritising depending on user interaction and preferences could be applied to study their effect on the constraint definition process. The current inference engine uses rules for inference. Instead, other artificial intelligence technique could be used, such as neural networks or genetic algorithms, to explore the effect of enabling the system to learn its rule set. These experiments may help to answer questions like: is it possible to apply other artificial intelligence techniques within the programming by example technique? Will that make the constraint definition process easier?

9. REFERENCES

[1] Argall, B., Chernova, S., Veloso, M., & Browning, B. (2009). A survey of robot learning from demonstration. *Robotics and Autonomous Systems , 57* (5), 469-483. DOI= 10.1016/j.robot.2008.10.024.

[2] Borning, A. (1986). Defining Constraints Graphically. *Conference on Human Factors in Computing Systems* (pp. 137-143). Boston, USA: ACM, USA. DOI= http://doi.acm.org/10.1145/22627.22362

[3] Cypher, A. (1993). *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

[4] Ferguson, R., & Hunter, A. (2000). MetaBuilder: The Diagrammer's Diagrammer. *Lecture Notes In Computer Science; Vol. 1889* (pp. 407 - 421). London, UK: Springer-Verlag.

[5] Gray, P., & Welland, R. (1999). Increasing the flexibility of modelling tools via constraint-based specification. *Proceedings of the 1999 conference of the Centre for Advanced Studies on Collaborative research.* Mississauga, Canada: IBM Press.

[6] Hudson, S., & Hsi, C.-N. (1993). A Synergistic Approach to Specifying Simple Number Independent Layouts by Example. *Conference on Human Factors in Computing Systems* (pp. 285-292). New York, USA: ACM. DOI= http://doi.acm.org/10.1145/169059.169221.

[7] Isazadeh, H., & Lamb, D. A. (1997). *CASE Environments and MetaCASE Tools.* Technical Report 1997-403, Queen's University.

[8] Kelly, S. (2009). *Domain-Specific Modeling: A Toolmaker Perspective*. MetaEdit+ Workbench questions. [online]. Available at: http://www.metacase.com/faq/showfaq.asp?cate=MWB [Accessed 1 March 2009].

[9] Kurlander, D., & Feiner, S. (1993). Inferring Constraints From Multiple Snapshots. *ACM Transactions on Graphics , 12* (4), 277-304. New York, USA: ACM. DOI= http://doi.acm.org/10.1145/159730.159731

[10] Liu, N., Hosking, J., & Grundy, J. (2007). A Visual Language and Environment for Specifying User Interface Event Handling in Design Tools. *Conferences in Research and Practice in Information Technology Series; Vol. 241* (pp. 87 - 94). Ballarat, Victoria, Australia : Australian Computer Society, Inc., Australia.

[11] Myers, B. (1993). Peridot: Creating User Interfaces by Demonstration. In A. Cypher, *Watch What I Do: Programming by Demonstration.* Cambridge, Mass.: MIT Press.

[12] Prabaker, M., Bergman, L., & Castelli, V. (2006). An Evaluation of Using Programming by Demonstration and Guided Walkthrough Techniques for Authoring and Utilizing Documentation. *Conference on Human Factors in Computing Systems, Proceedings of the SIGCHI conference on Human Factors in computing systems* (pp. 241 - 250). Montréal, Québec, Canada: ACM, New York, USA. DOI= http://doi.acm.org/10.1145/1124772.1124809

[13] Sazonov, E. (2004). *Open source fuzzy inference engine for Java*. [online]. Available at: http://people.clarkson.edu/~esazonov/FuzzyEngine.htm [Accessed 27 March 2009].

[14] Scott, L., Horvath, L., & Day, D. (2000). Characterising CASE Constraints. *Communications of the ACM, 43 (11)*, 232-238. DOI= http://doi.acm.org/10.1145/352515.352533.

[15] Smith, D., Cypher, A., & Spohrer, J. (1994). KidSim: programming agents without a programming language. *Communications of the ACM , 37 (7)*, 54-67. DOI= http://doi.acm.org/10.1145/176789.176795.