SiMoL– A Modeling Language for Simulation and (Re-)Configuration*

Iulia Nica and **Franz Wotawa**†
Technische Universität Graz, Institute for Software Technology
Inffeldgasse 16b/2, Graz, Austria
{inica,wotawa}@ist.tugraz.at

Abstract

Simulation and configuration play an important role in industry. Modeling languages like Matlab/Simulink or Modelica, which are often used to model the dependencies between the components of physical systems, are less suitable for the area of knowledge-based systems. For these languages, the description of knowledge and its connection to a theorem prover for nonmonotonic reasoning (needed for configuration tasks) is, due to technical reasons, almost impossible. In this paper we focus on a language that can be used for both simulation and configuration purposes. SiMoL is an object-oriented language that allows representing systems comprising basic and hierarchical components.

1 Introduction

The adaptation of technical systems after deployment to ensure the desired system's functionality over time is an important task and can never be avoided. Reasons for adaptation are necessary corrections due to faults in system parts, changes in user requirements, or changes of technology among others. All activities necessary for increasing the lifetime of a system and retaining its usefulness are summarized under the general term maintenance.

In our research we focus on system changes due to changes in requirements. For example, consider a cellular network where the base stations are initially configured to ensure current and future needs to some extent. Due to changes in the environment, i.e., new apartment buildings constructed in reach of the base station or an increased use of cellular networks for data communication, the base station or even the local topology of the network has to be adapted. This adaption can more or less be classified as a re-configuration problem where the current system's structure, behavior, and the new requirements are given as input. Changes in the structure and behavior of the system in order to cope with the changes in the requirements are a solution of the re-configuration problem.

In order to provide a method for computing solutions for a given re-configuration problem we need to state the problem in a formal way. Therefore, we require a modeling language for stating systems comprising components and their relationships. In principle, formal languages like first order logic or constraint languages would be sufficient for this purpose. But using such languages usually is not easy and prevents systems based on such languages to be used in practice. Hence, there is a strong need for easy to learn and use modeling languages that are expressive enough to state configuration problems. The SiMoL language we introduce in this paper serves this purpose. The language itself is from a syntactical point of view close to Java. The idea behind SiMoL is to provide a language that can be used for (restricted) simulation and configuration at the same time.

SiMoL is an object-oriented language with multiple inheritance and allows for stating constraints between variables. Beside the basic data types like integer and boolean, SiMoL makes use of component instances. All component instances are statically declared. In this paper we focus on describing the syntax and the semantics of SiMoL.

2 Related research

Over time, the AI community has developed a large variety of configuration tools that fitted the different necessities and goals in each practical area, thus creating a strong foundation for newcomers. As preamble to our approach, we shortly recall three configuration systems, that make use of constraint programming.

ConBaCon [John and Geske, 1999] treats the special case of re-configuration, using the conditional propagation of constraint networks and has its own input language - ConBaConL. In [John and Geske, 1999], the authors present ConBaConL, a "largely declarative specification language", by means of which one can specify the object hierarchy, the context-independent constraints and the context constraints. Furthermore, the constraints are divided into Simple Constraints, Compositional Constraints and Conditional Constraints.

LAVA is another successful automated configurator [Fleischanderl *et al.*, 1998], used in the complex domain of telephone switching systems. It makes use of generative constraints and is the successor of COCOS [Stumptner *et al.*,

*Authors are listed in alphabetical orde. The work presented in this paper has been supported by the BRIDGE research project Simulation and configuration of Mobile networks with M2M Applications (SIMOA), which is funded by the FFG.

†Corresponding author.

1994], a knowledge-based, domain independent configuration tool. The modeling language is ConTalk, an enhanced version of LCON that follows the Smalltalk notation. A ConTalk constraint is a statement which describes a relationship between components ports or between the attributes values.

A powerful configuration system that combines constraint programming(CP) with a description logic(DL) is the ILOG (J)Configurator [Junker and Mailharro, 2003]. The combined CP-DL language, in which the configuration problem is formulated provides, on the one hand, the constraints, needed in the decision process, and on the other hand, the constructs of the description logic, able to deal with unknown universes. When solving the problem, the constructs of description logic, which are well-suited to model the configuration specific taxonomic and partonomic relations, are mapped on constraints and thus the wide range of constraint solving algorithms may be used.

The other field of interest for our research has been the modeling languages currently used for simulation of technical systems. Matlab/Simulink [4] and Modelica [5] are the most famous ones in the area of dynamic systems modeling and simulation. When working with Simulink, the user is capable of modeling the desired system in the graphical interface, based on the large library of standard components (called blocks). Also making use of predefined building blocks, Modelica, on the other side, is an equation-based object-oriented language with multi-domain modeling capability. Although both of them are complex languages, capable of modeling a great variety of components, neither Simulink or Modelica can be used for re-configuration purposes, as the description of knowledge and its connection to a theorem prover for nonmonotonic reasoning (needed for configuration tasks) is, due to technical reasons, almost impossible.

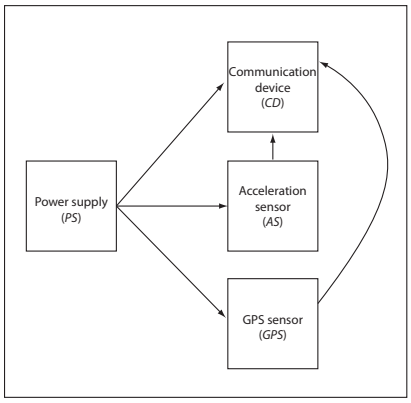

Figure 1: A small sensor systems

Throughout the rest of this paper, we present our modeling language - SiMoL. SiMoL can be applied in both simulation and re-configuration domains, using the powerful mechanism of constraint solving and hence being highly scalable for complex simulation and re-configuration tasks.

3 An example

In this paper we make use of the following small example to discuss SiMoL, as well as re-configuration using SiMoL for modeling systems. Figure 1 depicts a small system comprising 4 components, i.e., a power supply (PS), an acceleration sensor (AS), a GPS sensor (GPS), and a communication device (CD). The communication device is used for sending the measured sensor information to a server. The power supply is for providing electricity to the connected components. All these components have a behavior and provide functionality.

For the purpose of specifying functionality we introduce a function fct that maps a component to a set of attributes, which indicate a certain functionality. For our example, we introduce the attributes **ad**, **gps**, **comm** to state the acceleration sensor functionality, the gps functionality, and the ability for communication respectively.

$$fct(AS) = \{\textbf{ad}\} \;\; fct(GPS) = \{\textbf{gps}\} \; fct(CD) = \{\textbf{comm}\}$$

We now specify additional constraints of the system. The following constraint formally represents the requirement that the power provided by PS must be larger or at least equivalent to the sum of the power consumption of the other components:

$$power(PS) \geq power(AS) + power(GPS) + power(CD)$$

Moreover, we state that the device has to provide at least **ad**, **gps**, **comm** functionality.

$$fct(AS) \cup fct(GPS) \cup fct(CD) \supseteq \{\textbf{ad}, \textbf{gps}, \textbf{comm}\}$$

Finally, we have the requirement that the sum of the cost of each part of the device is not allowed to exceed a certain pre-defined maximum cost.

$$cost(PS) + cost(AS) + cost(GPS) + cost(CD) \leq max\_cost$$

In configuration we are interested in providing specific implementations of the components PS, AS, GPS, and CD such that all requirements are fulfilled and no constraint is violated. Hence, what we do now for our running example, is to introduce specific instances of the generic components with different costs and power consumptions. Table 1 summarizes all the used concrete component implementations.

A valid configuration is now a set of components that fulfills all constraints. For example, when assuming maximum cost of 60, the set $\{PS_1, AS_2, GPS_1, CD_2\}$ is a valid configuration but $\{PS_2, AS_2, GPS_1, CD_2\}$ is not because of violation of the cost constraint.

Throughout this paper we make use of this example and show how SiMoL can be used for modeling such systems.

4 SiMoL definition

In order to define SiMoL we discuss its syntax and semantics as well as its capability to be used for re-configuration purposes.

SiMoL syntax: As already mentioned, SiMoL uses a Java-like syntax and the common conventions compass most of the defined tokens: identifiers for any type of component and attribute, integer and boolean literals, separators, arithmetic and relational operators ($+, -, *, /, =, <, >, <=, >=, !=$), special tokens - comments, reserved words and literals.

Additionally, SiMoL offers support for using *units of measurement*, thus creating a more realistic model.

Another feature of the language, that provides direct control over the possible values of a component attribute, is the

[4] www.mathworks.com

[5] www.modelica.com

Generic component	Instance 1	Instance 2
PS	$PS_1 : costs(PS_1) = 10, power(PS_1) = 10$	$PS_2 : costs(PS_2) = 20, power(PS_2) = 15$
AS	$AS_1 : costs(AS_1) = 2, power(AS_1) = 4$	$AS_2 : costs(AS_2) = 20, power(AS_2) = 1$
GPS	$GPS_1 : costs(GPS_1) = 6, power(GPS_1) = 5$	
CD	$CD_1 : costs(CD_1) = 10, power(CD_1) = 10$	$CD_2 : costs(CD_2) = 20, power(CD_2) = 4$

Table 1: The component instances for our small sensor system

```
component CD{
     attribute int power, costs;
     constraints{
       power={4,6,10} W;
       costs={10..30}; } }
```

Figure 2: SiMoL: initialization of attributes with integer valued ranges

initialization of attributes with integer valued ranges, as illustrated in fig. 4.

Basically, a program written in SiMoL comprises 3 sections: *a knowledge base declaration section*, which is optional, *an import declaration section*, which is also optional, and *a component definition section*, that is the main constructing unit of a SiMoL program and it is mandatory. Generally, each component will posses a set of attributes and will introduce constraints in the system. The attributes declaration is marked by the `attribute` keyword, whilst the relations stated between the component attributes and new-component instance-declaration statements appear enclosed in the `constraints{ ... }` block. By convention, an empty component definition section is not allowed, i.e., if the constraints block is missing, we have to declare at least one attribute for the current component. Furthermore, in the case of *derived components*, the opposite holds: even with no attributes declared, we may state constraints over the inherited attributes. For instance:

```
component AS{
     attribute int power, costs;
     constraints{
       power={4,6} W;
       costs={2..30}; }}
component AS1 extends AS{
     constraints{
       power=4;
       costs=2;}}
```

The ability to extend the functionality and behavior of existing components is of great importance for the taxonomic structure of a configuration domain. In any object oriented languages, the taxonomy relations are represented through the inheritance mechanism. We designed SiMoL with multiple inheritance. In order to demonstrate the necessity of this feature, let us consider the following scenario. For our small system described in Section 3, we introduce a new requirement that refers to a specific signal modulation which can be accomplished by a new component - a modem (M). The modem receives the measured sensor information and transmits the modified signal to the communication device. The function fct from Section 3 will similarly depict for M the modulation-demodulation functionality :

$$fct(M) = \{\textbf{mdm}\}$$

Now the additional constraints of the system become:

$$power(PS) \geq power(AS) + power(GPS) + power(CD) + power(M)$$
$$fct(AS) \cup fct(GPS) \cup fct(CD) \cup fct(M) \supseteq \{\textbf{ad}, \textbf{gps}, \textbf{comm}, \textbf{mdm}\}$$
$$cost(PS) + cost(AS) + cost(GPS) + cost(CD) + cost(M) \leq max\_cost$$

The problem appears if the pre-defined maximum cost is always exceeded, because of the new added component. In other words, we can not afford both a modem and a communication device. Therefore, a new component type - a communication device with integrated modem (MDC)- will solve the case (under the assumption that $cost(MDC) \leq cost(CD) + cost(M)$). In SiMoL, the MDC definition has the following syntax:

```
component MDC extends DC,M {
     constraints{
       power={4,6} W;
       costs={2..30};}}
```

In the constraints section, we may have the following types of statements:

- an empty statement: `;`,
- a component instance declaration: `GPS1 gps1;` Optionally, one can also initialize its attributes: `GPS1 gps1{costs=100};`

  Using this kind of statements, we define the subcomponent hierarchy in our model, i.e., the partonomy relations. The cardinality of these relations (i.e., the number of subcomponents which can be connected to a certain component) is always finite - we cannot have an unlimited number of components in our model.
- an arithmetic or/and boolean expression: `ps1.power>=sum([as1,gps2,cd1],power);`
- a conditional block:

```
if(sum([ps1,as1,gps1,cd1],costs)
<= max_cost)
     cost=sum([ps1,as1,gps1,cd1],costs);
else cost=100;
```

- a `foralll` block:

```
forall(AS1){ power=10 W; costs={1..10};}
```

- an `exist` statement, e.g.:
  `exist(at_most(1),GPS1,costs=30);`.

We also mention the built-in functions `min, max, sum, product`, meant to ease the manipulation of large sets of component instances.

Adopting a clear Java-like syntax, SiMoL is a functionality-based, declarative language, creating a

good environment for simulation, and, at the same time, it provides many embedded functionalities specially designed for configuration purposes.

Semantics of SiMoL: Because of space reasons we only briefly define the semantics of the language SiMoL where we rely on mathematical equations. In particular, we map every statement to a mathematical equation, and combine these equations for a component, taking care of component inheritance and component instances.

For each component defined in SiMoL we have a set of equations that is defined within the `constraints { ... }` block. Moreover, a component also receives equations from its super components and the instances used in the component definition. For example, when specifying `GPS1 gps;` in the variable declaration a new instance of `GPS1` is generated. All constraints of `GPS1` are added to the constraints of the component. The semantics of SiMoL is now nothing else than the union of all constraints defined including inherited constraints and constraints coming from component instances.

We discuss the expressiveness of the language by classifying its capabilities with respect to the framework offered in the chapter on configuration from [Rossi *et al.*, 2006]. In the context of the successful integration of constraint programming in solving a large variety of configuration problems, the author defines several distinguishing constraint models, each corresponding to a specific type of configuration problem. To set up the constraint model, the appropriate variables and constraints are deduced from the given configuration knowledge. The author states that this knowledge may have three different forms: the component catalogs, the component structure and the component constraints.

The catalog knowledge, as defined in [Rossi *et al.*, 2006], is modeled in SiMoL by means of the generic components (correspondent to the term of technical types in [Rossi *et al.*, 2006]) and the concrete components(derived (extended) from generic component/s or from other concrete component/s, in our case, and correspondent to the term of concrete or functional types in [Rossi *et al.*, 2006]). Both generic and concrete components have a set of attributes, mapped to variables in the constraint model. Based on this kind of knowledge, we build the catalog constraints ([Rossi *et al.*, 2006]), which are stated over the set of variables and formulated by means of C_{attr_val} and C_{attr_attr} constraints.

The structural knowledge of a SiMoL model is determined by the component instances declared in the current model. In this manner, we generate for our system the set of subcomponents, that are either generic or extended components. The logic behind this mechanism has been previously detailed, when presenting the semantics of the language. We recall that the SiMoL model is in fact a component, which describes the configuration problem. The connection ports defined in [Rossi *et al.*, 2006] have no correspondent term in SiMoL yet, but the connection between component instances is possible through C_{attr_attr} constraints. Also the statement in [Rossi *et al.*, 2006] according to which "the sets of direct subtypes of two types are mutually disjoint" does not hold in our approach, because we accept multiple inheritance.

Finally, the configuration constraints are divided into compatibility constraints, requirement constraints and resource constraint. The first ones specify which value combinations are legal for the attributes given in the model and they are modeled in SiMoL through C_{attr_val} and C_{attr_attr} constraints. The requirement constraints describe a relation between two component attributes ([Rossi *et al.*, 2006]), which is best depicted by combining C_{cond} with C_{attr_val} or C_{attr_attr}. Moreover, the resource constraints on numerical attributes were intensively addressed throughout this paper.

Consequently, we find the expressive power of the language sufficient for modeling the discussed configuration knowledge forms. As also stated in [Rossi *et al.*, 2006], the configuration problem complexity may vary from very simple option selection problems to complex cases, but they all appear as combinations of the specified knowledge forms.

5 Conclusion

In this paper, we have presented SiMoL- a new functional-based, declarative modeling language, that serves simulation and re-configuration purposes. The novelty of our approach is designing a language that is easy to learn and capable of modeling large and complex systems. SiMoL can cope with large models and be also efficient with respect to computation time (simulation). Although re-configuration is not fully implemented for the SiMoL language, several ideas are currently analyzed and implemented, such that in the near future a fully working re-configurator can be used for SiMoL models. In future research we mainly focus on providing a sound and complete configuration algorithm that takes SiMoL models and requirements as input and computes valid configurations as output.

References

[Fleischanderl *et al.*, 1998] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselböck, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. In *IEEE Intelligent Systems & their applications*, pages 59–68, 1998.

[John and Geske, 1999] Ulrich John and Ulrich Geske. Reconfiguration of Technical Products Using ConBaCon. In *Proceedings of WS on Configuration at AAAI99*, Orlando, 1999.

[Junker and Mailharro, 2003] Ulrich Junker and Daniel Mailharro. The logic of ILOG (J)Configurator: Combining Constraint Programming with a Description Logic. In *Proceedings of IJCAI-03 Configuration WS*, pages 13–20, 2003.

[Rossi *et al.*, 2006] Francesca Rossi, Peter van Beek, and Toby Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006.

[Stumptner *et al.*, 1994] Markus Stumptner, Alois Haselböck, and Gerhard Friedrich. COCOS - a tool for constraint-based, dynamic configuration. In *Proceedings of the 10th IEEE Conference on AI Applications (CAIA)*, San Antonio, March 1994.