(Re-)configuration of Communication Networks in the Context of M2M Applications

Iulia Nica and Franz Wotawa
Institute for Software Technology
Graz University of Technology, Graz, Austria
{inica,wotawa}@ist.tugraz.at

Abstract

Machine-to-machine (M2M) communication is of increasing importance in industry due to novel applications, i.e., smart metering or tracking devices in the logistics domain. These applications provoke new requirements for mobile phone networks, which have to be adapted in order to meet these future requirements. Hence, reconfiguration of such networks depending on M2M application scenarios is highly required. In this paper, we discuss modeling for reconfiguration of mobile phone networks in case of M2M applications and present the foundations behind our tool including the used modeling language and the reconfiguration algorithm.

1 Introduction

Because of the availability of communication networks almost everywhere new applications arise. Besides mobile phones for accessing the internet or simple making phone calls, machine-to machine (M2M) communication becomes a still growing application domain for mobile phone networks. Connecting home appliances like alarm systems or heating installations remotely is one application area. Others are tracking of goods in the logistics domain and smart metering. The latter deals with metering of electrical power or water consumption of homes on a fine granular basis of minutes or hours instead of months or even years. In many countries the administration enforces the use of smart metering in order to rise the customer's awareness of their current consumptions in order to reduce the need for electricity, water or other resources.

Besides this educational effect, there are other advantages of smart metering systems. The overall costs for metering might be reduced because the more labor intensive manual metering is not longer necessary. The supplier of resources gains more information regarding current consumptions, which likely improves prediction of consumptions and further allows for improving the stability of the overall supply network. This is especially important for power networks where electricity has to be generated when needed. Unfortunately, power plants cannot be turned on or off without a substantial delay. Another advantage is that the supplier gains direct remote access to the interface between the network and customer. This allows for instance turning off consumer loads whenever needed in order to prevent for example from a blackout.

M2M communication has been a growing market that causes more and more communication over mobile phone networks. Hence, mobile phone companies providing the infrastructure have to adapt their networks due to future needs. Moreover, a M2M application provider has to be ensured that the current mobile phone network is capable of providing enough resource in order to carry out communication requirements. Within the *Simulation and Configuration of Mobile Networks with M2M Applications (SIMOA)* project the objective has been to develop a simulation and reconfiguration environment for smart metering applications in order (1) to ensure that current mobile phone networks are capable of providing enough resources (through simulation), and (2) to give advice for changing either the smart metering solution or the communication network in cases of lack of resources (through reconfiguration). Hence, we first simulate a network configuration, in order to check whether this can support the set of user requirements, and, if the simulation fails (the requirements can not be fulfilled by the mobile network), a working reconfiguration of the given system has to be delivered.

In this paper, we focus on the SIMOA approach to reconfiguration comprising the modeling language SIMOL, which is an object-oriented language, and the reconfiguration engine that makes use of constraint solving and ideas from diagnosis in order to compute system changes. The key concept of SIMOL is the definition of basic and hierarchical components, which are used to represent the desired system. The behavior of a component has to be provided as set of equations. If a component is a subclass of another component, the equations of the superclass are taken together with the equations of the component in order to specify the component's behavior. In SIMOL, it is also possible to assign equations to specific behavior modes. Such a mode might represent a potential configuration like stating a component to be active or inactive in a certain configuration. Alternatively, a mode might represent the range of values assigned to a certain parameter.

Another feature of SIMOL is its ability to model the systems behavior over time. In this case, SIMOL allows for spec-

Michel Aldanondo and Andreas Falkner, Editors
Proceedings of the 15th International Configuration Workshop
August 29-30, 2013, Vienna, Austria

ifying state transfer functions. Such a function itself is a set of equations, that connect values of variables between one state and its successor state. SIMOL does not allow to deal with continuous time. Only systems which can be modeled using discrete time can be represented in SIMOL. The decision to restrict modeling to discrete time via states is due to the requirements of the smart metering application domain, where not the continuous evolution of parameters is important, but the their discrete values. Besides the model of the system, also the system's requirements can be modeled using SIMOL. Hence, every information needed for stating a reconfiguration problem can be formalized using SIMOL.

The reconfiguration algorithm is based on model-based diagnosis, where the idea is to select one mode for each component such that all system's requirements are fulfilled. This problem can be easily stated as constraint satisfaction problem. Hence, we make use of an available constraint solver for computing simulations and reconfigurations. In the following, we discuss the whole SIMOA system architecture, the SIMOL language, and the reconfiguration algorithm in more detail. A discussion on related research and a summary of the content conclude this paper.

2 The SIMOA architecture

In Figure 1, we depict the SIMOA system architecture. The architecture comprises at the highest level two parts: a graphical user interface (*SIMOA M2M GUI*) and the configuration core (*ConfigCore*). The latter is general and can be used in various applications, whereas the other is application specific and has to be tailored accordingly to the requirements. The configuration core itself comprises a compiler that translates models written in SIMOL into MINION constraints [Jefferson *et al.*, 2012; Gent *et al.*, 2006]. MINION is a constraint solver coming with its own constraint language, which is not easily accessible for non-experts in constraint solving. The reasons for choosing MINION were the easy integration into the program written in Java and the very good reasoning performance (being able to solve 8,000 constraints in less than 2 seconds). The MINION program is used in the reconfiguration engine *ReConf* together with the MINION constraint solver to compute valid reconfigurations, which are given back as *Results*.

The interface between the graphical user interface and the configuration core is represented by the SIMOL program and the results obtained from *ReConf*. The SIMOL program comprises the information necessary to specify the system to be reconfigured and the given pre-specified requirements. The reconfiguration result is basically nothing else than a set of possible component modes, that are necessary to fulfill the requirements, together with the computed values for the attributes. The presentation of these results to the user has to be implemented in the user interface and is application specific. In Figure 2, the current user interface of the SIMOA M2M application is given. This graphical user interface enables the user to specify a smart metering application, by placing the meters as well as the cells, which provide access to the mobile phone network, among other components at the appropriate positions. Moreover, the user might specify additional

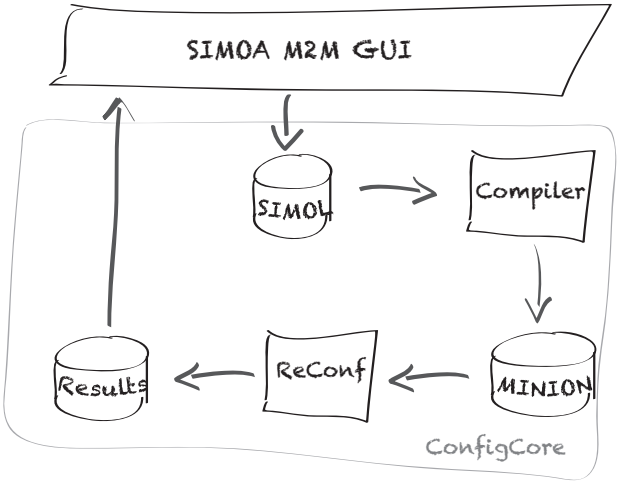

Figure 1: The SIMOA architecture

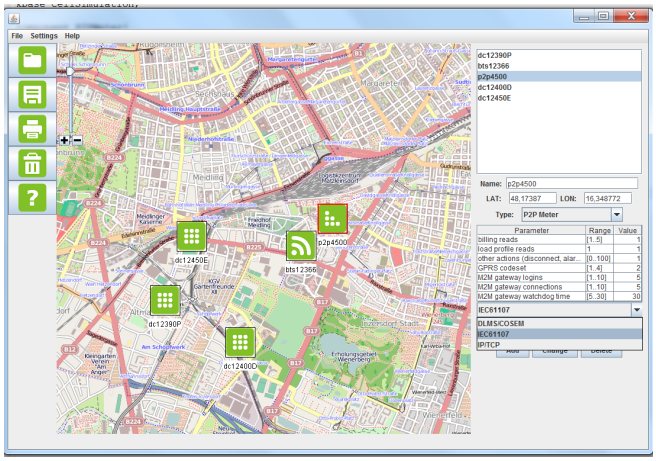

Figure 2: The M2M user interface for the smart metering application

parameters for components. In case of a reconfiguration, the GUI generates a SIMOL program that makes use of the components, their behavior and additional parameters. It is also worth noting that also the positions of the components in the map are used. For example, when specifying a base load (that represents all the non-smart-metering traffic) for the mobile phone network, the concrete assignment to cells is done considering the distance between the base load and the cell. If a base load is not within reach, there is no effect. If a base load might influence two or more cells, the load is assigned to each cell accordingly to their distance. For example, closer cells will have a larger percentage of the communication base load than cells that are more far away.

The *ConfigCore* of the SIMOA architecture is general and can be used in various reconfiguration applications. In this respect, we have conducted a series of experiments using, for instance, combinational circuits from the well-known ISCAS85 benchmark suite. Due to limitations of the *ReConf*

part, we do not handle generative constraints. Hence, building systems from scratch using the given constraints is not possible in SIMOA. However, to some extent, configuration of systems is possible by providing a model of a larger system, where parts can be activated or inactivated. In our application domain there is no impact due to restrictions, because the network, as well as the M2M application structure, are already known and only small deviations are possible. Therefore, providing information regarding structural system changes and changes in parameter is sufficient. In the next section, we discuss SIMOL in more detail. Moreover, we introduce a basic algorithm for reconfiguration using SIMOL programs. For more information regarding the application domain we refer the interested reader to [Nica *et al.*, 2012].

```
1.  kbase GPRSCell;
2.  component P2PMeter {
3.    attribute int mdist,codeset,mRate;
4.    constraints {
5.      mdist = {1..3};
6.      codeset = {1..4};
7.    }
8.  }
9.  component FPC {
10.   attribute int value;
11.   constraints(default) {
12.     value = 1;
13.   }
14.   constraints(x1) {
15.     value = {2..4};
16.   }
17.   constraints(unknown) {
18.
19.   }
20. }
21. component BTS {
22.   attribute int fpc;
23.   constraints {
24.     FPC fpc1;
25.     fpc = fpc1.value;
26.   }
27. }
28. component Cell {
29.   attribute int neededR, realR;
30.   constraints {
31.     BTS b1;
32.     P2PMeter s[100];
33.     realR=sum([s], mRate)/P2PNo;
34.     realR>=neededR;
35.   ..
36.   }
37.   transition {
38.     forall ( P2PMeter ) {
39.       if (mdist=1 and codeset=2 )
40.         codeset.next = {2,3};
41.       if (mdist=3 and codeset=2 )
42.         codeset.next = {2,1};
43.     }
44.   }
45. }
```

Figure 3: A (partial) SIMOL program

3 SIMOL syntax and semantics

As already said, SIMOL is an object-oriented programming language. Most of the basic features have been already described elsewhere [Nica and Wotawa, 2011; 2012b]. However, in order to be self-contained, we briefly introduce and discuss SIMOL's syntax and semantics. To be more accessible for non-experts in configuration and constraint solving, we decided to adopt the syntax of Java. The program depicted in Figure 3 is a partial model used in our M2M application domain. The program comprises 4 components, which model a Point-to-Point (P2P) individually addressable smart meter, a base transceiver station (BTS), the number of serving frequencies (FPC) and a mobile cell. Every component definition starts with declaring the name of the component. Within a component, its attributes, constraints, and transitions are defined. The latter is for defining the next state in order to model discrete time and state machine models.

Syntax: Since SIMOL has a Java-like syntax, most of the defined tokens are Java-like, i.e., identifiers for any type of components and attributes, integers, and boolean literals, separators, arithmetic and relational operators ($+, -, *, /, =, <$ $, >, <=, >=, !=$), comments and also reserved keywords. In addition, it is also possible to use physical units like Watt (W), or Ampere (A), etc., for a more realistic description. Another feature of the language is that the domain of the variables values can be restricted. In Line 15 of the program depicted in Figure 3 only the values 2, 3, and 4 are allowed for variable `value`.

Every SIMOL program comprises 3 sections: (1) a knowledge based declaration section (Line 1) for organizing the files similarly to Java packages, (2) an import declaration section where knowledge bases can be loaded, and (3) component definitions (Line 2 to 45). The first 2 sections are optional, whereas the component definition section is mandatory. Each component definition starts with the keyword **component** followed by the name of the component and with an optional **extends** followed by a comma-separated list of component names. If **extends** is used, we know that the new component has one or more super components from which constraints are inherited.

In every component definition, we firstly define the component's attributes after the **attribute** keyword. For example, in Line 3 the attributes `midst`, `code set`, and `mRate` are defined. All these attributes are of type integer (**int**). Besides attributes, constraints can be defined. There are two ways of doing this. Either we use the keyword **constraints** directly followed by a block in surrounding parentheses $\{\}$, or we use the same keyword **constraints** followed by a mode name under parentheses () and again a block statement. The first definition of constraints only allows for specifying a single component behavior. The other definition makes use of modes that are needed later on for configuration. For example, in Line 11 to 19 three modes for the component `FPC` are defined. The `default` mode sets the value of variable `value` to 1. The `x1` mode restricts the domain of `value`, where the constraint solver can select one value from the range $\{2..4\}$ when computing reconfigurations. The last mode (`unknown`) does not specify any value.

In the constraint section of a component definition the following types of statements are allowed in SIMOL:

- an *empty statement*: `;`,
- a *component instance declaration*, with the possibility of initializing its attributes. See for example Line 24 of the `GPRSCell` knowledge base, where a new component

`fpc1` is generated as an instance of component `FPC`. 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.

- an *arithmetic or/and boolean expression*
- a *conditional block* starting with the keyword **if** and optionally followed by an **else**.
- special functions like **forall**, **exist**, **sum**, **product**, **min**, and **max**, that allow to state constraints over an array of instances or values. For example, in Line 33 we sum the `mRate` attribute of all `P2PMeter` stored in variable `s`.

In addition, models written in SIMOL might be described over discrete time. For this purpose SIMOL makes use of the **transition** section. Within the transition section the new values of some, but not necessarily all variables, have to be defined. In our running example Line 37 to 43 define the next values of the `codeset` variables for all `P2PMeters`. In order to distinguish the new value of a variable in the successor state we make use of the keyword **next**. It is worth noting that in the transition section we can use all statements from the constraint section.

Semantics: The following informal definition of the semantics of SIMOL relies on mathematical equations. In particular the idea behind the semantics is to map all constraints that are assigned to one component to a set of equations. This also requires the combination of equations in case of multiple inheritance and component instances. In the first part of the definition of the semantics we ignore discrete time. We discuss this issue later in this section.

For each component C defined in SIMOL we assume a set of equations $constr_0(C)$, representing the set of constraints within the **constraints**(mode) $\{\ldots\}$ blocks. Then each constraint C_{mode} within a **constraints**($mode$) $\{\ldots\}$ block contributes to $constr_0(C)$, only if $mode$ is active. Hence, we can define this as a *conditional mode constraint* $C_{cond_{mode}}$: $if(mode\ is\ active)\ C_{mode}\ must\ be\ satisfied$ and therefore $constr_0(C)$ becomes:

$$constr_0(C) = \bigcup_{mode \in MODES(C)} constr_{mode}(C)$$

where $MODES(C)$ is the set of functional modes, defined for component C, and $constr_{mode}(C)$ is the set of *conditional mode constraints*.

Moreover, the component C also receives equations from its super components and the instances used in the component definition. Because of the possibility of having more than one instance of a component, we have to rename the variables used in the constraints of an instance. For this purpose, we assume a function $replace$ that takes constraints M and a name N and changes all variables x in M to $N.x$. Hence, the set of equations that corresponds to a particular component C is given by the following definition:

$$constr(C) = constr_0(C) \cup constr_I(C) \cup constr_V(C)$$

where $constr_I$ are the constraints inherited from the super components of C

$$constr_I(C) = \bigcup_{C' \in super(C)} constr(C')$$

$constr_V$ are the constraints coming from the components used in the definition of C (and requiring variable renaming using the function $replace$ that add a new pre-fix to the variables used in the components in order to make them unique)

$$constr_V(C) = \bigcup_{(C',N) \in vd\_inst(C)} replace(constr(C'), N)$$

Each constraint within the **constraints** $\{\ldots\}$ block contributes to $constr_0(C)$ as follows:

- C_{attr_val} : *attribute-equals-value/s* constraints, formulated with $=$ operator and applied on component attributes together with one single integer/boolean value or with a set of values;
- C_{attr_attr} : *attribute-equals-attr* constraints, formulated with $=$ operator and applied on component attributes;
- C_{num} : *numeric constraints*, formulated with basic relational operators over numeric expressions;
- C_{cond} : *conditional constraints*, $if(C_x\ is\ satisfied)\ C_y\ must\ be\ satisfied\ else\ C_z\ must\ be\ satisfied$;
- C_{exist} : *existence constraints*, $exist(at\_least(NR)\ |at\_most(NR)|NR, C, ATTR = VALUE)$, with the meaning that at most, at least or exactly NR components of a given type C have $ATTR = VALUE$.

  Note that the **forall**, **sum**, ... constraints are similarly defined.

How to handle time? Within the **transition** section we have constraints that define a relationship between the variables of a state and its successor state. In order to represent states, we introduce an index that is assigned to each variable used in $constr(C)$. Hence, what we do is to define constraints that hold in each state $i \in \{0, \ldots, s\}$, where s represents the maximum number of considered states within a reconfiguration model. These constraints are obtained from $constr(C)$ by adding an index i to the variables. We represent these constraints using the function $constr_i(C)$. For example, if `value = 1` is element of $constr(C)$, then `value_4 = 1` is element of $constr_4(C)$. Such constraints are valid within a state and therefore called *state constraints*.

In addition to state constraints we require *transition constraints*. The transition constraints can be easily computed from the **transition** section. In principle we make use of the same translation as in the **constraints** block, but also take care of the **next** attribute assigned to variables. If a variable v has such an attribute and we consider state i we replace

v.**next** with $v\_i + 1$. Variables v without the **next** attribute are changed to v_i. Hence, we obtain all transition constraints $trans_i(C)$ for state i and component C.

In summary, the constraints for the whole SIMOL program can now be obtained as follows:

$$constr = \bigcup_{i \in \{0,\ldots,s\}} \bigcup_{C} (constr_i(C) \cup trans_i(C))$$

Hence, the set of the obtained equations represents the behavior of the SIMOL program. It is worth noting that we took the semantic definition based on equations from the semantics of Modelica [Fritzson and Bunus, 2002]. In contrast to Modelica the handling of time is different as well as some of the functions that can be used within SIMOL.

4 Reconfiguration in SIMOA

Before stating a configuration algorithm, which is based on the diagnosis algorithm ConDiag [Nica and Wotawa, 2012a], we introduce and discuss some basic definitions. We first formalize the reconfiguration problem. The reconfiguration problem requires information on the current system and the new requirements. Note that the current system may fulfill the given requirements. In this case no changes of the current system are required. For the information needed of the current system we follow a component-oriented engineering approach and assume that the structure as well as the behavior has to be represented. The behavior of course has to capture those aspects relevant for configuration. In particular the functionality of the system has to be modeled.

In addition we assume that for each component of the system we know how to reconfigure the component. Here we borrow the idea coming from Model-Based Diagnosis (MBD) [Reiter, 1987; de Kleer and Williams, 1987] and introduce modes for components. Hence, every component has at least one mode. We assume the $default$ mode to be the standard mode of a component, and all other modes to be potential reconfigurations of this component. For simplicity, we introduce a function $modes : COMP \mapsto MODES$ mapping components from $COMP$ to their $MODES$. At least $default$ has to be element of $modes(c)$ for all components $c \in COMP$. The SIMOL language allows for specifying models of systems comprising components and their modes. For example, in lines 2–27 of our running example from Figure 3 the components `P2PMeter`, `FPC` and `BTS` are defined. `P2PMeter` and `BTS` only have one mode (i.e., the $default$ mode), whereas for `FPC`, 3 modes ($default$, $x1$, and $unknown$) are defined.

Besides the structure and behavior of the system, we have to define the new system requirements. System requirements in our context are nothing else than constraints, which a system has to fulfill. For example, we might say that the mobile phone network has to be capable of servicing 100 smart meters at once in a particular area, given the communication requirements of the smart meters. In the context of SIMOA this information again is specified using SIMOL. For example, lines 28–45 of the program from Figure 3 are for specifying exactly those requirements.

Definition 1 (Reconfiguration problem) *A reconfiguration problem can be defined as a tuple $(KB, COMP, MODES)$, where $KB = SD \cup REQ$ is the knowledge base comprising the model of the system SD and the requirements REQ, $COMP$ is a set of system components, and $MODES$ is the set of functional modes for the elements of $COMP$.*

The reconfiguration problem consists in searching for an assignment of modes to each component, such that the knowledge base together with this assignments is satisfiable.

As already mentioned, all information regarding the reconfiguration problem can be obtained from SIMOL programs. The program from Figure 3 allows us to derive the knowledge base KB, which is the set of equations $constr$ representing the semantics of the SIMOL program, the set of components $COMP = \{\texttt{P2PMeter}, \texttt{FPC}, \texttt{BTS}, \texttt{Cell}\}$, and the set of modes $MODES = \{default, x1, unknown\}$.

Definition 2 (Mode assignment) *Given a set of components $COMP$ and a set of functional modes $MODES$. A mode assignment M is a function $M : COMP \mapsto MODES$ mapping each component to one of its modes, i.e., for all $c \in COMP : M(c) \in modes(c)$.*

Having now all ingredients we are able to formally state a reconfiguration as follows:

Definition 3 (Reconfiguration) *Given a reconfiguration problem $(KB, COMP, MODES)$. A mode assignment M is a valid reconfiguration iff $KB \cup \{M(c) | c \in COMP\}$ is satisfiable.*

In reconfiguration we are interested in finding mode assignments that do not imply too many changes. Hence, we can use the number of required system changes to indicate the optimality of a reconfiguration. The number of changes necessary in a mode assignment is the number of used modes that are not equivalent to the $default$ mode.

Definition 4 (Number of changes) *Given a reconfiguration M for a reconfiguration problem $(KB, COMP, MODES)$. The number of changes (NOC) of M is equivalent to the number of modes in M deviating from the $default$ modes, i.e., $NOC(M) = |\{M(c) | c \in COMP \wedge M \neq default\}|$.*

We say that a reconfiguration M is optimal with respect to its NOC if it is minimal, i.e., there exists no other reconfiguration M' with $NOC(M') < NOC(M)$. This definition of minimality corresponds to cardinality minimality in diagnosis, which is different from the usually used subset minimality of diagnosis (see [Reiter, 1987]). However, for the purpose of reconfiguration minimality based on cardinality seems to be a better choice.

After stating the underlying definitions we introduce an algorithm for reconfiguration that is based on ConDiag [Nica and Wotawa, 2012a]. Computing reconfigurations in our context is nothing else than searching for minimal mode assignments, i.e., mode assignments that are as close to the original assignments as possible. When assuming that small changes lead to a satisfiable knowledge base, it would be good to start search considering small deviations of mode assignments from the $default$ mode first. The number of changes

can be increased if no solution is found. Therefore, an iterative algorithm seems to be sufficient.

Algorithm 1 reconfig$(KB, COMP, MODES, n)$

Input: A reconfiguration problem $(KB, COMP, MODES)$ and the maximum NOC n
Output: All minimal reconfigurations (up to the predefined cardinality n)

1: **for** $i = 0$ **to** n **do**
2: $CM = \{|\{M(c)|c \in COMP \wedge M \neq default\}| = i\} \cup KB$
3: $S = \mathcal{P}(\textbf{CSolver}(CM))$
4: **if** $S \neq \emptyset$ **then**
5: **return** S
6: **end if**
7: **end for**
8: **return** $\emptyset$

Algorithm 1 **reconfig** takes a reconfiguration problem and a maximum number of changes and computes all minimal reconfigurations. Algorithm 1 is an iterative algorithm that starts with no changes of modes and continues search if necessary up to the predefined value n. The termination criteria before reaching n is given in Line 4, where an non-empty solution obtained from the satisfiability check is returned as result. In case no solution is found the empty set is returned (Line 8). The **CSolver** is a constraint solver taking a set of constraints CM and is expected to return a set of mode assignments if a satisfiable solution can be found. Otherwise, the empty set is returned indicating that no reconfiguration of the given size is possible. The function $\mathcal{P}$ is assumed to map the output of the solver to a set of solutions.

In the SIMOA prototype implementation we make use of the MINION [Jefferson *et al.*, 2012; Gent *et al.*, 2006] constraint solver for this purpose, but every other constraint solver would also be sufficient providing that it is capable of handling the constraints stored in CM. Line 2 of Algorithm 1 adds a new constraint to the model stating that we are interested in finding solutions that comprise exactly i modes that are not equivalent to $default$.

Algorithm 1 obviously terminates assuming that **CSolver** terminates. The complexity is of $O(n \cdot k)$ where k is the complexity of **CSolver**. In the worst case searching for solutions for a finite constraint satisfaction problem is exponential in the size of the problem. Therefore, **reconfig** is also exponential in the worst case. However, in practice solutions can be found in a much faster way. See for example the results reported in [Nica and Wotawa, 2012a] and more recently [Nica *et al.*, 2013]. In these paper search for solutions up to a size of 3 is within seconds even for constraint satisfaction problems comprising up to 3,800 constraints. Although these results are for diagnosis, they also can be applied to configuration because of the similarity of the algorithms.

5 Empirical results

In this section we report on first empirical results obtained using a SIMOL model of our application domain, i.e., smart metering. The SIMOL source code has 95 lines of code, describing a model with one, two, or three cells, where each cell contains from 7 up to 100 `P2PMeter` components. When compiling the SIMOL program to its MINION representation, considering at maximum 5 states, we obtain up to 2,387 variables and 7,320 constraints, depending on the the number of smart meters considered. In principle, there are many possibilities of mapping SIMOL to MINION and also for computing solutions for a given maximum number of changes NOC. In the following, we discuss the encoding of SIMOL modes within MINION and show that the choice of certain MINION parameters influence the computation of reconfigurations substantially.

The mode encoding in MINION is rather straightforward. In principle, a mode of a component can be either active or inactive. Therefore, we map each mode $mode_x$ to a Boolean variable in MINION, which is 1 (true) if the corresponding component is in mode $mode_x$, or 0 (false) otherwise. In order to compute a solution for a particular NOC we have somehow to maximize the number of $default$ modes in the solution. In the first version of our implementation we used the $MAXIMISING$ option of MINION for this purpose. In addition, we decided to control the way the solver searches for a solution also by directly specifying the instantiation order for the MINION variables representing a mode. Hence, we used the MINION variable ordering ($VARORDER$) as well as the corresponding value ordering ($VALORDER$) with the following settings: for all the $default$ mode variables their values should be searched in descending order, whereas for the other mode variables the searching should be done in ascending order. The intuition behind is to prefer solutions with more $default$ modes to be true over the other solutions.

For the experiments we made use of a notebook with Intel(R) Core(TM) i7 CPU 1.73 GHz and 4 GB of RAM running under Windows 7. We obtained the results presented in the upper diagram of Figure 4 for models containing a rather small number of `P2PMeters` ranging from 7 to 50. It is worth noting that when using the $MAXIMISING$ function the measured running times exceeded 300 seconds for more than 100 meters (which is unacceptable in some situations). Hence, we decided to use only the $VARORDER$ and $VALORDER$ and ignore the $MAXIMISING$ function. From the results depicted in the bottom diagram of Figure 4 we see a substantial improvement in the measured running time. Note that the obtained results without $MAXIMISING$ were always correct.

From the diagram at the bottom of Figure 4 we can extract two observations. First, when checking only that a given system fulfills the requirements, the running time even in case of 100 `P2PMeters` is within seconds. Second, even for a NOC of size 6 the reconfiguration time never exceeds 25 seconds. Since, for the application domain these running time results are sufficient and the proposed approach is feasible.

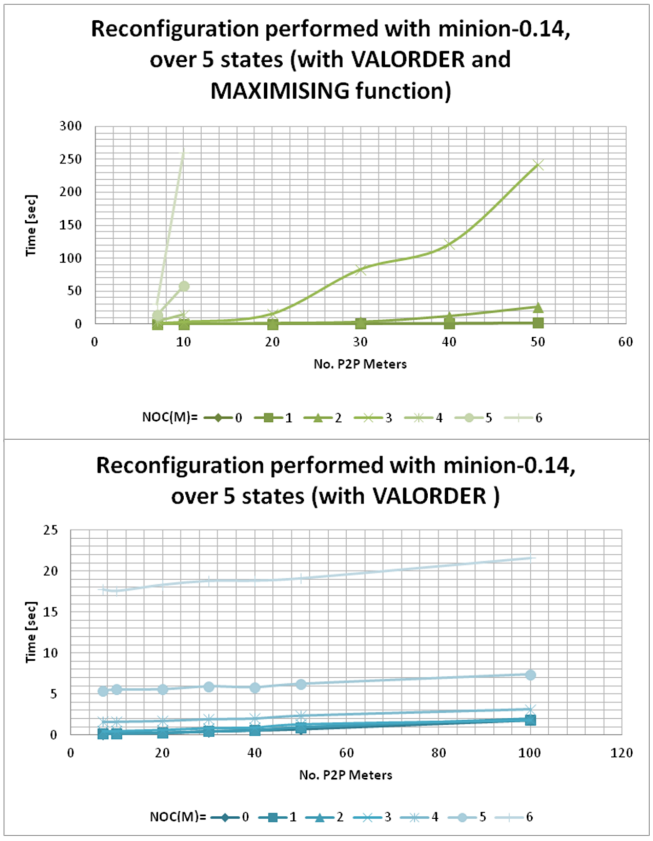

Figure 4: Comparing running times for reconfiguration, when the Integer variable domain is fixed to [0..20,000] and the number of solutions is limited to 1.

6 Related research

The idea of using constraint solvers for configuration is not new. In [Haselböck and Stumptner, 1991] the authors formalize the configuration and design problem as constraint satisfaction problem. Similarly in [Stumptner and Wotawa, 1998] the authors discuss the use of constraint solving for reconfiguration and in particular parameter reconfiguration in detail. The latter also makes use of model-based diagnosis for obtaining reconfigurations. In contrast to these previous papers our reconfiguration algorithm although relying on constraint solving is different because we compute configurations directly without making use of hitting set computation [Reiter, 1987; Greiner *et al.*, 1989] or other means for computing diagnoses [de Kleer and Williams, 1987].

The application of configuration for solving problems in the engineering domain has a long tradition. In [Stumptner *et al.*, 1994; Fleischanderl *et al.*, 1998] the authors describe the use of generative constraints for configuring large technical systems comprising thousands of components within a reasonable amount of time. Other applications include the use of configurations for web services [Felfernig *et al.*, 2002], technical products [John and Geske, 1999; John, 2000], and even telecom systems [Emde *et al.*, 1996]. Haag [Haag, 2010] discussed experiences obtained from product configuration. Although, configuration of technical products from various domains is more or less a well developed and researched field, the application to the M2M domain that requires models from the application itself and the used communication infrastructure is to our knowledge new. Moreover, besides the logical model also spatial information has to be integrated accordingly in order to come up with a correct model. The SIMOA approach provides a good bases because it allows to specify constraints dealing with Boolean and Integer values as well as discrete time. Moreover, also arrays can be used for modeling. Extensions in the direction of handling floats or strings can be implemented but require to change the underlying reasoning engine.

There are of course many languages for simulation like Modelica [Fritzson and Bunus, 2002] or Simulink [Henson, 2005] used in industry. However, these languages are mainly optimized towards simulation and therefore can be hardly used for reconfiguration. In particular such languages do not allow under-constrained models, which are necessary for our purpose when searching for appropriate modes that do not contradict the given requirements while ensuring that the requirements can be fulfilled. There are some similarities between Modelica and SIMOL but also many differences including the tight integration of component modes and the handling of discrete time.

Our previous papers mainly deal with either the application domain [Nica *et al.*, 2012] or the SIMOL language [Nica and Wotawa, 2011; 2012b]. In contrast to the latter paper, we extend the SIMOL language using the **transition** block in order to handle discrete time in the underlying models. Moreover, we discuss the algorithm for configuration in more detail in this paper.

7 Conclusions

In this paper we discussed the underlying language, definitions, and algorithms of the SIMOA approach to reconfiguration. Although, the approach has been applied to the machine-to-machine communication domain, it is not restricted to this domain. Any reconfiguration problem that can be represented using the underlying modeling language SIMOL can also be solved using the proposed SIMOA approach. SIMOL itself is an object-oriented programming language where components can be defined. The syntax of SIMOL is close to Java. The semantics has been mainly taken from the modeling language Modelica. Within the developed SIMOA prototype SIMOL is converted in MINION constraints. Hence, MINION is used as underlying constraint solver. This again does not restrict the approach since changing constraint solvers is still possible. Only, the conversion of SIMOL has to be adapted.

Besides SIMOL we also discuss the basic definitions of reconfiguration and state an algorithm that allows to find minimal reconfigurations up to a predefined size. Size in this context is defined as number of necessary changes of the system in order to fulfill all constraints. The reconfiguration algorithm derives solutions directly from the constraints (i.e., equations coming from SIMOL). This distinguishes this approach from other similar approaches where search for valid configurations is often based on conflicts and conflict resolu-

tion. First empirical results indicate that computation is sufficiently fast and that the results are within expectations. In the future it is planned to further evaluate the approach.

Acknowledgement

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.

References

[de Kleer and Williams, 1987] Johan de Kleer and Brian C. Williams. Diagnosing multiple faults. *Artificial Intelligence*, 32(1):97–130, 1987.

[Emde *et al.*, 1996] Werner Emde, Christian Beilken, Josef Bording, Wolfgang Orth, Ulrike Petersen, Jörg Rahmer, Michael Spenke, Angi Voss, Stefan Wrobel, and Schlo Birlinghoven. Configuration of Telecommunication Systems in KIKon, 1996.

[Felfernig *et al.*, 2002] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, and Markus Zanker. Semantic Configuration Web Services in the CAWICOMS Project. In *ISWC '02 Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 192–205, 2002.

[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.

[Fritzson and Bunus, 2002] Peter Fritzson and Peter Bunus. Modelica - a general object-oriented language for continuous and discrete-event system modeling and simulation. In *Proceedings 35th Annual Simulation Symposium*, pages 365–380, 2002.

[Gent *et al.*, 2006] I. P. Gent, C. Jefferson, and I. Miguel. MINION: A Fast, Scalable, Constraint Solver. *17th European Conference on Artificial Intelligence*, ECAI-06, 2006.

[Greiner *et al.*, 1989] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.

[Haag, 2010] Albert Haag. Experiences with Product Configuration? http://www.minet.uni-jena.de/dbis/lehre/ss2010/konfsem/, 2010.

[Haselböck and Stumptner, 1991] Alois Haselböck and Markus Stumptner. Configuration and design as a constraint satisfaction task. In *Artificial Intelligence in Design – Proceedings of the Workshop of the 12th International Joint Conference on Artificial Intelligence*, Sydney, Australia, August 1991. Also appeared as Technical Report DBAI-CSP-TR 91/1.

[Henson, 2005] William Henson. Real time Control and Custom Components in the Matlab Environment. Technical report, 2005.

[Jefferson *et al.*, 2012] Christopher Jefferson, Lars Kotthoff, Neil Moore, Peter Nightingale, Karen E. Petrie, and Andrea Rendl. TheMinion Manual, Minion Version 0.15. http://minion.sourceforge.net/, 2012.

[John and Geske, 1999] Ulrich John and Ulrich Geske. Reconfiguration of Technical Products Using ConBaCon. In *Proceedings of WS on Configuration at AAAI-99*, Orlando, 1999.

[John, 2000] Ulrich John. Solving large configuration problems efficiently by clustering the ConBaCon model. In *Proceedings of the 13th international conference on Industrial and engineering applications of artificial intelligence and expert systems: Intelligent problem solving: methodologies and approaches*. Springer-Verlag New York, Inc., 2000.

[Nica and Wotawa, 2011] Iulia D. Nica and Franz Wotawa. SiMoL- A Modeling Language for Simulation and (Re-)Configuration. In *Workshop on Configuration*, pages 40–43, 2011.

[Nica and Wotawa, 2012a] Iulia D. Nica and Franz Wotawa. ConDiag – Computing minimal diagnoses using a constraint solver. In *Proc. 23rd International Workshop on Principles of Diagnosis (DX)*, 2012.

[Nica and Wotawa, 2012b] Iulia D. Nica and Franz Wotawa. The SiMoL Modeling Language for Simulation and (Re-) Configuration. In *Proc. Conference on Current Trends in Theory and Practice of Informatics (SOFSEM)*, pages 661–672, 2012.

[Nica *et al.*, 2012] I. D. Nica, F. Wotawa, R. Ochenbauer, C. Schober, H. Hofbauer, and S. Boltek. Model-based simulation and configuration of mobile phone networks - the SIMOA approach. In *Proc. of the ECAI 2012 Workshop on Artificial Intelligence for Telecommunications & Sensor Networks*, pages 12–17, 2012.

[Nica *et al.*, 2013] Iulia D. Nica, Ingo Pill, and Thomas Quaritsch Franz Wotawa. The Route to Success - A Performance Comparison of Diagnosis Algorithms. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2013.

[Reiter, 1987] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.

[Stumptner and Wotawa, 1998] Markus Stumptner and Franz Wotawa. Model-based reconfiguration. In *Proceedings Artificial Intelligence in Design*, Lisbon, Portugal, 1998.

[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.