The SiMoL Modeling Language for Simulation and (Re-)Configuration

Iulia Nica and Franz Wotawa
Technische Universität Graz, Institute for Software Technology

Index

- Motivation
- SiMoL Definition
  - SiMoL Syntax
  - Semantics of SiMoL
- An Example
- SiMoL in Practice
- Conclusions and Future Work

Motivation

- *Simulation* and *configuration* play an important role in industry
- For *configuration*: currently, a wide range of knowledge representation paradigms is used
- For *simulation*: we recall Matlab/Simulink and Modelica
- Our scope is: combine the two different directions in one language - SiMoL

Motivation (2)

- Why a new modeling language?

  (1) We want an easy to learn and use language → the existing configuration languages *are not*

  (2) We want a language that can be used for both purposes → *neither* Matlab/Simulink or Modelica *can be used*

SiMoL – a declarative object-oriented language, designed with multiple inheritance and which allows for stating constraints between variables.

SiMoL Syntax

- SiMoL uses a Java-like syntax
- Beside the basic data types like integer and boolean, SiMoL makes use of component instances
- SiMoL offers support for using *units of measurement*
- Another feature of the language is the *initialization of attributes with integer valued ranges*

SiMoL Syntax (2)

- We mention the built-in functions *min, max, sum, product*
- The taxonomy relations are represented through the inheritance mechanism
- The implementation of SiMoL is based on the Minion constraint solver http://minion.sourceforge.net/files/Manual012.pdf/

SiMoL Syntax (3)

- A program written in SiMoL contains basically 3 sections:
  - a *knowledge base declaration section:* `kbase X1Model;`
  - an *import declaration section:* `import X0Model.*;`
  - a *component definition section,* that contains:
    a. *the attribute declaration block*
    b. *the constraints block*

```
component  X1{
 attribute int a1,a2;
 constraints{
  a1={4,6};
  a2={4..6};
  …
  }
}
```

Semantics of SiMoL

- Here we rely on mathematical equations
- We map every statement to a mathematical equation and combine these equations for a component, taking care of *component inheritance* and *component instances (1)*
- There can be more than one instance of a component ➔ the variables used in the constraints of an instrance have to be renamed ➔ we assume the *replace* function *(3)*

Semantics of SiMoL (2)

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

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

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

(4) $$constr_0 = \bigcup_i C_i$$

Semantics of SiMoL (3)

- A constraint *Ci*:
  - *Cattr_val: attribute-equals-value/s constraint*
  - *Cattr_attr: attribute-equals-attribute constraint*
  - *Cnum: numeric constraint*
  - *Ccond: conditional constraint*

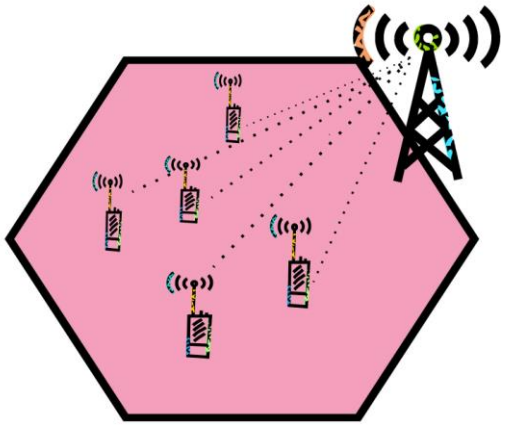
if (Cx is satisfied*) Cy* must be satisfied *else Cz* must be satisfied

  - *Cexist: existence constraint*

exist(at_least(NR)| at_most(NR)| NR, C, ATTR=VALUE)

An Example

- A (simplified) mobile network cell with embedded machine-to-machine communication functionality



An Example (2)

```
kbase CellSimulation;
component  SmartMeter{
   attribute int dataAmount ;
   constraints{
     dataAmount={10..30} KB/day ;
   }
}
component DataConcentrator {
   attribute  int dataAmount, maxDevicesManaged;
   constraints{
     dataAmount={1000..3000} KB/day ;
     maxDevicesManaged=300;
   }
}
component Cell {
   attribute int baseLoad, P2PNo,  DataCNo;
   constraints{
     baseLoad={15..30} %;
     P2PNo= {0..1000};
     DataCNo= {0..10};
   }
}
```

```
component  MyCell  extends  Cell {
   attribute int total;
   constraints {
     P2PNo=21;
     SmartMeter smSet1[20];
     SmartMeter smSet2[30];
     SmartMeter s1;
     DataConcentrator d1;
     forall(SmartMeter) {
         dataAmount=20  KB/day;
     }
    if(P2PNo>20)
       20+sum([smSet1,  smSet2, s1, d1],
            dataAmount) <1000 KB/day;
    else
       10+max([smSet1,smSet2],
              dataAmount) >200  KB/day
  }
}
```

SiMoL in practice

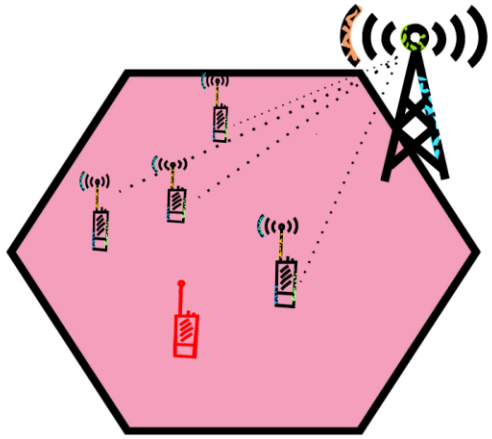
- We present two scenarios from the same domain

- The cell serves a given number of smart meters, which communicate via a given number of channels and there are new desired requirements, that must be fulfilled.

- *First Scenario*:

  - We consider the possibility

  to inactivate specific smart meters, if the constraints stated over their attributes are no longer satisfied.

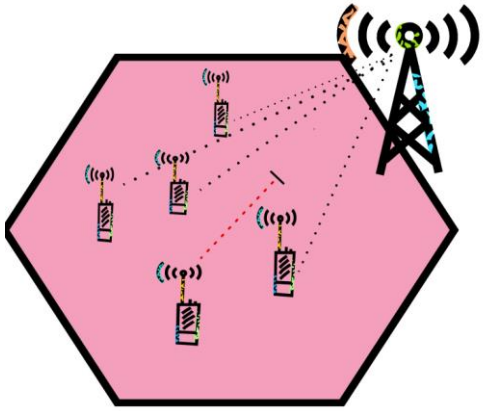


SiMoL in practice (2)

- The generated MINION model for the case with 51 smart meters contains 270 constraints and the running time for computing the minimal cardinality reconfiguration (3, in our case) is 234 ms.

- *Second Scenario*:

  - We assume the meters should be all active (not changeable), but the channels may be active, permitting data communication, or inactive (blocked for transmission).



SiMoL in practice (3)

- *Second Scenario*:
  - At least one channel is available for transmitting data, whereas the rest of the channels must be activated, if a constraint in the MINION model is violated.
  - The MINION model, generated for the case with 300 smart meters - contains 690 constraints and the running time for computing the minimal cardinality reconfiguration (1, in this case) is 218 ms.

Conclusions and future work

- In our implementation, the simulation task consists in checking whether the current configuration (the system modeled in SiMoL) is consistent.
- We may get two possible outcomes: system consistency – the simuation was successful or system inconsistency – there exist no values in the attributes domains that can satisfy all the component requirements.
- Currently, we use the SiMoL modeling approach in our simulation tool, designed with both desktop and web user interfaces.
- Future work includes optimizing the SiMoL syntax, the MINION models and providing a complete configuration algorithm.

Thank You!

Q & A