Supporting Model-Based Diagnostics with Equation-Based Object Oriented Languages

Peter Bunus[1] Karin Lunde[2]

[1]Department of Computer and Information Science, Linköping University, Sweden,
petbu@ida.liu.se
[2]University of Applied Sciences Ulm, Germany,
k.lunde@hs-ulm.de

Abstract

The paper focuses on the application of equation-based object oriented languages to creating models for model-based diagnosis. We discuss characteristics and language constructs essential for diagnostic purposes. In particular, we describe the main features of the declarative modeling language Rodelica, which is based on the well-known language Modelica but enhances it with additional features related to diagnosis. Rodelica is used in a commercial model-based diagnosis tool to build and exploit complex diagnostic models of industrial size. Developed models can be used in an interactive diagnostic process as well as for the generation of more compact forms of diagnostic knowledge like diagnostic rules or decision trees which are popular for on-board diagnostics or trouble-shooting in the service bay. A case study concludes the paper, illustrating those applications and emphasizing their implications for the language itself.

Keywords: model-based diagnostics, Modelica, Rodelica, constraint propagation, interval arithmetic, failure mode, decision tree.

1. Introduction

In today's global economy, time to market decreases due to the global competitive pressure and due to the increased customer's demand for new products with new and improved functionality. A company's market share depends largely on its capability to satisfy the ever increasing customer requirements with respect to functionality and reliability. The shortened development times and the increasing complexity of the products - as indicated by the significantly increasing electrical and electronic content - may lead to difficulties if not handled appropriately. Despite careful development and construction, some of these components may eventually fail. To avoid unnecessary damage, environmental or financial, there is a need to locate and diagnose these faults as fast as possible. This can be done with a diagnostic system, which should alert the user if there is a fault in the system and, if possible, indicate the reason behind it.

The importance of the ability to perform high quality and high reliability diagnostics on a system lies in:

- Lowering the repair and maintenance time of the real system which results in lower maintenance costs and increased customer satisfaction.
- Lowering the number of sound components that are replaced erroneously during maintenance and repair.
- Lowering the downtime and non-operational time of critical systems.

In the last decade, model-based technology in diagnosis matured so far that it was transferred from academic research into real applications. It provides an alternative to more traditional techniques based on experience, such as rule-based reasoning systems or case-based reasoning (CBR) systems. Early model-based diagnosis tools include MDS (Mauss, et al. 2000 [11]), RAZ'R (Sachenbacher, et al. 2000 [15]) and RODON (Lunde, et al. 2006 [9]).

In early diagnosis systems, the knowledge about interrelations between observations (symptoms), possible failure causes, and repair actions were encoded in simple rules like this: "*If the back rest of a passenger seat system (business class) cannot be moved to the upright position although the forward activation button is pushed then the hydraulic controller may be defective or the forward activation button might be disconnected*". Rule-based systems are fairly simplistic. As it can be seen from the previous example, they provide a set of assumptions and a set of rules that specify how to act on the set of assertions. The advantage of the rule-based systems is that they are very efficient with respect to memory and computing time. For this reason, they are still widely used in practice, especially for on-board diagnostics when the lack of computational power is an issue. However, in modern systems on-board rules are automatically generated by model-based tools, which are able to produce more complete and systematic rule sets compared to the traditional hand-written ones.

In a case-based reasoning system, system expertise, fault detection and fault isolation is embodied in a library of past cases rather than in classical rules. When presented with a new target problem, a case-based reasoning

2nd International Workshop on Equation-Based Object-Oriented Languages and Tools. July 8, 2008, Paphos, Cyprus.

Copyright is held by the author/owner(s). The proceedings are published by Linköping University Electronic Press. Proceedings available at:
http://www.ep.liu.se/ecp/029/

EOOLT 2008 website:
http://www.eoolt.org/2008/

system will first attempt to retrieve from the case data base a similar case that is relevant to solve the target problem. For a diagnostic problem, a case usually consists of a symptom, and a repair solution. The foundations of cased-based reasoning systems are described by Pal and Shiu 2004 [13], and an overview of industrial cases is given by Althof, et al. 1995 [1].

In comparison, the model-based approach tries to use the knowledge incorporated in a model to derive the symptom-failure associations as well as appropriate repair actions automatically. The existence of a modeling language that can be used for capturing model knowledge for diagnostics purposes is central for model-based-diagnosis. Early model-based diagnosis systems used traditional general purpose programming languages for specifying models. However, in some aspects, general purpose programming languages are inadequate to the task of formalizing significant domains of engineering practice. They lack the expressiveness and power of abstraction required by engineers. In this paper, we propose a declarative equation-based language called Rodelica. It is derived from the standardized modeling language Modelica (Modelica Association 2007 [2]) and differs mainly in the behavior representation, by using constraints rather than differential equations. This deviation is due to the requirements of model-based diagnosis.

The rest of the paper is organized as follows: In Section 2, we give a brief description of the principles of model-based diagnosis, by presenting some classical textbook examples to illustrate and explain the main concepts. In Section 3, we introduce the Rodelica language, and we highlight the main language constructs that facilitate various failure analyses. The extended case study presented in Section 4 illustrates the potential and the benefits of the Rodelica modeling language. The model presented in the paper was built using RODON, a commercial model-based reasoning (MBR) system. The Rodelica language is an essential part of RODON's integrated modeling environment. We show how, from the developed model, we are able to automatically derive decision trees for troubleshooting of the system in the workshop, and decision rules for on-board diagnosis. We will also illustrate an interactive model-based diagnostics process in which additional measurements are proposed in case that the initial diagnosis does not result in a single candidate as a root case. Finally, Section 5 presents our conclusions.

2. Principles of Model-Based Diagnosis

The basic principle of model-based diagnosis consists in comparing the actual behavior of a system, as it is observed, with the predicted behavior of the system given by a corresponding model. A discrepancy between the observed behavior of the real system and the behavior predicted by the model is a clear indication that a failure is present in the system. Diagnosis is a two stage process: in the first stage, the error should be detected and located in the model, and in the second stage, an explanation for that error needs to be provided. Diagnoses are usually performed by analyzing the deviations between the nominal (fault free) behavior of the system and the measured or observed behavior of the malfunctioning system.

In Figure 1, a model of a real system (an airplane passenger seat system) is depicted at the lower left corner. It might contain, for example, the behavior of the mechanical components incorporated in the seats, or the behavior of the in-flight entertainment system, or both. Note that, like all models, the model is only an abstraction of the real system (depicted at the upper left corner) and can be incomplete. The granularity of the model and the amount of information and behavior that is captured into it will directly influence the method employed by the reasoning engine as well as the precision of the diagnostic process.

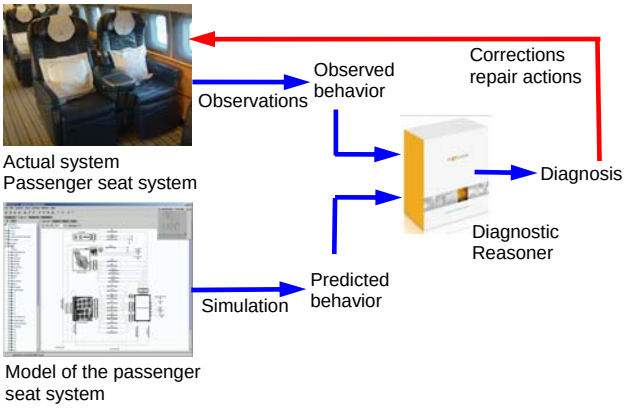

Figure 1. Basic principle of model-based diagnosis

As a general rule, the models are built to enable the identification of failed least repairable units (LRUs). Once a model of the real system is built, simulation or prediction can be performed on the model. The predicted behavior, which is the result of the simulation, can then be compared with the observed behavior of the real system. This comparison is usually done by a reasoning engine (in our case RODON) that is able to detect discrepancies and also to generate and propose corrective actions that need to be performed on the real system to repair the identified fault. We should note that the process of diagnosis (incorporated in the diagnostic reasoner) is separated from the knowledge about the system under diagnosis (the model). This ensures that the model can be reused for other purposes as well, such as optimization and reliability analysis.

The main ideas of model-based reasoning can be illustrated by the following simple multiplier-adder circuit taken from Kleer and Kurien 2003 [7].

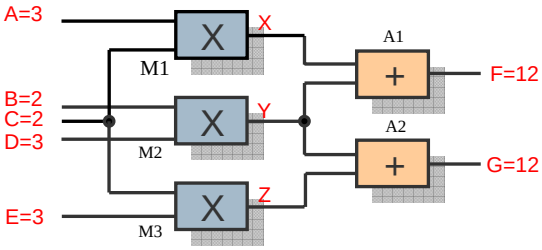

Figure 2. Multiplier-Adder circuit

The inputs to the system are `A`, `B`, `C`, `D` and `E`, while the outputs of the system are `F` and `G`. `X`, `Y` and `Z` are inter-

mediary probing points in the system. If requested, the user can perform an observation and measure the value in these points. If the system is supplied with the following input set: A=3, B=2, C=2, D=3, E=3, then the expected output should be F=12 and G=12. The output is calculated by using a deduction-based reasoning, first computing the intermediate results X=6, Y=6, and Z=6 at the output of the multiplier gates, which then are becoming inputs to the adder gates. For computing the results, a simple inference engine like the ones used by most simulation environments is enough to perform the calculations. Note that the behavior of the components is formulated in terms of relations between model variables which are called constraints. In general, constraints are not directed (unlike assignments), and may be of various nature, e.g. equations, inequalities, or logical relations.

A *conflict* is any discrepancy or difference between the prediction made by the inference engine and the observed behavior. Now let us suppose that by observing the real system one notices that the output value of F is 10, which is different from the expected calculated value of 12. In our case, the observation F=10 leads to a conflict that will be the triggering point for the diagnostic engine to indicate that something is wrong. It means that one of the components in the system does not work correctly, in other words: there is a contradiction between the assumptions and the observed behavior. In the next step, the diagnostic reasoner should find an explanation for the contradictory observation.

A possible explanation of this behavior might be that adder A1 or the multiplier M1 is defective which might cause the defective output F=10 (as depicted in Figure 3).

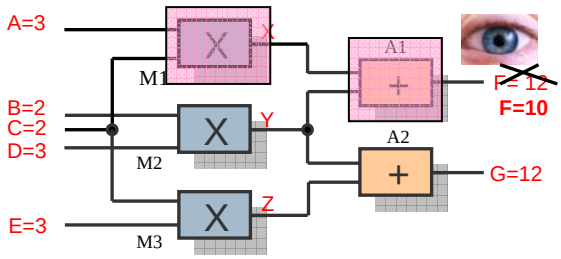

Figure 3. A defective adder A1 or multiplier M1 might explain the wrongly computed value F=10

A wrongly computed output by the adder A1 might be due to the fact that the adder itself is defective, in which case we can add the adder to the possible list of candidates, or due to the wrong inputs received by the adder. It should be noted that the inputs of adder A1 are the outputs of multiplier M1 and M2. The output of M2 is also input to the second adder A2 that computes the value of G. Since the value of G was correctly computed, we can exclude M2 from the suspected candidates. A defective M2 would certainly influence the computed value of G. Therefore, with the current information about the system we can conclude that either M1 or A1 is defective.

So far, we have considered only single faults in our multiplier-adder circuit. Considering multiple simultaneous faults will give us a new set of candidates, as depicted in Figure 4.

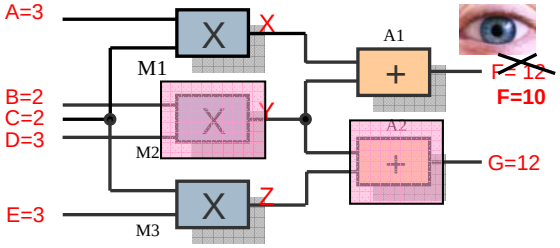

Figure 4. Multiple fault: Both M2 and A2 are defective

Let us analyze the set of multiple fault candidates shown in Figure 4 and see if it is consistent with the observation. The multiplier M2 is defective, which means that it will produce a wrong output, which will be input to the adder A1. With a wrong input, the adder A1 will produce a wrong output, which might explain that we observe F = 10. Moreover, the output of M2 is also input to the adder A2, which is also defective. It might happen that the defect inside A2 compensates the defect of the multiplier M2, and by chance, it produces a good value for the output G. With the given information, this is what we can conclude about the system. This reasoning method is called abductive reasoning. It starts from a set of accepted facts and infers their most likely explanations in the form of hypotheses. If at a later time, new evidence emerges which disagrees with a hypothesis, this hypothesis will be proven false and must be discarded. In our example, the hypothesis "M2 and A2 are faulty simultaneously" might be disproved by an additional measurement of the value Y = 6. But without additional information about the system, it is a valid hypothesis that explains the abnormal observation. By using the same type of reasoning, more hypotheses can be advanced, e.g. “M2 and M3 are simultaneously faulty”.

3. Rodelica, a Modeling Language for Diagnosis

For modeling diagnosis-related problems, like the example presented in the previous section, we describe the Rodelica language which is strongly related to the equation-based object-oriented modeling language Modelica. Rodelica was first proposed by Lunde 2000 [8]. Afterwards, it was implemented, and it is now the modeling language of the commercial model-based reasoning tool RODON. Since 2001, Rodelica was exploited in a large number of industrial-size projects.

The class concept of Rodelica is identical to the class concept of Modelica. However, we added set-valued data types as well as a new behavior description with semantics which differ from the equation-based behavior description in Modelica. Those deviations are motivated by the requirements of applications in model-based diagnosis. We describe the most important diagnosis-related semantic features of the Rodelica language in the following subsections.

3.1 The need for over-constrained systems

Modeling and simulation environments associated to traditional equation-based modeling languages like Modelica (Modelica Association 2007 [2]) , gProms proposed by Barton and Pantelides 1993 [3], Ascend proposed by Piela, et al. 1991 [14] or VHDL-AMS (Christen and Bakalar 1999 [6]) use numerical solvers for performing simulations. They rely on the fact that the system of differential equations extracted from the model is structurally and numerically nonsingular. The structural singularity checks whether the system of equations is well-posed or not, but cannot guarantee anything regarding existence or uniqueness of the solution. For this reason, the structural consistency checking is considered as a preprocessing phase to the more powerful notion of numerical singularity. The need to ensure that a system of equations extracted from the model is structurally nonsingular imposes some additional restrictions on the semantics of traditional equation-based object-oriented languages. For example, a necessary but not sufficient condition for ensuring the structural nonsingularity is that the number of equations must be equal to the number of variables. The second necessary condition is that the sparsity matrix associated to the structural Jacobian with respect to the higher order derivatives in the system can be permuted in such a way that it has a non-zero free diagonal. This restriction, imposed by the existence of a numerical solver for computing the solution of the equations, makes it impossible to formulate over- or under-constrained models. Models formulated in traditional equation-based languages need to be well-constrained. Several methods have been proposed to check the structural nonsingularity of the underlying system of equations associated to a model built using a traditional equation-based language. For the Modelica language, Bunus and Fritzson 2004 [5] proposed a graph theoretical approach for checking the structural nonsingularity and for debugging over- or under-constrained systems. Broman, et al. 2006 [4] proposed a concept called structural constraint delta to determine over- and under-constrained systems of equations in models by using static type checking and a type inference algorithm. Recently, additional restrictions have been added into the Modelica language in order to ensure that each model is "*locally balanced*", which means that the number of unknowns and equations must match on every hierarchical level. The rationale behind these restrictions introduced in Modelica 3.0 is presented by Olsson, et al. 2008 [12].

As we have seen in Section 2, a model-based diagnosis reasoning algorithm is triggered by conflicts. In Section 2 Figure 3, we present a situation in which the observed value `F` was equal to 10 compared to the computed value `F=12`. A model-based diagnosis system should have the possibility to specify and enter symptoms consisting of observations. Adding an observation to the model will automatically add an extra constraint (equation) making the model over-constrained. Model-based diagnosis systems like RODON use constraint solvers for performing diagnosis tasks that do not require the model to be well-constrained. It should be also noted that the main task of a model-based diagnosis system is to compute a diagnosis and not to perform a simulation.

The inference engine (constraint solver) will make use of the constraint network that is automatically extracted from the model. A constraint network is a set of variables and relations between them, described by constraints. Together, they define the admissible values for all variables. The constraint network is the equivalent to the flattened form of equations extracted by the Modelica compiler. Recall that there are no structural singularity conditions imposed on the extracted constraint network. The inference engine is able to operate with both insufficient information and redundant information. Inference strategies transform the constraint network into equivalent networks which describe the set of solutions in a more explicit way. Often they do not solve the problem directly, but they reduce the search space by problem reformulation. Transformations include reduction of variable domains and addition, removal, or modification of constraints. The reasoning process is explained in detail in Lunde 2006 [10].

3.2 The need for failure modes

The multiplier-adder circuit described in Section 2 has only used the correct behavior description of its constitutive components. The "*correct behavior*" models are usually easy to acquire; this kind of information should be available at the product design phase. As we have seen in the previous example, one could compute a list of candidates whose abnormal behavior might explain the faulty observed behavior. However, this list can be significantly reduced in size if additional information is available, in the form of models describing the most probable modes of faulty behavior. A diagnostic engine can use these additional behavior modes to check whether the assumption of an abnormal behavior mode explains the observed system behavior. The following example will illustrate why the specification of the faulty behavior (fault modes) is very helpful in order to achieve physically sensible diagnostic results.

Let us consider the simple electrical circuit from Figure 5 consisting of three electrical bulbs (`B1`, `B2`, `B3`) connected in parallel to a battery (`BAT`). The wire connection between the battery `BAT` and bulb `B1` is marked `w1` in Figure 5. The connection between `B1` and `B2` is marked `w2`, while a wire connection named `w3` connects bulb `B2` and bulb `B3`.

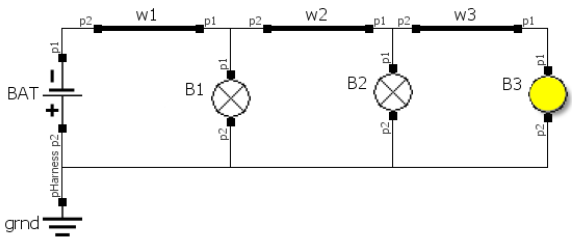

Figure 5. Simple electrical circuit consisting of three electrical bulbs connected in parallel, and illustrating the use of failure modes in diagnosis

Now let us consider the following symptom: the circuit is correctly powered by the battery, and we observe that only bulb B3 emits light, while B1 and B2 are off (are not emitting any light). If we admit the possibility of multiple faults in our circuit, we can conclude that a simultaneous failure in both B1 and B2 explains the behavior that we just observed. An alternative explanation consists in a simultaneous defect of wire w1 and wire w2. As long as we do not know anything about the nature of the defective behavior, it is thinkable that the defects in w1 and w2 will cause bulb B1 and B2 to be switched off, while they compensate each other causing B3 to behave normally, as observed. Actually, this is physical nonsense, since we know that an electrical wire can be either disconnected or have a short to ground, and there is no possible physical situation in which the failures in two different wires connected in series will compensate each other in that way and make bulb B3 emit light. Based on a similar formal reasoning, a diagnostic reasoning engine might erroneously consider the failure in the battery BAT and wire w2 as a potential candidate pair that might explain the observed behavior. To avoid those physically impossible candidates, additional information about how a component is most likely to fail should be integrated into any model intended for diagnosis purposes.

In Rodelica, it is possible to define several failure modes associated with each component. Let us consider a simple electrical component that has two pins:

```
model TwoPin
  Pin p1;
  Pin p2;
  FailureMode fm(max =1);
behavior
   // current balance that defines the
   // nominal behavior
   p1.i + p2.i = 0;
   // constraints for the failure mode
   // "disconnected"
   if (fm == 1){
      p1.i = 0;
   }
end TwoPin;
```

Compared to a Modelica representation of a TwoPin component, it can be noticed that the Rodelica TwoPin component, besides the nominal behavior, defines the behavior of the component when it is disconnected. The disconnected failure mode will have the effect that the current in pin1 will be zero (p1.i = 0). The alternative behavior of the component is specified with the help of a type variable FailureMode that acts like a switch between the two operation modes of the component. In our case, the failure mode behavior is enclosed between the brackets of the if(fm==1) statement.

A Resistor component that extends the TwoPin component can be defined as follows:

```
model Resistor extends TwoPin(fm (max = 2) );
   public     Resistance rNom(final min = 0);
   protected  Resistance rAct(final min = 0);
behavior
   // Basic constraints for nominal case:
   if (fm == 0){
      // Actual resistance is nominal resistance
      rAct = rNom;
      // Ohms law for voltage drop between pins
      p1.u - p2.u = rAct * p1.i;
    }

   // Extensions for failure mode "disconnected":
   if (fm == 1){
      // Infinite resistance between pin 1 and 2:
         rAct = INF_PLUS;
   }

   // Extensions for failure mode "short circuit
   // between pin 1 and 2":
   if (fm == 2) {
       // Same potential at pins 1 and 2:
       p1.u = p2.u;
       // No resistance between pin 1 and 2:
       rAct = 0.0;
    }
end Resistor;
```

It should be noticed that a Resistor component will inherit all constraints, and thus all failure modes, from the TwoPin component. By extending TwoPin, the resistor class has the possibility to add new constraints, thus extending the inherited failure modes or even adding new failure modes. By default, nominal behavior is assigned to the failure mode fm = 0. In the example, it is extended by specifying that the actual resistance will take the value of the nominal resistance, and by specifying Ohm's law for the voltage drop between pins. Note that constraints which are not enclosed in any if-statement (like Kirchhoff's law in the TwoPin class) are valid in all behavior modes. It should be also noticed that the Resistor has an extra failure mode that captures the situation when there is a short circuit between p1 and p2. In this case, p1 and p2 will have the same potential (p1.u = p2.u), and due to the short circuit the resistance of the Resistor will be equal to zero (rAct = 0). The short-circuit current is not specified within the resistor class.

Note that the number of constraints in each of the if-cases is not necessarily identical. This distinguishes Rodelica's if-statement from the if-statement in Modelica, where each branch is required to contain exactly the same number of equations, thus ensuring nonsingularity of the resulting system of equations. However, allowing different numbers of constraints is very useful in diagnosis. For some components, it may be appropriate to specify a generic failure mode which summarizes all kinds of faulty behavior which is too complex to describe in detail. For instance, imagine an electrical connector block with N pins. By the laws of combinatorics there is a large number of ways how those pins can be shorted, and it is unfeasible to provide the equations for each of those potential failure modes. An elegant way to avoid this complexity is the additional definition of a universal failure mode containing no constraint at all, which then may serve to explain any unexpected behavior which is not specified explicitly.

3.3 Interval data types

Another characteristic of the Rodelica language is the use of set-valued data types for defining model variables. This

is motivated mainly by the constraint solver, which does not always compute a single solution but rather constricts the values of all model values in an iterative way as far as possible without loosing any solution. In particular, most continuous model variables have the data type `Interval`. This is especially useful when working with data that is subject to measurement errors or uncertainties. For instance, a leak in a pipe with an uncertain size can be modeled by assigning the diameter a range of reasonable values, thus avoiding a potentially infinite number of failure modes.

Consequently, the inference engine uses interval arithmetics to propagate the values of the variables through the constraint network. The basic arithmetic operations for two intervals [a, b] and [c, d] are given below:

```
[a,b] + [c,d] = [a + c, b + d]
[a,b] - [c,d] = [a - d, b - c]
[a,b] × [c,d] = [min (ac, ad, bc, bd),
                 max (ac, ad, bc, bd)]
[a,b] / [c,d] = [min (a/c, a/d, b/c, b/d),
                 max (a/c, a/d, b/c, b/d)]
```

As an example the following simple Rodelica model

```
model testIntervalAddition
   Interval x = [1,6];
   Interval y = [3,7];
   Interval z;
behavior
   z = x + y;
end testIntervalAddition;
```

will restrict the possible values of the variable `z` in the interval [4 13].

Using interval variable types will have certain unexpected effects. For instance, consider the piecewise defined function given below in Figure 6:

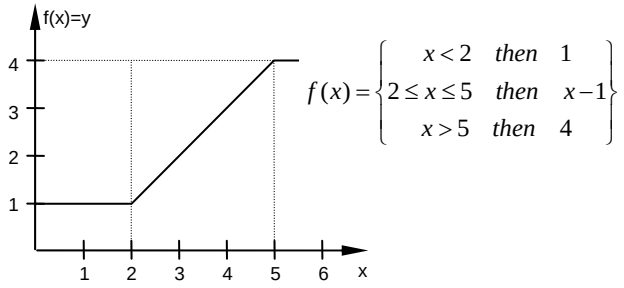

Figure 6. Piecewise defined example function

In Modelica, this function can be easily represented by the following model:

```
model PiecewiseRealFunction
   Real x;
   Real y;
equation
   y = if     (x < 2) then 1;
       elseif (x >= 2 & x <= 5) then x - 1;
       else   (x > 5) then 4;
end PiecewiseRealFunction;
```

Now, let us consider that the variables `x` and `y` are of type `Interval` and the initial value of `x` is the interval [0 10]. In this case, the conditions `x<2`, `x>=2 & x<=5` and `x>5` are undecidable – for some values in x, they are true, but for others they are false. As a consequence, none of these constraints can be evaluated and `y` remains undetermined, although it is actually clear that the resulting value range should be `y=[1 4]`. Modeling with conditional constraints in a conventional style when interval type variables are involved in the condition can lead to a loss of information. For this reason, the "`or`" clause was introduced as an alternative to the `if`-statement. In Rodelica, the piecewise real function from the example above would be more appropriately formulated as follows:

```
model PiecewiseRealFunction
   Interval x(min = 0, max = 10);
   Interval y;
behavior
   or { (x < 2; y = 1;}
        {x = [2 5]; y = x - 1;}
        {x > 5; y = 4;}
      }
end PiecewiseRealFunction;
```

An `or`-clause is treated as a single very complex constraint, whose evaluation is a two-step process: firstly, the branches of the `or` clause are evaluated separately; in a second step, the final result for each variable is calculated as the set union of the value ranges from all branches. In case that a conflict in one of the branches is detected (which means that there is no solution for at least one variable involved) the branch is excluded from the merging. As an example, assume that the variable `x` can take values between 1.5 and 4 (`x=[1.5 4]`) and y can take any real value (`y=[- +]`). By propagating the interval [1.5 4] for `x`, the three `or` branches of the piecewise real function previously defined will result in the following:

```
x = [1.5 2]; y = 1;
x = [2 4]; y = [2 4] - 1 = [1 3];
x = {}; y = 4;
```

The third branch will be excluded because it results in a conflict, which is easily detectable by the empty set assigned to x. It is the result of the set intersection of the initial value of x (`x = [1.5 4]`) and the solution of the constraint `x > 5` which is `x = [5 +]`. The unification of the solutions from the other two branches results in the overall solution `x = [1.5 4]` and `y = [1 3]`.

4. Industrial Example

Let us consider as an example a front-light power window system of a Volvo V70 car. The model was built and analyzed by means of RODON, and it is formulated using the Rodelica language. The model of the front light system contains an Electronic Control Unit (ECU), the front lights and the associated electrical harness consisting of electrical wires, fuses and connector blocks. The ECU is able to set and detect diagnostic fault codes. The activation of a diagnostic fault code is an indication that something is wrong in the system. A small section of the front light system is depicted in Figure 7.

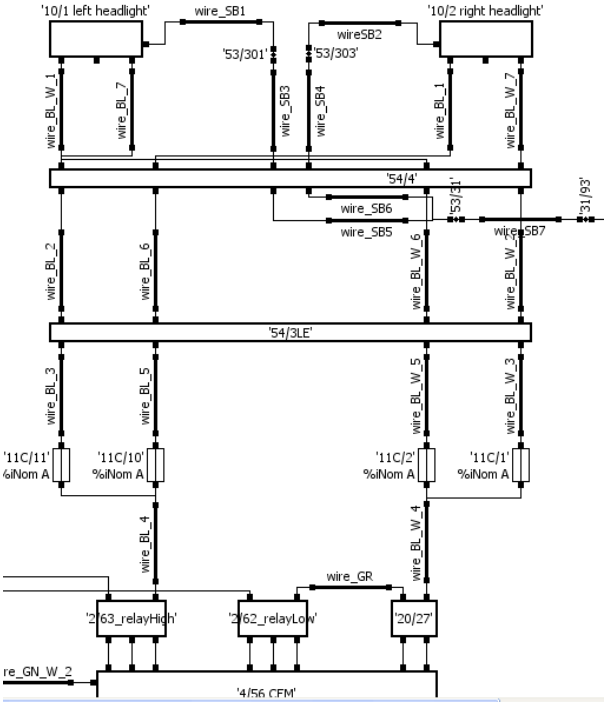

Figure 7. The front-light subsystem model of a Volvo V70 car developed in RODON.

The front light system is powered by the car battery which also powers the power-window system of the car. The power window subsystem contains an ECU of its own as well as two power window motors with hall sensors, fuses, and connector blocks and wires. The ECU detects and sets diagnostic trouble codes for power failure and Hall sensor failure. The Hall sensors are used to precisely locate the position of the window.

A small part of the power window model is shown in Figure 8.

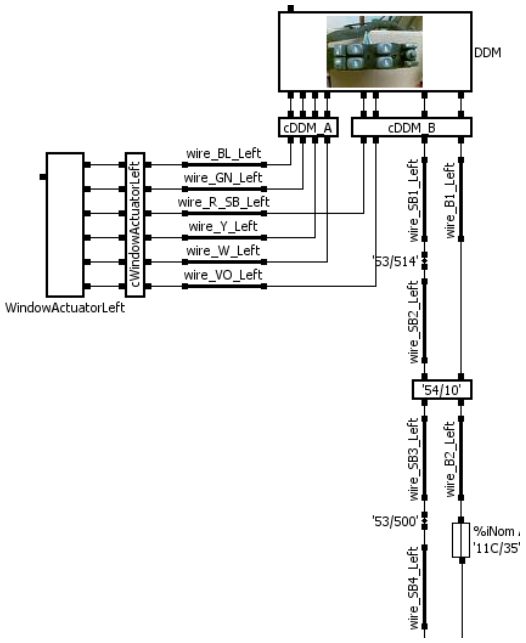

Figure 8. A part of the power-window subsystem model of a Volvo V70 car developed in RODON

The implementation details of the front-light and power-window subsystem models depicted in Figure 7 and Figure 8 are not relevant for the discussion in the paper. Let us just mention that for each component, the nominal behavior was modeled and augmented with the relevant failure modes, and that variables whose values can be measured in the real system have been marked as observable in the model.

Once the model has been created, RODON supports several diagnostic methods:

- Model-Based Diagnosis (MBD), including interactive MBD which means that additional measurements can be provided by the user to narrow down the number of diagnostic candidates.
- The automatic generation of decision trees (or diagnostic trouble-shooting trees), which can serve as a model documentation or to assist the mechanic in a workshop in a guided diagnosis.
- The automatic generation of diagnostic rules for on-board diagnostics.

In the following, two of these approaches are illustrated using the model described above.

4.1 Interactive Model-Based Diagnosis

Model-based diagnosis is the most powerful, but also the most resource-consuming diagnostic approach. By using both nominal and faulty behavior, as specified by the Rodelica model, it is able to detect single or multiple faults, or to propose additional measurements in an interactive way. Available observations and measurements can be fed to the model in several ways: there are a file interface, a GUI, and a CAN-bus interface for direct communication with the car. The main principles of MBD were described in Section 2.

As an example, we consider the situation in which the user pushes the power window button with the intent to slide down the window pane, but the pane does not move. Obviously, there is a failure in the system that immobilizes the window pane. After entering this symptom into the tool, we can start the model-based diagnostic process to find an explanation for the observed behavior, and to isolate the component that caused that particular behavior. In the first step, the diagnostic engine will compute a list of candidates (hypotheses) that explain the observed behavior:

```
PowerWindowSystem.'11C/35' disconnected,
PowerWindowSystem.'54/10' disconnected,
PowerWindowSytem.DDM.powerWindowSwitchFrontLeft.
                  switchWindowFront disconnected,
PowerWindowSystem.WindowActuatorLeft.
                  WindowUpDown disconnected,
PowerWindowSystem.cDDM_B disconnected,
PowerWindowSystem.cWindowActuatorLeft
                  disconnected,
PowerWindowSystem.wire_B1_Left disconnected,
PowerWindowSystem.wire_B2_Left disconnected,
PowerWindowSystem.wire_R_SB_Left disconnected,
PowerWindowSystem.wire_SB1_Left disconnected,
PowerWindowSystem.wire_SB2_Left disconnected,
PowerWindowSystem.wire_SB3_Left disconnected,
PowerWindowSystem.wire_SB4_Left disconnected,
```

```
PowerWindowSystem.wire_VO_Left disconnected
```

So far, 14 candidates have been identified where each corresponds to a single fault which can fully explain the symptom. The list is ordered by the associated confidence values. These confidence values are part of the model and can be imagined as “rough order of magnitude” reliability figures. Components with a lower confidence value are listed first because they are less reliable than others. In the absence of confidence values, the tool will sort the candidates by secondary criteria, for instance lexically. In the graphical user interface, the candidates are highlighted using color shades ranging from red to blue, with red representing lower confidence value and blue representing less probable candidates. The highlighting of candidates in the GUI is depicted in Figure 9.

Dealing with such a big number of candidates is not very efficient in a workshop environment where the mechanic needs to isolate the failure in a very short period of time. There is a need to narrow further down the number of candidates. This can be done by providing extra information to the tool in the form of measurements.

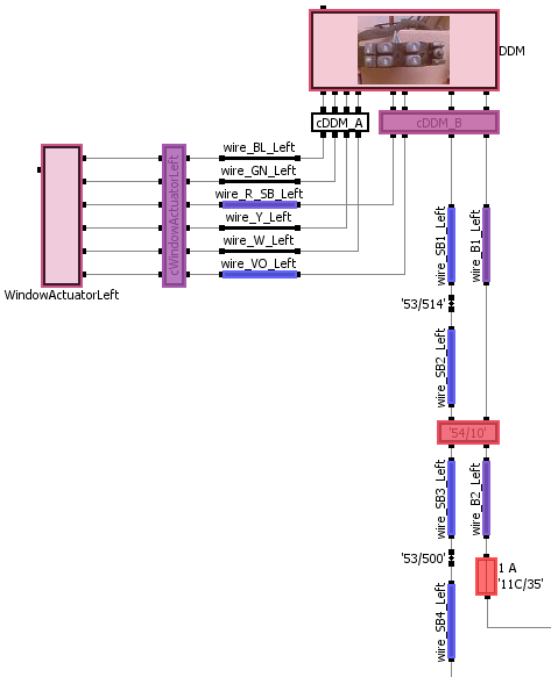

Figure 9. Diagnostics candidates are highlighted in the model browser

The inference engine can profit from this new information to validate the previously computed candidates, and possibly retract those that do not match the measured values. In Figure 10, in the upper part of the window, the list of candidates is presented, whereas the lower part shows a list of potentially useful measurements or observations to be performed on the system. The latter are ordered by the estimated impact they will have in reducing the number of candidates. The first measurement in the list has the biggest potential to reduce the number of candidates.

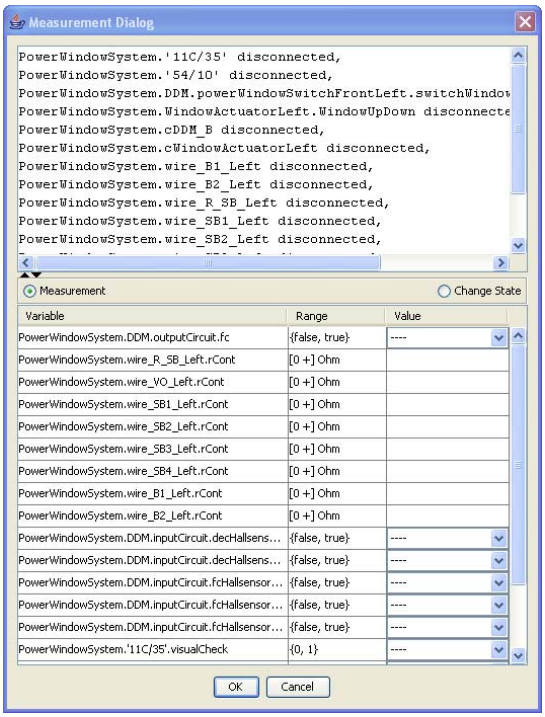

Figure 10. Lists of diagnostic candidates (hypotheses) and of proposed measurements

However, the mechanic is free to choose any measurement from the list. In practice, there might be other selection criteria which are unknown to the tool. For instance, checking a fault code activation on the dash board is less expensive than a voltage measurement on a connector block, which involves dismounting the door to have access to the electrical harness. In the present context, we choose the first proposal in the list. It is a fault code reading which can be automatically read from the car by the off-board diagnosis device. If the car dashboard is activated, the fault code may be read off the dashboard, too. The status of the fault code is given to the tool by means of the measurement GUI. It leads to the assignment of the corresponding value (true or false) to the variable `outputCircuit.fc` which is part of the subsystem `PowerWindowSystem.DDM`.. We assume that the fault code is active. This additional information is used by the reasoning engine to exclude some of the candidates from the list. In the described situation, there are only 4 candidates left:

```
PowerWindowSystem.WindowActuatorLeft.
                  WindowUpDown disconnected,
PowerWindowSystem.cWindowActuatorLeft
                  disconnected,
PowerWindowSystem.wire_R_SB_Left disconnected,
PowerWindowSystem.wire_VO_Left disconnected,
```

The diagnostic process can be continued by entering further measurements from the proposed list until the final diagnosis is produced (only one single-fault candidate is left). We call this process, in which the user is requested to provide additional measurements to progressively refine the diagnosis, *interactive model-based diagnosis* (IMBD).

4.2 Generation of Decision Trees

In environments where fewer resources are available, a more compact form of diagnostic knowledge representation is desirable. RODON is able to derive several forms of compiled diagnostic knowledge from the object-oriented Rodelica model, automatically, by means of a systematic simulation of all essential system states. To this end, the modeler has to specify which single faults and which operational states of the system are relevant for the analysis. The Cartesian product of all those operational states with the set of fault states (plus the state *System ok*) defines a so-called state space. An automatic simulation control module can then be used to simulate each state in the state space, systematically, and to write the results into a data base, which we call *state data base* (SDB). The SDB can be used for risk analyses, like failure-modes and effects analysis (FMEA), and it provides the necessary information for generating decision trees and diagnostic rules.

Decision trees are used to determine which system state explains a symptom, with minimal effort and costs. The root node of a decision tree is the symptom. Leaf nodes are result nodes describing a fault state, e.g. "*w1 is disconnected*". The intermediate nodes are decision nodes which help to discriminate the system state. Decisions may involve a measurement or a visual check to be done by the mechanic. To perform a diagnosis for a selected symptom, the decision tree is traversed starting from the root node, finally arriving at the leaf node with the correct diagnosis. The path through the tree to the diagnosis depends on the answers given at each passed decision node.

The generation process is configurable in a very high degree. In particular, actions required at the decision nodes may be more or less expensive. Consequently, the decision nodes in the generated tree are ordered with respect to a cost measure defined by the modeler. For instance, if fault code checks are declared to be cheap in comparison to actual measurements, then the mechanic will be asked by the resulting trees to check all helpful fault codes before encountering a decision node containing a measurement.

Figure 11 displays a decision tree whose root symptom is an active fault code at one of the input circuits of the Hall sensors in the power window subsystem. If this fault code is activated, the user (the mechanic) is instructed to check another fault code PDN-0020. In case that this fault code is active as well, the user is further asked to perform a visual check on `cPDM_A`, which is a switch in the power window system. Otherwise, the user is instructed to make a continuity test (Ohmic measurement) on one of the wires. Similarly to the interactive model-based diagnosis, the user is asked to perform a certain measurement or to make an observation. Based on the result of the user action, a certain branch of the failure tree is followed until the component that caused the failure is isolated.

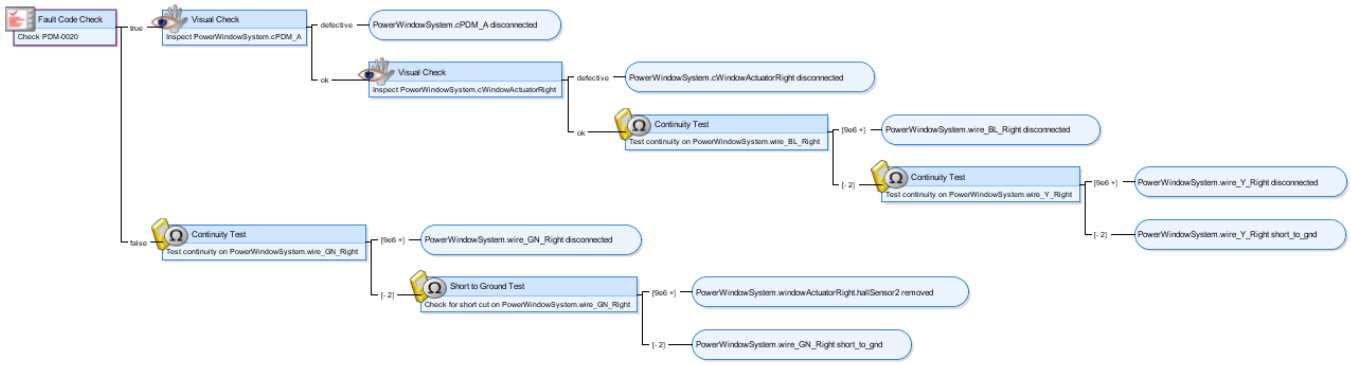

Figure 11. Generated Fault Tree

Traditionally, the generation of decision trees is done manually by the system experts, which is an extremely time consuming and error-prone task. Model-based generation of decision trees provides a systematic and safer way to analyze the combinations of all relevant operational states and component failures that can occur in a system, thus serving as a valuable tool in the authoring of troubleshooting documentation.

5. Summary and Conclusions

In this paper, we have presented Rodelica, an equation-based object oriented language derived from Modelica and adapted to model-based diagnosis purposes. Some of the characteristics of the language that makes it suitable for diagnosis are:

- Numerical representation of values in general as value sets (intervals, sets of discrete or Boolean values); qualitative representation is possible as well. This allows to cope with sensor and manufacturing tolerances as well as with insufficient information in case of faulty behavior. Both situations are common in diagnostic applications.
- Relations between variables are formulated as constraints. Supported constraint types include equations, but also inequalities, conditional constraints (if and or clauses), Boolean relations (formulas or truth tables), and spline interpolation.
- The number of constraints in a model is not restricted by the number of variables or by any notion of regularity. A model may be under- or over-determined. Underdetermined models lead to large value ranges

as the result of the inference process, over-determined models lead to conflicts which can be used as a starting point for diagnosis.

- In particular, it is possible to define failure modes for each class, in addition to modeling the nominal behavior. Any number of failure modes can be defined per class or component.
- The solver provides the appropriate computational methods based on constraint propagation and interval set arithmetic.

The benefits of using the Rodelica language have been illustrated in many industrial-size projects. Like the Modelica language, which is considered to be the "de facto" standard for modeling and simulation of hybrid systems, we believe that Rodelica can be proposed to constitute the standard for the exchange of diagnostic models.

Acknowledgements

We would like thank to the development team of RODON at Sörman Information AB Sweden for valuable discussions and the feedback received for this paper.

References

[1] Klaus-Dieter Althof, Eric Auriol, Ralph Barleta and Michel Manago (1995). *A Review of Industrial Case-Based Reasoning Tools.* AI Intelligence, Oxford, UK

[2] Modelica Association (2007). *Modelica - a Unified Object-Oriented Language for Physical Systems Modeling - Language Specification Version 3.0.* September 2007

[3] Paul Inigo Barton and C.C. Pantelides (1993). *Gproms - a Combined Discrete/Continuous Modelling Environment for Chemical Processing Systems.* The Society for Computer Simulation, Simulation Series, vol: 25, issue: 3. pg 25-34, 1993

[4] David Broman, Kaj Nyström and Peter Fritzson (2006). *Determining over- and under-Constrained Systems of Equations Using Structural Constraint Delta*. In Proceedings of Fifth International Conference on Generative Programming and Component Engineering (GPCE'06), Portland, Oregon, USA, 2006

[5] Peter Bunus and Peter Fritzson (2004). *Automated Static Analysis of Equation-Based Components.* Simulation: Transactions of the Society for Modeling and Simulation International. Special Issue on Component Based Modeling and Simulation., vol: 80, issue: 8. pg 2004

[6] E. Christen and K. Bakalar (1999). *Vhdl-Ams-a Hardware Description Language for Analog and Mixed-Signal Applications.* IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol: 46, issue: 10. pg 1263-1272, 1999

[7] Johan de Kleer and James Kurien (2003). *Fundamentals of Model-Based Diagnosis*. In Proceedings of IFAC SafeProcess, Washington USA, June 2003

[8] Karin Lunde (2000). *Object-Oriented Modeling in Model-Based Diagnosis*. In Proceedings of Modelica Workshop, Lund, Sweden, Oct 23-24 2000

[9] Karin Lunde, Rüdiger Lunde and Burkhard Münker (2006). *Model-Based Failure Analysis with Rodon*. In Proceedings of ECAI 2006 - 17th European Conference on Artificial Intelligence Riva del Garda, Italy, August 29 -- September 1 2006

[10] Rüdiger Lunde (2006). *Towards Model-Based Engineering: A Constraint-Based Approach.* Shaker, Aachen

[11] Jakob Mauss, Volker May and Mugur Tatar (2000). *Towards Model-Based Engineering: Failure Analysis with Mds*. In Proceedings of ECAI-2000 Workshop on Knowledge-Based Systems for Model-Based Engineering, Berlin, Germany, 2000

[12] Hans Olsson, Martin Otter, Sven Erik Mattsson and Hilding Elmqvist (2008). *Balanced Models in Modelica 3.0 for Increased Model Quality*. In Proceedings of 6th International Modelica Conference, University of Applied Sciences, Bielefeld, Germany, March 3rd-4th 2008

[13] Sankar K. Pal and Simon C. K. Shiu (2004). *Foundation of Soft Case Based Reasoning*. John Wiley & Sons, Inc., Hoboken, New Jersey.

[14] P.C. Piela, T.G. Epperly, K.M. Westerberg and A.W. Westerberg (1991). *Ascend: An Object-Oriented Computer Environment for Modeling and Analysis: The Modeling Language*. Computers and Chemical Engineering, vol: 15, issue: 1. pg 53-72, 1991

[15] Martin Sachenbacher, Peter Struss and Claes M. Carlén (2000). *A Prototype for Model-Based on Board Diagnosis of Automotive Systems*. AI Communications, vol: 13, issue: 2. pg 83 - 97, 2000