An Analyzer for Declarative Equation Based Models

JIAN-WAN Ding[1] LI-PING Chen[1] FAN-LI Zhou[1] YI-ZHONG Wu[1] GUO-BIAO Wang[2]
[1] National CAD Support Software Engineering Research Center, Huazhong University of Science and Technology, Wuhan 430074, China
[2] National Natural Science Foundation of China, Beijing 100085, China

Abstract

Along with its benefits, object-oriented modeling with Modelica language also brings the risk of missing or redundant equations, thus leads to an under-constrained problem or an over-constrained problem. This paper aims at facilitating debugging of structurally singular model through an analyzer in terms of reducing the number of tests to correct the model. When checking for singularities of a component, the analyzer first uses some fictitious equations to replace the constraint equations generated by the outside connections of the component, and then identifies whether the singularities derive from the component or not by structural analysis. The checking procedure is done recursively. The proposed analyzer can automatically identify faulty components that are responsible for the singularities.

Keywords: Modelica; debugging; structural analysis; structural singularity

1 Introduction

The need for mathematical modeling and simulation in engineering is continuously rising since technical systems become increasingly complex and physical prototyping becomes too expensive. Modelica is an object-oriented equation based language for efficient modeling and simulation of complex, heterogeneous, multi-domain physical systems described by ordinary differential and algebraic equations. Modelica allows describing simulation models in a declarative object-oriented manner, so that a model can be used to solve various problems and model components easily can be modified to describe similar systems. Following object-oriented modeling methodology, a Modelica model is structured as closely as possible to the corresponding physical system. In particular, models are defined in acausal form, independently of the context in which it is used. Complex models can be realized by aggregating more submodels and their connections within a composite larger model, which may in turn be connected to other models. For continuous time modeling, by assembling the declarative equations of component models, this form of model representation gives rise to a larger scale system of differential algebraic equations (DAEs).

Along with its benefits, object-oriented modeling with Modelica language often unconsciously leads to an under-constrained problem or an over-constrained problem. In such situations, numerical solvers fail to find correct solutions to the underlying system of equations. Due to the high-level abstraction of equation based models, it is extremely hard to find and localize model singularities. Moreover, especially for complex physical systems, the underlying system of equations is of very large dimension. Even if a modeler suspects a redundant equation, it is very time-consuming to identify the equation in a large scale system of equations. Therefore, it is a key problem to check for the singularity of the model before using it for analysis or design purposes, or before generating the simulation code.

To address the aforementioned problem, Bunus and Fritzson[1,2] have developed a debugging framework for Modelica models and have adapted traditional debugging techniques and algorithms to it. Equations and variables that probably cause the irregularities are isolated by applying graph decomposition techniques and reduced by using structural information and semantic information. The developed algorithms and methods help to statically detect and repair a broad range of errors without having to execute the simulation model.

This paper focuses on checking for structural singularities of equation based models. Our goal is to develop an analyzer, called *Model Singularity Analyzer* (MSA), to help the modeler to localize model singularities more quickly and more efficiently.

The paper is organized as follows: Section 2 discusses the three main steps of the MSA. The examples are demonstrated in Section 3. Section 4 gives a comparison of the MSA to existing methods. Section 5 presents our conclusions.

2 Structural analysis

2.1 Detecting structural singularities

Before solving a Modelica model, the Modelica source code is first translated into a so called "flat model", which is a flat set of equations, constants, parameters and variables. For continuous time modeling, the flat model can be described by a system of differential algebraic equations of the form

$$\boldsymbol{F}(\boldsymbol{x}, \dot{\boldsymbol{x}}, \boldsymbol{u}, \boldsymbol{p}, t) = 0 \qquad (1)$$

where $\boldsymbol{x}$ is the variable vector, $\boldsymbol{u}$ is the input vector, $\boldsymbol{p}$ is the parameter vector and t is time.

The system of equations resulting from a Modelica model can naturally be represented as a bipartite graph $G = \{V_1, V_2, E\}$ where V_1 denotes the set of equations, V_2 denotes the set of variables, and there is an edge $e \in E$ between a variable $v \in V_2$ and an equation $u \in V_1$ if variable v appears in equation u.

When using a model for simulation, it is a basic requirement that there are as many equations as variables, and that we can determine a mapping between equations and variables in such a way that each equation is related one and only one variable. If the requirement is satisfied, we call the model structural nonsingularity. Otherwise, the mode is structurally singular.

Therefore, a crucial step of checking for structural singularities is to assign each variable v_i to a unique equation e_j such that v_i appears in e_j. If it is impossible to pair variables and equations in this way then the model is structurally singular. This assignment procedure can be realized by calculating a perfect matching in the bipartite graph associated with the system of equations. If the perfect matching does not exist then the corresponding system of equations is structurally singular.

For a structurally singular model, we can isolate the over-determined and under-determined subsets of equations by the DM decomposition[4]. However, when applied to a large system of equations, the DM decomposition finds a large under-determined or over-determined block, which is less helpful for localizing the structural singularities, so methods to help are necessary.

To check for structural singularities of DAE systems, we should not distinguish between the appearance of x and the appearances of its derivative, the appearances of $\dot{x}$ are considered as appearances of x. Thus to check if the DAE system $F(x, \dot{x}, t) = 0$ is structurally singular, we check if the algebraic system $F(x, x, t) = 0$ is structurally singular with respect to x.

2.2 Generating fictitious equations

A complete Modelica model is always made up of components, which maybe consist of other sub-components. For a composite model, the structural singularities may be caused by improper use of components, or come from structurally singular components. So if a composite model is structurally singular, we should check for structural singularities of its components to determine whether the singularities comes from its components or not.

However, a component of a model always has outside connections to communicate with the rest of the model. If we isolate a component from the environment where it is used, and check for structural singularities of the component, we can not determine which connection equations generated by outside connections should be included into the system of equations resulting from the component. The reason is connections between components are often acausal, i.e., the data flow in connection is not explicitly specified.

The object-oriented modeling methodology uses energy flow for modeling[5,6]. It is necessary for the modeler, when designing model interfaces, to guarantee that connecting such models in an arbitrary fashion will always ensure that power, momentum, and mass are balanced at the interfaces. This means that the sum of incoming energy flows must equal the sum of outgoing energy flows at connection points. Hence, acausal connections in Modelica models in fact are a kind of connection based on energy flow.

Therefore, when checking for structural singularities of components, we disregard the connection equations generated by outside connections, and use some fictitious equations to compensate the lost constraints in the following way.

1. We generate a fictitious equation for each flow variable and make the equation contain all variables appearing in the same connector with the flow variable.

2. We generate for each input variable a fictitious equation which assigns a value to the corresponding input variable.

3. Potential variables may be more than flow variables in a connector. In such case, we assume some outside constraints on these redundant potential variables. To construct the most general constraint, we generate a fictitious equation for each redundant po-

tential variable, and make these equations contain all potential variables of all the connectors. Let C denote a model component, k be the number of the fictitious equations added for the redundant potential variables, r be the number of connectors of C, m_i be the number of potential variables in the ith connector, n_i be the number of flow variables in the ith connector, p be the number of variables of C, and q be the number of equations of C. There is the following dependence:

$k = 0$, if $p - (q + \sum_{i=1}^{r} n_i) < 0$;

$k = p - (q + \sum_{i=1}^{r} n_i)$, if

$0 \le p - (q + \sum_{i=1}^{r} n_i) \le \sum_{i=1}^{r} (m_i - n_i)$;

$k = \sum_{i=1}^{r} (m_i - n_i)$, if $p - (q + \sum_{i=1}^{r} n_i) \ge \sum_{i=1}^{r} (m_i - n_i)$.

Where $\sum_{i=1}^{r} n_i$ is the number of fictitious equations added for flow variables.

The aim of case 1 is to construct a general energy flow constraint equation. To explain the idea, we first define a generic physical connector class as follows:

connector generic
  Real *e*; //potential variable
  flow Real *f*; //flow variable
end generic;

We assume *c*1 and *c*2 are two instances of the connector class generic. According to Modelica semantics, the connection connect(*c*1, *c*2) produces two equations, namely: *c*1.*e*=*c*2.*e* and *c*1.*f*+*c*2.*f*=0. From this two connection equations, we can get *c*1.*e***c*1.*f*+ *c*2.*e***c*2.*f* =0. We further assume *c*1.*e***c*1.*f*=*C*, where *C* is a constant, then *c*2.*e***c*2.*f*=–*C*. Since we only focus on which variables appear in each equation rather than how they appear when checking for structural singularities, we can use the general form equations *f*1(*c*1.*e*, *c*1.*f*)=0 to express *c*1.*e***c*1.f=*C*, and *f*2(*c*2.*e*,*c*2.*f*)=0 to express *c*2.*e***c*2.*f*=–*C*. However, there are sometimes more than one matched flow variable and potential variable in a connector. Hence, without loss of generality, we generate for each flow variable a general form equation which contains all flow and potential variables of a connector, i.e., *g*(*e*1,*e*2,…,*f*1,*f*2,…)=0, where *f*i(i=1,2,…) is flow variable, *e*i(i=1,2,…) is potential variable.

The fictitious equations generated for flow variables can be regarded as constraint equations about power, momentum, and mass which passes connectors. This is because that, in any physical system, the power can be written as the product of two adjugate variables, called the potential and the flow in Modelica language. For example, in electrical systems the power *P*=*v***i* where *v* denotes the voltage and *i* denotes the current, and in translational mechanical systems the momentum *P*=*v***f* where *v* denotes the velocity and *f* denotes the force.

By generating fictitious equations, we can obtain the system of equations resulting from a component. By checking for structural singularities of the system of equations, we can determine that the component is structurally singular or not.

2.3 Locating faulty components

A connector can be connected, and also be not connected. When checking for singularities of a component, two possibilities are considered. It is noticed that, once a connector is assumed to be not connected, all other connectors of the same component are considered to be connected. If a connector is assumed to be not connected, the fictitious equations for flow variables of the connector are not generated, instead flow variables are set to zero.

If a structurally singular component is make up of subcomponents, the checking procedure can be done recursively, until the fault component is a primitive component, or each subcomponent of the fault component is structurally nonsingular.

For that, we introduce the following definition.

Definition 1. Let *C* be a structurally singular component. *C* is a minimal structurally singular (MSS) component if either of the following two conditions is satisfied:

1. *C* is a primitive component described in terms of equations;

2. *C* is a composite component consisting of other connected subcomponents, and none of its subcomponents is structurally singular.

We therefore propose the following strategy for locating structural singularities in Modelica models. Our aim is to locate all the MSS components of a structurally singular model. First, we check whether a perfect matching exists or not in the bipartite graph associated with a whole model. This can be done by solving the maximum matching problem[7,8]. If a perfect matching does not exist, we apply the DM decomposition to isolate the over-determined and under-determined subsets of equations. Then we check

for structural singularities of each component in turn to determine whether the singularities derive from the component or not. So it is very natural to consider the following subproblem procedure.

Procedure *P*(*C*)

Input: a structurally singular component *C*

Output: the structurally singular subcomponent set *T*

```
begin
  set T := ∅;
  let Q be a subcomponent set; set Q := ∅;
  add the subcomponents of C to Q;
  for each C′ ∈ Q do
    begin
      generate fictitious equations for C′;
      obtain the system of equations E of C′;
      construct the bipartite graph G for E;
      determine a maximum matching W in G;
      if W is not perfect then
        add C′ to T;
    end
end
```

By performing the above subproblem procedure, we can obtain all the singular subcomponents of a singular component, if they exist. For each singular subcomponent, we further perform the above procedure to obtain its singular subcomponents. This procedure is performed iteratively until the singular subcomponent is a MSS component. Clearly, the recursive application of the above procedure can construct a tree of subproblems. By using the depth first rule, we obtain our algorithm which outputs all the MSS components of a structurally singular model. The algorithm is described as follows:

Algorithm 1: Obtaining the MSS components

Input: a structurally singular component *M*

Output: the MSS components set *S*

```
begin
  set S := ∅;
  if M is composite then
    begin
      let L be the list of components;
      set L := P(M);
      while L is not empty do
        begin
          let C be the last component in L;
          remove C from L;
          if C is composite then
            begin
              let K be a component set;
              set K := P(C);
              if K = ∅ then
                add C to S;
              else
                add each element of K to the end
                of L;
            end
          else
            add C to S;
        end
    end
  else
    add M to S;
end
```

Finally, for each MSS component it is desirable to give the user some hints what is wrong. If a MSS component is a composite component, we inform the user that some subcomponents of the MSS component may be improperly used. If a MSS component is a primitive component, we produce some hints by locating critical parts of the component that are responsible for singularities. For an over-constrained problem, the redundant equations must appear in both the over-determined subset and the MSS component. Similarly, for an under-constrained problem, the free variables must appear in both the under-determined subset and the MSS component.

3 Examples

The first example is an oscillator model depicted in figure 1. A mass Ma is hanging in a spring Sa which is connected to a fixed housing Fa. The mass Ma is subject to the gravitational force and the force from the spring. It is given an initial position $s = -0.5$, which is offset from the equilibrium position and therefore starts an oscillating movement up-and-down. The positive coordinate direction is upward in the figure, which applies to both positions and forces.

The Modelica description of the oscillator is presented as follows:

```
model Oscillator
  Mass Ma(L=1, s(start=-0.5));
  Spring Sa(s_rel0=2, c=10000);
  Fixed Fa(s0=1.0);
equation
```

connect(Sa.flange_b, Fa.flange_b);
connect(Ma.flange_b, Sa.flange_a);
end Oscillator;

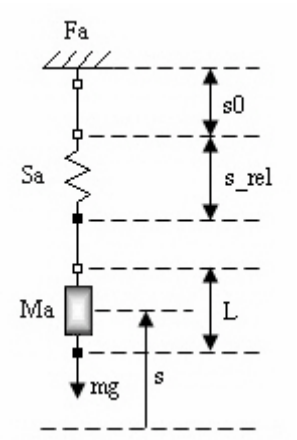

Figure 1. The oscillator Model

The component model Spring and Fixed are available in Modelica class libraries. The definition of the component model Mass is presented as follows:

model Mass
extends Rigid;
parameter Real m =1;
constant Real g =9.81;
Real v;
Real a;
equation
v = der(s);
a = der(v);
flange_b.f = m*a – m*g;
v = 6; //an additional equation
end Mass;

In order to obtain an over-constrained problem, we introduce an additional equation (v=6) in the model Mass. The set of equations generated from the Oscillator model is presented in table 1.

Table 1. The set of equations and variables corresponding to the Oscillator model

e1: Ma.v = der(Ma.s)	v1: Ma.s
e2: Ma.a = der(Ma.v)	v2: Ma.v
e3: Ma.flange_b.f=Ma.m*Ma.a	v3: Ma.a
-Ma.m *Ma.g	v4: Ma.flange_a.s
e4: Ma.v=6	v5: Ma.flange_a.f
e5: Ma.flange_a.s=Ma.s-Ma.L/2	v6: Ma.flange_b.s
e6: Ma.flange_b.s=Ma.s+Ma.L/2	v7: Ma.flange_b.f
e7: Sa.f=Sa.c*(Sa.s_rel-Sa.s_rel0)	v8: Sa.s_rel
e8: Sa.s_rel=Sa.flange_b.s	v9: Sa.f
-Sa.flange_a.s	v10: Sa.flange_a.s
e9: Sa.flange_a.f=-Sa.f	v11: Sa.flange_a.f
e10: Sa.flange_b.f=Sa.f	v12: Sa.flange_b.s
e11: Fa.flange_b.s=Fa.s0	v13: Sa.flange_b.f
e12: Ma.flange_b.s=Sa.flange_a.s	v14: Fa.flange_b.s
e13: Ma.flange_b.f+Sa.flange_a.f=0	v15: Fa.flange_b.f
e14: Fa.flange_b.s=Sa.flange_b.s	
e15: Fa.flange_b.f+Sa.flange_b.f=0	
e16: Ma.flange_a.f=0	

Performing the DM decomposition, the over-constrained subgraph is found and represented graphically in figure 2, where the covered edges by the maximum matching are marked by thick lines.

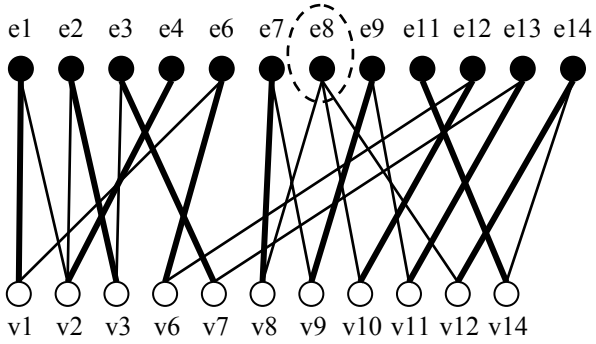

Figure 2. The over-constrained subgraph

When check for structural singularities of the component Ma, we first assume that both the connectors flange_a and flange_b are connected, and generate e1': f(Ma.flange_a.f, Ma.flange_a.s)=0 for the flow variable Ma.flange_a.f and e2': f(Ma.flange_b.f, Ma.flange_b.s)=0 for the flow variable Ma.flange_b.f. The corresponding bipartite graph to the component Ma is shown in figure 3, where a maximum matching is marked by thick lines.

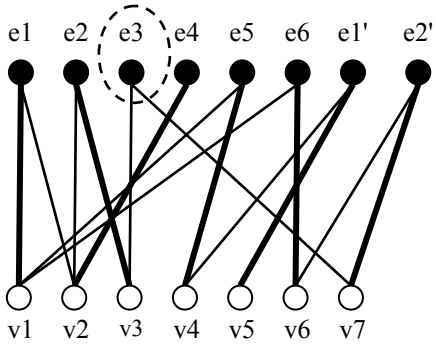

Figure 3. The bipartite graph corresponding to the component Ma with a maximum matching

In figure 3, e3 is a free vertex, so the component Ma is structurally singular, and there exists one redundant equation. It means the primitive component Ma is a MSS component. The equations that appear in both the over-constrained subgraph and the component Ma are e1, e2, e3, e4 and e6, one of which is redundant. Similarly, by generating fictitious, we can determine the components Sa and Fa are structurally

nonsingular. In this case, the following message is presented to the modeler.

```
Error: The model Oscillator is structurally singular.
The singularity comes from the component Ma.
There is 1 redundant equation in the equations:
  v = der(s);
  a = der(v);
  flange_b.f = m*a- m*g;
  v = 6;
  flange_b.s = s+L/2;
```

Figure 4. The error message for the model Oscillator

The second example is an AC motor model depicted in figure 5. This model contains components from the two domains: mechanical domain and electrical domain.

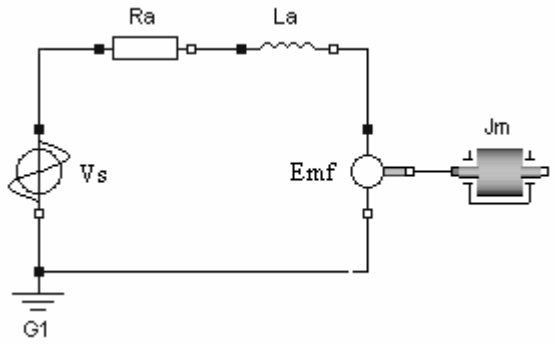

Figure 5. The AC motor model

The Modelica description of the ACMotor model appears as follows:

```
model ACMotor
  SineVoltage Vs(V=220,freqHz=50);
  Resistor Ra(R=0.5);
  Inductor La(L=0.1);
  EMF Emf;
  Inertia Jm(J=0.001);
  Ground G1;
equation
  connect(Vs.p, Ra.p);
  connect(Ra.n, La.p);
  connect(La.n, Emf.p);
  connect(Emf.flange_b, Jm.flange_a);
  connect(Emf.n, G1.p);
  connect(Vs.n, G1.p);
end ACMotor;
```

The component models Inductor, EMF, Inertia and Ground are available in Modelica class libraries. In order to make the ACMotor singular, the following Resistor model is defined.

```
model Resistor
  extends OnePort;
  parameter Real R=1;
  Real s;
equation
  R*i = v+s;
  p.v=12;
end Resistor;
```

The complete set of equations (shown in Table 2) generated from the ACMotor class consists of 37 differential algebraic equations and 37 variables. This is a structurally singular problem where under-constrained and over-constrained situations appear simultaneously. The DM decomposition will find the over-constrained, well-constrained and under constrained subgraphs. The over-constrained subgraph contains equations e1, e4, e9, e20, e29, e30 and e37, and variables v1, v3, v5, v7, v25 and v29. The well-constrained subgraph contains equation e33 and variable v34. All other equations and variables are contained in the under-constrained subgraph. Because of space limitation, the over-constrained and under-constrained subgraphs are not depicted here.

Table 2. The set of equations and variables corresponding to the AC motor model

e1:Vs.v = Vs.p.v-Vs.n.v	v1: Vs.p.v
e2: 0 = Vs.p.i+Vs.n.i	v2: Vs.p.i
e3: Vs.i = Vs.p.i	v3: Vs.n.v
e4: Vs.v = Vs.V*sin(2*Vs.PI*Vs.freqHz *time)	v4: Vs.n.i
e5: Ra.v = Ra.p.v-Ra.n.v	v5: Vs.v
e6: 0 = Ra.p.i+Ra.n.i	v6: Vs.i
e7: Ra.i = Ra.p.i	v7: Ra.p.v
e8: Ra.R*Ra.i = Ra.v+Ra.s	v8: Ra.p.i
e9: Ra.p.v = 12	v9: Ra.n.v
e10: La.v = La.p.v-La.n.v	v10: Ra.n.i
e11: 0 = La.p.i+La.n.i	v11: Ra.v
e12: La.i = La.p.i	v12: Ra.i
e13: La.L*der(La.i) = La.v	v13: Ra.s

e14: Emf.v = Emf.p.v-Emf.n.v	v14: La.p.v
e15: 0 = Emf.p.i+Emf.n.i	v15: La.p.i
e16: Emf.i = Emf.p.i	v16: La.n.v
e17: Emf.w =der(Emf.flange_b.phi)	v17: La.n.i
e18: Emf.k*Emf.w = Emf.v	v18: La.v
e19: Emf.flange_b.tau = -Emf.k*Emf.i	v19: La.i
e20: G1.p.v = 0	v20: Emf.v
e21: Jm.w = der(Jm.phi)	v21: Emf.i
e22: Jm.a = der(Jm.w)	v22: Emf.w
e23: Jm.J*Jm.a = Jm.flange_a.tau+Jm.flange_b.tau	v23: Emf.p.v
e24: Jm.flange_a.phi = Jm.phi	v24: Emf.p.i
e25: Jm.flange_b.phi = Jm.phi	v25: Emf.n.v
e26:Jm.flange_a.phi=Emf.flange_b.phi	v26: Emf.n.i
e27: Emf.flange_b.tau+Jm.flange_a.tau = 0	v27: Emf.flange_b.phi
e28: Emf.n.i+G1.p.i+Vs.n.i = 0	v28: Emf.flange_b.tau
e29: G1.p.v = Emf.n.v	v29: G1.p.v
e30: Vs.n.v = Emf.n.v	v30: G1.p.i
e31: Emf.p.i+La.n.i = 0	v31: Jm.flange_a.phi
e32: La.n.v = Emf.p.v	v32: Jm.flange_a.tau
e33: Jm.flange_b.tau = 0	v33: Jm.flange_b.phi
e34: La.p.i+Ra.n.i = 0	v34: Jm.flange_b.tau
e35: Ra.n.v = La.p.v	v35: Jm.phi
e36: Ra.p.i+Vs.p.i = 0	v36: Jm.w
e37: Vs.p.v = Ra.p.v	v37: Jm.a

When check for structural singularities of the component Ra, if we assume the connector p is connected and the connector n is not connected, we can determine that Ra is structurally singular. The bipartite graph corresponding to Ra is presented in figure 6, where e1':f(Ra.p.v, Ra.p.i)=0 is the fictitious equation generated for flow variable Ra.p.i, and e2': Ra.n.i=0 is used to set flow variable Ra.n.i to zero.

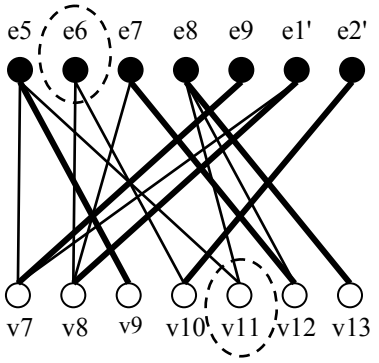

Figure 6. The bipartite graph corresponding to the component Ra with a maximum matching

In figure 6, e6 and v11 are free vertices, so the primitive component Ra is a MSS component. All other components of the model ACMotor are structurally nonsingular.

In this case, only the equation e9 appears in both the over-constrained subgraph and the component Ra. The variables that appear in both the under-constrained subgraph and the component Ra are v8, v9, v10, v11, v12 and v13. For this model, the error message is presented in figure 7.

```
Error: The model ACMotor is structurally singular.
The singularity comes from the component Ra.
There is 1 redundant equation in the equations:
   p.v=12;
1 equation is missing for the variables:
   p.i;
   n.v;
   n.i;
   v;
   i;
   s;
```

Figure 7. The error message for the model ACMotor

The third example is a modified AC motor depicted in figure 8, where the motor contains two ground points instead of one. The Modelica description of the modified motor model appears as follows:

model ModifiedMotor

```
  SineVoltage Vs(V=220,freqHz=50);
  Resistor Ra(R=0.5);
  Inductor La(L=0.1);
  EMF Emf;
  Inertia Jm(J=0.001);
  Ground G1;
equation
  connect(Vs.p, G2.p);
  connect(Ra.p, G2.p);
  connect(Ra.n, La.p);
  connect(La.n, Emf.p);
  connect(Emf.flange_b, Jm.flange_a);
  connect(Emf.n, G1.p);
  connect(Vs.n, G1.p);
end ModifiedMotor;
```

All the component models of the model ModifiedMotor are available in Modelica class libraries.

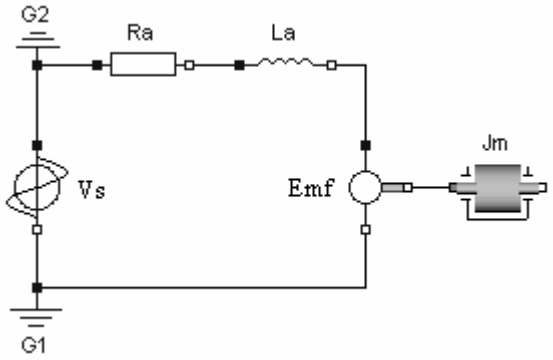

Figure 8. The AC motor with two ground points

The modified motor model also leads to a structurally singular problem where under-constrained and over-constrained situations appear simultaneously. When checking for structural singularities, all the components are determined to be structurally non-singular. So the top model ModifiedMotor is the MSS component.

In this case, the singularities are caused by improper use of components. To correct the model, one should remove the redundant ground point G2 instead of some equations and variables. Hence, for this model the following message is presented.

> Error: The model ModifiedMotor is structurally singular.
> The singularity may be caused by improper use of the components.
> Please check whether the components are used properly or not.

Figure 9. The error message for the modified motor

4 Comparison

The three examples presented in Section 3 illustrate that the MSA can automatically identify fault components and localize model singularities. It is very useful for the modeler to correct singular models. For a complex singular model, it is advisable to localize model singularities in such a way.

Currently, there are only a few methods that can help the modeler to debug singular equation-based models. For the first and the second examples, the method proposed in [1,2] is helpful, and can present efficient messages. However, for the third example where the singularities are not caused by equations and variables, the method can not deal with it. Moreover, it may be less efficient to debug complex models only by using structural information and semantic information.

If a structurally singular problem is caused by an over-constrained or under-constrained component, Dymola can identify such singular components. For the first example, Dymola can find the faulty component Ma and give the modeler efficient message. For the second example, Dymola does not find the faulty component Ra and considers the singularity is at the top level, and only informs the modeler that there is 1 one equation too many in a set of 7 equations and that 1 equation is missing for a set of 33 variables. For the third example, Dymola also consider the singularity is at the top level, and inform the modeler that there is 1 one equation too many and that 1 equation is missing, so the presented message is less helpful.

5 Conclusions

In this paper we have discussed an analyzer for declarative equation-based models. The examples presented in Section 3 are all quite trivial. However, they illustrate that it is possible to identify faulty components of a structurally singular model. From the modeler's point of view, the MSA is very beneficial because it can make correcting structurally singular models more quickly by automatically identifying faulty components and providing efficient error messages to show what is wrong.

The proposed techniques and strategies are also suitable for other object-oriented equation based modeling languages and not only for Modelica.

Acknowledgement

This work was supported by the National Natural Science Foundation of China (Grant No. 60574053), the National High-Tech Development 863 Program of China (Grant No. 2003AA001031), and the National Basic Research 973 Program of China (Grant No. 2003CB716207).

References

[1] Bunus P, Fritzson P. Automated static analysis of equation-based components. Simulation: Transactions of the Society for Modeling and Simulation International, 2004, 80(8):321-345

[2] Bunus P. An empirical study on debugging equation-based simulation models. In Proceedings of 4th International Modelica Conference, Hamburg, Germany, 2005

[3] Asratian A S, Denley T, Häggkvist R. Bipartite Graphs and their Applications, Cambridge University Press, 1998

[4] Dulmage A L, Mendelsohn N S. Coverings of bipartite graphs. Canadian Journal of Mathematics, 1963, 10:517-534

[5] Cellier FE, Elmqvist H, Otter M. Modeling from Physical Principles. The Control Handbook, CRC Press, pp.99-108

[6] Fritzson P. Principles of object-oriented modeling and simulation with Modelica 2.1, IEEE Press, 2003

[7] Hopcroft J E, Karp R M. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. SIAM Journal of Computing, 1973, 2(4): 225-231

[8] Uno T. Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs. In Lecture Note in Computer Science 1350, Springer-Verlag, 1997, pp. 92-101