Model–Based Diagnosis for Open Systems Fault Management

F. Steimann, P. Fröhlich and W. Nejdl
Institut für Rechnergestützte Wissensverarbeitung
Universität Hannover
Lange Laube 3, 30159 Hannover
$\{$*steimann, froehlich, nejdl*$\}$*@kbs.uni-hannover.de*

The information model chosen by the ISO for the management of open systems is object–oriented. We provide an effective mapping from the structural and behavioural specification of the managed objects of open systems to a compact logical form suitable for model–based diagnosis. Based thereon, we present an efficient algorithm that localizes faults by repairing logical models invalidated through system observations and show that it computes all minimal diagnoses.

Keywords: Open Systems Interconnection; fault management; fault localization; order-sorted logic; model-based diagnosis; predicate completion.

1. Introduction

Open systems are characterized by the possible interaction of an unlimited number of different devices, often made by different manufacturers, coming from different technological generations, and operated by different owners. Any of these devices can fail, and without the use of standardized procedures and protocols, the detection, localization, and correction of faults remains intractable. Fault management is therefore a prominent functional area of open systems interconnection (OSI) management ([8], [14]).

In theory, model-based diagnosis delivers on the requirements of fault management, fault localization especially. Based solely on a logical description of system behaviour and a set of logical sentences representing the topology and the observations made, it isolates potentially faulty components by simulating the system's correct operation and resolving the discrepancy with the observations by assuming the behaviour of selected components as abnormal [22]. In practice, however, model-based diagnosis poses two nontrivial problems that hinder its widespread utilization:

– the creation of logical descriptions of systems that are not themselves based on logic is, at least for non-logicians, difficult [13], and
– simulating the logical descriptions is computationally expensive.

With the work presented in this article, we address these problems

– by defining a conceptually clean and easy to adopt notation for causal system descriptions that is based on a syntax standardized in the context of OSI management (Section 3),
– by specifying an effective mapping from these descriptions to a compact logical form (Section 4), and
– by adapting our proven model repair algorithm DRUM-II to efficiently localize faults in an open systems environment (Section 5).

2. An example

Typical representatives of open systems are *G*lobal *S*ystem for *M*obile communication (GSM) management networks. Hand sets, base stations, switching equipment, etc. are made by different manufacturers and operated by different service providers. Resources such as transmission lines or satellite links are utilized by different parties, and the management of these resources requires a coordinated approach. The standardization of all pertinent management activities falls under the telecommunication management network (TMN) standards, which are built upon the X.700 series [14, 17, 15, 16].

For illustrative purposes, let us consider the following scenario. In a base station system (BSS), a subunit of a GSM network consisting of a number of base stations (BSs) linked to a base station controller (BSC), all information is conveyed via microwave links (MLs). The BSC collects all management information from the connected BSs and passes them on to a superordinate instance. For economical reasons, some base stations are not directly linked to the base station controller, but are chained using multi-drop connectors. Therefore, when a microwave link fails, depending on the actual topology one or more base stations may become unreachable.

To achieve higher availability of individual BSs, redundant links may be implemented. A simple BSS in which every BS but bs3 has two uplinks (links that lead in the direction of the BSC) is shown in Figure 1. In this BSS, there exist at least two alternative paths between every BS (but bs3) and the BSC.

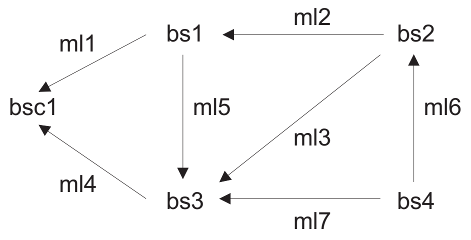

Figure 1: A sample base station system

A typical fault situation in such a BSS is detectable when the BSC receives a signal from a BS at one or more, but not at all of its ports at which the signal would have been expected (that is, at which paths from the BS end). In our example, such is the case if the BSC receives a signal from, say, bs1 at the port of ml1 or ml4, but not at both. No indication of a fault is given if there are no signals at the BSC, because a BS does not have to send signals. On the other hand, signals at all ports of the BSC do not prove the absence of a fault, since redundancy may mask the failure of one or more distant links connecting remote BSs.

The problem of identifying the potentially abnormal links solely from the observations made at the BSC can be quite tricky even for small topologies; it becomes unwieldy in larger BSS with 20–30 BSs, because the number of available paths grows combinatorially. Another aspect of the problem discouraging *ad hoc* solutions is that automatic reconfiguration techniques triggered by fault detection may accept diagnosed links as broken and activate alternative ones, dynamically changing the network configuration.

3. The OSI management view

ISO norm 7498–4 requires that, for the management of open systems, all relevant entities be modelled as managed objects. “A managed object is the OSI management view of a resource that is subject to management, such as a layer entity, a connection or an item of physical communications equipment. Thus, a managed object is the abstracted view of such a resource that represents its properties as seen by (and for the purposes of) management.” [14] The managed objects of our example are the network elements (NEs) comprising the base stations, the microwave links and the base station controller, and the signals that are forwarded between them. The management information describing these objects is specified according to standardized templates called managed object class (MOC) definitions, where a MOC is defined as a named set of managed objects sharing the same characteristics [17].

3.1. Managed object class definitions

In our example, the MOC definitions are the following[1]:

```
ne MANAGED OBJECT CLASS
   DERIVED FROM top;
   CHARACTERIZED BY
      neMBDPackage PACKAGE;
;
bs MANAGED OBJECT CLASS
   DERIVED FROM ne;
   CHARACTERIZED BY
      bsMBDPackage PACKAGE;
;
ml MANAGED OBJECT CLASS
   DERIVED FROM ne;
   CHARACTERIZED BY
      mlMBDPackage PACKAGE;
;
bsc MANAGED OBJECT CLASS
   DERIVED FROM ne;
   -- BSCs have no special characteristics
;
signal MANAGED OBJECT CLASS
   DERIVED FROM top;
   CHARACTERIZED BY
      signalMBDPackage PACKAGE;
;
```

The `DERIVED FROM` clause establishes a multiple hierarchy by specifying a subclass relation among classes. The defined class inherits the characteristic of all its direct and indirect superclasses. `Top` is predefined and the highest class in the MOC hierarchy.

The `CHARACTERIZED BY` clause associates the class with a *package*, a logical grouping of character-

[1] We designed our MOC definitions to suit illustrative purposes. They correspond roughly to the MOC definitions bsc, btsSiteManager, and lapDLink provided with the GSM standard 12.20 [7].2

istics that is named for possible (re)use by other MOC definitions. The package introduces *attributes* and a specification of *behaviour* (plus a few other characteristics which are of no concern for our purposes).

Attributes associate managed objects with values or other objects. They correspond to the fields of a record in common programming languages. The behaviour clause of a package is to specify, among other things, dependencies between values of particular attributes ([15] Sect. 5.1.2.4).

We use attributes to specify the relationships among managed objects. This includes topological data, i.e. the interconnection of the NEs, and the association of NEs with signals. Following the OSI management information model, attributes may be single valued (the default) or set–valued. Special, untyped attributes such as `active` serve as binary status indicators required in the specification of behaviour. They accord to the standard since an attribute value assertion is itself either true or false ([15] Sect. 3.8.8)[2].

```
neMBDPackage PACKAGE
   ATTRIBUTES
      -- the next NE in the direction of
      -- the BSC
      uplink PERMITTED VALUES ne,
      -- the signal at the port associated
      -- with the uplink
      signalAtUpport PERMITTED VALUES
         SET OF signal
      -- the signal at the port in downward
      -- direction
      signalAtDownport PERMITTED VALUES
         SET OF signal
   BEHAVIOUR
      nePropagationMBDBehaviour BEHAVIOUR;
;
bsMBDPackage PACKAGE
   ATTRIBUTES
      uplink PERMITTED VALUES SET OF ml;
      -- a BS is exclusively connected to MLs
      active;
      -- flag indicating whether the bs has
      -- sent a signal
   BEHAVIOUR
      bsPropagationMBDBehaviour BEHAVIOUR;
;
mlMBDPackage PACKAGE
   BEHAVIOUR
      bsPropagationMBDBehaviour BEHAVIOUR;
;
signalMBDPackage PACKAGE
   ATTRIBUTES
      sender PERMITTED VALUES bs;
   BEHAVIOUR
      signalGenerationMBDBehaviour BEHAVIOUR;
;
```

[2]Deviating from the standard notation, we omit separate attribute type definitions and specify the type of an attribute inline. Here, the class label following the `permitted values` keyword takes the place of a `with attribute syntax <ASN.1 type definfition>` clause, which is to specify the attribute's value set. If we adhered to the standard, either a distinct ASN.1 type would have to be specified for every MOC, or the permitted value restrictions would have to be expressed as behaviour constraints, which is a less obvious place for such definitions.

3.2. *Management information base*

The management information necessary to operate a BSS can now be specified in terms of MOC definitions. Each attribute value is specified in the form of an attribute value assertion. For example, the topological information of the BSS depicted in Figure 1 would be expressed as:

```
bsc1 :  bsc []
bs1 :  bs [uplink = ml1, ml5]
bs2 :  bs [uplink = ml2, ml3]
bs3 :  bs [uplink = ml4]
bs4 :  bs [uplink = ml6, ml7]
ml1 :  ml [uplink = bsc1]
ml2 :  ml [uplink = bs1]
ml3 :  ml [uplink = bs3]
ml4 :  ml [uplink = bsc1]
ml5 :  ml [uplink = bs3]
ml6 :  ml [uplink = bs2]
ml7 :  ml [uplink = bs3]
```

The collection of such management information constitutes what is referred to as the management information base (MIB). However, the above listed entries of a MIB are not complete: for example, all events (reported through notifications) are recorded in a log, and this log is also a part of the MIB. The difference is that the former are static or invariant over the lifetime of the network configuration, while the latter are inherently dynamic.

3.3. *Specifying behaviour*

Model-based diagnosis is based on a structural and a functional or causal description of the system to be diagnosed. While the structural aspects of the system description are covered by the attribute type definitions and the MIB, the behavioural aspects are naturally placed in the behaviour clause of a package definition. Although the standards specify all dependencies between attributes using natural language ([16] Sect. 8.9.3.1), model-based diagnosis requires that behaviour be specified formally. Therefore, and to alleviate the difficulty of formalization, we introduce in the following an intuitive syntax whose relationship to logical system descriptions will be detailed in the next section.

We identify four distinct patterns of causal dependency in our example.

1. If an NE is connected to another NE (via the uplink attribute), any signal that is present at its upport should also be present at the connected NE's downport. Being a typical example of the type of behaviour commonly referred to a propagation, this is expressed by

```
nePropagationMBDBehaviour BEHAVIOUR
   DEFINED AS
      !$ signalAtUpport
         PROPAGATES TO
         uplink.signalAtDownport $!
;
```

   The logical interpretation of the dot notation and the `PROPAGATES TO` operator is subject of the next section. Note that, although `nePropagationBehaviour` is specified for the MOC `ne`, it is inherited by the classes `bs`, `ml` and `bsc`.

2. Base stations always pass on signals, i.e., they propagate signals from their downports to their upports. This is conveniently expressed by

```
bsPropagationMBDBehaviour BEHAVIOUR
   DEFINED AS
      !$ signalAtDownport
         PROPAGATES TO
         signalAtUpport $!;;
;
```

3. Microwave links are to convey signals between BSs and the BSC. However, MLs can break down and in this case fail to propagate signals. Thus, their behaviour is adequately described by

```
mlPropagationMBDBehaviour BEHAVIOUR
   DEFINED AS
      !$ NORMALLY signalAtDownport
         PROPAGATES TO
         signalAtUpport $!;;
;
```

   `NORMALLY` is to indicate that the precondition to signal propagation is that the ML is in normal operating state.

4. If a base station is active, it emits a signal the sender of which is the active BS itself. To avoid existential quantification of the signal, we place this behaviour as the signal's and write

```
signalGenerationMBDBehaviour BEHAVIOUR
   DEFINED AS
      !$ sender.active
         IMPLIES
         sender.signalAtUpport = SELF $!;
;
```

   meaning that a signal whose sender is active will always be found at the sender's upport. `SELF` is an implicit variable known from object–oriented programming, denoting the current object of the class for which the behaviour is specified. Of course, the usual logical connectives and, or etc. as well as parentheses are also admissible in our syntax.

Note that the behaviour of subclasses may be specialized under the premise that the redefinition does not contradict the inherited behaviour ([15] Sect. 5.2.2.6). While compliance with this requirement is difficult to check for the verbal specifications of behaviour that are commonplace in practice, using our formal notation it translates to testing the satisfiability of the clauses obtained from the new and inherited behaviour - a decidable problem within the given finite domains.

4. Transition to logic

So far, we have presented a framework for the creation of system descriptions that are intuitive as well as compatible with widely accepted standards. To perform model–based diagnosis in our sample BSS, however, we have to map the given description to a logical representation.

In model-based diagnosis [22] the device under consideration is described by a logical system description[3], SD, denoted as a set of clauses, and a set of objects, called OBJ (in our case the NEs and the signals). The predicate ab is introduced to indicate the operational status of each object $c \in OBJ$: $\neg ab(c)$ means that the object behaves as expected, while $ab(c)$ means that it functions abnormally. The models of SD correspond to the system's possible states. They comprise normal (in which all objects are operational) and abnormal (in which one or more objects do not behave as desired) states. The observed behaviour is characterized by a set of literals OBS, which reduces the num-

[3] To avoid confusion, we speak of a system description and not of a (system) model. We reserve the term model for logical models represented by sets of atoms.

ber of possible states (models) by fixing the interpretation of the observed atoms.

The usual unisorted approach to model–based diagnosis does not immediately capture type hierarchies. As the standard workaround of implementing inheritance through implication results in intermediate representations that are unnecessarily big, we resort to an interlingua that is at the intersection of object-orientation and predicate calculus: *order-sorted logic*.

4.1. Interlingua order–sorted logic

In order–sorted logic, the universe of discourse is partitioned into a number of overlapping subsets, S, called *sorts*. The set inclusion relationship induces a half-order $\leq$, called the *subsort* relationship, on the sorts which is naturally interpreted as a taxonomic polyhierarchy. The definition of order–sorted logic we rely on in the sequel follows the ones found in [2] and has been adapted to suit our needs.

A *signature* Σ is given by S, C, P, where

- S is a partially ordered set of sorts with greatest element top,
- $C = \{c_i : s_j | s_j \in S\}$ is a finite set of sorted constants, and
- $P = \{p_i : s_{i,1} \times ... \times s_{i,m_i} | s_{ij} \in S\}$ is a set of predicates the places of which are also sorted.

Since our signature contains no function symbols, our *terms* t are either sorted constants $c : s$ or sorted variables $x : s$. Each term has the sort of its constituent. From terms, formulae are constructed as in unisorted predicate logic, with the additional requirement that every atomic formula $p(t_1 : s_1, \ldots, t_n : s_n)$ must be *well–sorted*, i.e., $p : s'_1 \times \ldots \times s'_n$ and $s_1 \leq s'_1, \ldots, s_n \leq s'_n$.

As a prerequisite for our expansion procedure discussed later, we have to enforce Herbrand interpretations, which are no restriction to our problem domain. This is achieved by requiring

$$\forall x : top \quad \bigvee_{s_j \in S} \bigvee_{c_i : s_j \in C} x = c_i : s_j$$

and

$$\bigwedge_{c:s \in C} \bigwedge_{c':s' \in C \setminus \{c:s\}} c \neq c'$$

the order–sorted variants of the domain closure assumption (DCA) and the uniqueness of names assumption (UNA). We refer to an order–sorted theory which includes these axioms as an order–sorted fixed domain theory, the order–sorted equivalent of fixed domain theories as described in [20].

4.2. Translation scheme

Managed object classes map to sorts

A MOC is an abstraction of managed objects sharing the same characteristics. Each managed object represents an entity of the real world. MOCs correspond to subsorts of the universe, as do sorts; the mapping is therefore one–to–one.

The MOC definitions from our example map to the following sort declarations:

$$\begin{array}{l} ne \leq top \\ bsc \leq ne \\ bs \leq ne \\ ml \leq ne \\ signal \leq top \end{array}$$

Attributes map to predicates

We declare an attribute a of class C with value space V as a binary predicate $a : C \times V$ and write $a(c, v)$ for a single–valued attribute value assertion and $a(c, v_1), \ldots, a(c, v_n)$ for a set–valued attribute value assertion. In conformance with the rules of inheritance specified for the attributes' permitted value sets ([15] Sect. 5.2.2.3, [16] Sect. 8.3.3.1), predicate declarations can be overloaded, restricting the value space to a subset of the inherited value set. Untyped attributes have no value space and therefore correspond to unary predicates.

The predicate declarations obtained from our example are:

$$\begin{array}{l} uplink : ne \times ne \\ signalAtUpport : ne \times signal \\ signalAtDownport : ne \times signal \\ uplink : bs \times ml \\ active : bs \\ sender : signal \times bs \end{array}$$

The declaration

$$ab : top$$

is added automatically.

Behaviour clauses map to axioms

The behaviour clauses of a MOC definition express rules that apply to the instances of the involved classes, i.e. the defining class and the classes referred to through attribute labels. The logical equivalent of a behaviour clause is an axiom.

The unqualified attribute value reference a translates to the atomic formula

$$a(i:C, v:V)$$

in which i (corresponding to the implicit variable `this` or `self` in object-oriented programming languages) and v are variables for the object and its attribute value, respectively. The sorts C and V are the ones specified in the predicate declaration corresponding to the classes attribute definition. Note that if we want to enforce single–valuedness of an attribute, we have to add an axiom expressing this; however, for reasons given below in this particular example we do not.

If `a` is an untyped attribute, the reference translates to

$$a(i:C).$$

A chained attribute value reference `a.b` translates to the formula

$$a\,(i:C, v:V) \rightarrow b\,(v:V, w:W)\,.$$

Likewise, `a.b.c` translates to $a\,(i:C, v:V) \rightarrow (b\,(v:V, w:W) \rightarrow c\,(w:W, x:X))$, and so forth.

We express the propagation of a value from one attribute to another by `a=x IMPLIES b=x`, which translates to

$$a\,(i:C, x:V) \rightarrow b\,(i:C, x:W)\,.$$

As will be seen, implication covers only one direction of propagation, (the "causal" one from origin to destination). The reverse direction will be dealt with below.

Because propagation is a frequent form of causal behaviour and because the only purpose of the variable `x` is to force equality of the attribute values, we allow propagation to be abbreviated as `a PROPAGATES TO b`. Another convenient abbreviation is the expression `NORMALLY` φ, which puts φ as a consequent of $\neg ab(i)$, i.e.,

$$\neg ab(i) \rightarrow \varphi.$$

As a final step, we universally quantify each variable of a formula so obtained over the greatest common subsort of all its occurrences in that formula[4]. Following this translation scheme, the behaviour clauses of our MOC definitions produce the following four sentences, which are the axioms of our system description ($A1$, $A3$ and $A4$ are simplifications).

[4]This requires that the greatest common subsort of any two sorts exists and is uniquely determined. In our domain, the absence of such a sort indicates a design error and aborts translation.

$$\begin{aligned}
&A1 : \forall i : ne, x : ne, s : signal\\
&\quad (uplink(i,x) \wedge signalAtUpport(i,s))\\
&\qquad \rightarrow signalAtDownport(x,s)\\
&A2 : \forall i : bs, s : signal\\
&\quad signalAtDownport(i,s)\\
&\qquad \rightarrow signalAtUpport(i,s)\\
&A3 : \forall i : bs, s : signal\\
&\quad (\neg ab(i) \wedge signalAtDownport\,(i,s))\\
&\qquad \rightarrow signalAtUpport(i,s)\\
&A4 : \forall i : signal, x : bs\\
&\quad (sender(i,x) \wedge active(x))\\
&\qquad \rightarrow signalAtUpport(x,i)
\end{aligned}$$

The management information base maps to constants and facts

No system description would be complete without the enumeration of its objects and the topological information specifying their interconnection. These are generated from the elements of the MIB such that each object identifier maps to a constant of the corresponding sort and each attribute value assertion maps to an atomic formula (that is, of course, ground). In our example, the constants are

$bsc1 : bsc$	$bs1 : bs$	$ml1 : ml$
	$bs2 : bs$	$ml2 : ml$
	$bs3 : bs$	$ml3 : ml$
	$bs4 : bs$	$ml4 : ml$
		$ml5 : ml$
		$ml6 : ml$
		$ml7 : ml$

and the atoms are

$uplink\,(bs1, ml1)$	$uplink\,(ml1, bsc1)$
$uplink\,(bs1, ml5)$	$uplink\,(ml2, bs1)$
$uplink\,(bs2, ml2)$	$uplink\,(ml3, bs3)$
$uplink\,(bs2, ml3)$	$uplink\,(ml4, bsc1)$
$uplink\,(bs3, ml4)$	$uplink\,(ml5, bs3)$
$uplink\,(bs4, ml6)$	$uplink\,(ml6, bs2)$
$uplink\,(bs4, ml7)$	$uplink\,(ml7, bs3)$

Because the MIB contains no instances of signal, we add

$$\begin{array}{ll} s1 : signal & s2 : signal\\ s3 : signal & s4 : signal, \end{array}$$

one constant for each BS. We need four constants because the attribute sender of signal is single-valued and we expect all BSs to emit signals. An assignment of signals to the BSs is given by the facts

$$\begin{array}{ll} sender\,(s1, bs1) & sender\,(s2, bs2)\\ sender\,(s3, bs3) & sender\,(s4, bs4)\,. \end{array}$$

Not all attribute types have entries in the MIB. For those that have, however, we subscribe to the closed world assumption, i.e., we agree that the attribute value assertions recorded in the MIB are true, while those that are not are false. This implies, for example, that the proposition $uplink(ml1, bs2)$ is false.

Note that the single–valued attributes of our example are all covered by the MIB, so that we do not need axioms enforcing single–valuedness.

4.3. Reduction to propositional logic

First order predicate logic allows propositions over variables that range over infinite domains. In technical domains, however, the sets of objects dealt with are usually finite. In the case of OSI fault management, all objects of interest are listed in the MIB. This circumstance is exploited to reduce the complexity of the diagnostic task.

In an order-sorted fixed domain theory, the universal quantification $\forall x : s\ \varphi(x)$ is equivalent to

$$\bigwedge_{s_i \leq s} \bigwedge_{c : s_i} \varphi\left(x\right)_{[x \leftarrow c]}$$

Likewise $\exists x : s\ \varphi\left(x\right)$ is equivalent to

$$\bigvee_{s_i \leq s} \bigvee_{c : s_i} \varphi\left(x\right)_{[x \leftarrow c]} .$$

Because only well–sorted substitutions must be considered, the blowup effect of rolling out the quantifications is greatly reduced. Note how well–sorted substitutions elegantly account for inheritance: although axiom $A1$ quantifies over NEs, a class that itself does not have instances, i and x are instantiated with all BSs, MLs and the BSC. To implement inheritance in unisorted predicate logic, every axiom must be guarded by a number of sort or type predicates (one per variable) and new axioms of the form $\forall x\ subsort(x) \rightarrow supersort(x)$ (one per subsort relationship) must be added to the system description. It follows that the more use of inheritance is made, the larger the size of the clause set gets. By resorting to order–sorted logic, however, inheritance is free.

By applying quantifier expansion to all axioms of our system description, we arrive at a propositional representation, i.e., at a set of sentences that consist exclusively of ground atomic formulae and logical connectives. These are transformed into a clause set using a standard procedure for transformation into conjunctive normal form.

The size of the clause set is greatly reduced by incorporating the knowledge entered in the MIB. Every occurrence of an atom that is contained in the fact base can be replaced by true, while every atom that is known to be false can be replaced by false. Simplification allows it that every clause containing a literal evaluating to true be dropped, and every literal evaluating to false be removed from its clause.

4.4. Completing propagation

We set out with the objective to create a formalism that allows the easy specifications of system behaviour. And indeed, the system behaviour specified in our MOC definitions is straightforward and its logical translation is still fairly intuitive. Unfortunately, it turns out that it is also incomplete.

Axiom $A1$ implies that if a signal is present at the upport of a NE, then it must also be present at the downport of the next NE in line. If, however, the antecedent fails, that is, if a particular signal is not at the upport, the implication allows anything to be at the downport of the next NE, including the very same signal. This behaviour is partly desired, as the absence of a signal at one upport must not impede the presence of that signal at the connected downport (because it may be propagated by another link). On the other hand, by letting anything be assumed at the downport of a NE, the network systematically gets flooded (through the propagation expressed with $A2$ and $A3$) with assumptions for which there is no reason.

Axiom $A1$ has models that contradict our notion of propagation. To eliminate these models, we can add

$$\begin{array}{l} \forall x : ne, s : signal \\ \quad signalAtDownport(x, s) \\ \quad\ \rightarrow \exists y : ne \\ \quad\quad uplink(y, x) \wedge signalAtUpport(y, s) \end{array}$$

meaning that if an NE has a signal at its downport, this signal must have been propagated from the upport of at least one of its downward successors. This includes the case that the NE has no such element, in which case there can be no signal at its downport.

To cover their intended meaning, axioms $A2$ through $A4$ can be complemented accordingly. However, because they share the same conclusion (a proposition over $signalAtUpport$), only one of the three preconditions needs to hold, making a combined formula for the reversal of propagation necessary. This is precisely what is covered by predicate completion [4, 12].

We have shown elsewhere that a complete formalization of propagation is impossible in first order predicate logic [10], a result making any attempt to complete our axioms of propagation vain. However, the objections no longer persist in the domain of propositional logic in which the clause set has been instantiated against a given topology. In fact, in all acyclic causal theories the clause set derived from the causal axioms can be completed automatically [5, 18] to fully cover the semantics of propagation by adding for all atoms a which are an instance of the predicate in question the propositional sentence

$$a \rightarrow c_1 \vee \ldots \vee c_n$$

in which the c_1 through c_n are the remainders of all clauses containing a in positive form, i.e., the

$$\begin{array}{l} c_1 \rightarrow a \\ \vdots \\ c_n \rightarrow a \end{array}$$

contained in the clause set.

4.5. *The results of translation*

Our completed propositional clause set is the system description SD required in the framework of model–based diagnosis. This clause set is relatively compact, because the inflationary effect of quantifier expansion is compensated by the simplifications resulting from substituting the atoms covered by the fact base with their truth values. In fact, the size of SD is linear in the number of NEs and the number of signals.

The set of objects OBJ is equal to the set of constants derived from the MIB.

5. Doing diagnosis: localizing faults with the DRUM–II algorithm

The DRUM–II algorithm was motivated by Chou and Winslett, who have implemented a system (IMMORTAL) for model–based belief revision [3]. It starts with a system description SD and a model M containing no abnormals, such that M is a model of SD. This model is invalidated by inserting the possibly contradictory observations. DRUM–II repairs the resulting inconsistent interpretation by an exhaustive search procedure returning a set of models representing all minimal diagnoses.

5.1. *Theoretical foundation*

The behaviour clauses together with the topological entries in the MIB compile to the propositional clause set SD as described in the previous sections. Observations OBS add interpretations for the atoms observed. Computing a diagnosis is then equivalent to finding a model of SD of which OBS is a subset.

Definition 1: A *diagnosis* of (SD, OBJ, OBS) is a set $\Delta \subseteq OBJ$ such that $SD \cup OBS \cup \{ab\,(c)|\,c \in \Delta\,\} \cup \{\,\neg ab\,(c)\,|c \in OBJ - \Delta\}$ is consistent. Δ is called a *minimal diagnosis* if it is the minimal set (with respect to the subset relationship) with this property.

Next we characterize the diagnosis Δ in terms of the models of $SD \cup OBS$. Given a model M of $SD \cup OBS$, the extension of the ab predicate, denoted by $M|ab|$, tells us which components are considered faulty by M. Thus, every model M of $SD \cup OBS$ corresponds to a (possibly non-minimal) diagnosis $M|ab|$. To characterize the minimal diagnoses using models, we define an order $\leq_{ab}$ on the models, based on the abnormal components they contain.

Definition 2: Given models M_1 and M_2, we define $M_1 \leq_{ab} M_2$ iff $M_1|ab| \subseteq M_2|ab|$, $M_1 \sim_{ab} M_2$ iff $M_1|ab| = M_2|ab|$, and $M_1 <_{ab} M_2$, iff $M_1 \leq_{ab} M_2$ and not $M_1 \sim_{ab} M_2$.

The minimal diagnoses of (SD, OBJ, OBS) correspond to the $\leq_{ab}$ minimal models of $SD \cup OBS$.

Proposition 3: A set Δ of objects is a minimal diagnosis of (SD, OBJ, OBS) iff there exists a $\leq_{ab}$ minimal model M of $SD \cup OBS$ such that $\Delta = M|ab|$.

The proof of this proposition is found in [9].

Obviously, there can be several models corresponding to a single diagnosis. DRUM–II therefore computes only one model out of every $\sim_{ab}$ equivalence class. Such a representative set of minimal models is called a *transversal*.

Definition 4: Let $\mathcal{M}$ be a set of models and $\mathcal{M}/\sim_{ab} = \{[M]\,|M \in \mathcal{M}\}$ be a set of equivalence classes. A set $\mathcal{M}'$ is called a transversal of $\mathcal{M}/\sim_{ab}$ if $\mathcal{M}'$ contains exactly one member out of every equivalence class in $\mathcal{M}/\sim_{ab}$.

The following proposition shows that computing a transversal of the minimal models is a correct implementation of model–based diagnosis.

Proposition 5: Let $\mathcal{M}$ be the set of all $\leq_{ab}$ minimal models. Let $\mathcal{M}'$ be a transversal of $\mathcal{M}/\sim_{ab}$. Then $\{M\,|ab|\,|M \in \mathcal{M}'\}$ is the set of all minimal diagnoses of (SD, OBJ, OBS).

The proof of Proposition 5 can be found in [9]. While previous results on the relation of diagnosis and circumscription [1, 21] focus on formalizing stronger forms of explanation (than provided by consistency–based diagnosis), this proposition establishes an interesting connection between consistency–based diagnosis and circumscription, because $\mathcal{M}$ is the set of all models obtained by circumscribing $SD \cup OBS$ in ab, while varying all other predicates.

5.2. How DRUM–II works

DRUM–II repairs models invalidated by observations. The only way it can do this is by successively altering the truth values of atoms that have themselves not been observed until a new model has been found. The efficiency of this procedure depends critically on the choice of the next atom to flip - there is always the chance that flipping an atom does more harm than good. To minimize this chance, the DRUM–II algorithm incorporates the following characteristics.

1. *It only flips atoms that occur in invalidated clauses.*
   Doing this is guaranteed to make these clauses evaluate to true, but usually will invalidate other clauses. This is particularly the case for clauses implementing propagation.
2. *It expands the search tree depth first.*
   This simple strategy focuses the search by following the paths of propagation: the next atom to be flipped is from a clause that has been invalidated by the previous flip.
3. *It considers clauses with the fewest flippable atoms first.*
   This heuristic entails that branches with small branching factors are considered first. Because it is essential for cutting down the search space, it overrules the depth first strategy.
4. *It employs iterative deepening to find minimal diagnoses first.*
   Searching for minimal diagnoses only offers another chance of pruning the search tree. Maintaining a list of diagnoses already found, the choice of an abnormal as the next flippable atom is unproductive if this makes the current diagnosis a superset of one already found. By limiting the cardinality of Δ and by successively incrementing this limit (a technique known as iterative deepening [19]) it can be guaranteed that minimal diagnoses are found first [11].

The kernel of the DRUM–II algorithm is implemented by the following Java code:

```
void solve() {
   if (solvenda.resolveable()) {
      Atom atom = solvenda.nextFlippable();
      flip(atom);
      //locks atom & updates solvenda
      if (solvenda.isEmpty())
         diagnosis.keep(); // solution found
      else
         solve();
         // try with the atom flipped
      flip(atom);
      // flip it back; remains locked
      // (= not flippable)
      solve();
      // with the atom's original value,
      // but not flippable
      release(atom); // remove lock
   }
   // else no solution with current
   // truth values
   return;
}
```

Every atom maintains a lock flag, preventing it from being selected twice on the same branch. Also, by not unlocking the atom on flipping it back to its original value we implement a technique known as factorization. As can easily be verified, the search is exhaustive with the exception that

– it does not continue a branch once a solution has been found, since expanding such branches can only lead to non-minimal diagnoses; and
– it does not descend into branches that cannot lead to a solution, i.e., that contain invalidated clauses having no more flippable atoms.

DRUM–II maintains a list of invalidated clauses, called solvenda, which drives the algorithm. Implementing the above listed characteristics, its `nextFlippable` function returns the next atom to be flipped. The solvenda also recognizes unresolvable inconsistencies, clauses whose number of flippable atoms is zero.

Proposition 6: Given an initial model of the system description SD, the DRUM–II algorithm computes a transversal of the $\leq_{ab}$ minimal models of $SD \cup OBS$ and thus by Proposition 5 returns all minimal diagnoses.

Again, the formal proof can be found in [9].

5.3. A sample trace

It is in the nature of our problem that, as long as no observations are added, assuming all atomic propositions to be false is perfectly consistent with the system description as represented by the clause set. (The BSs are not active so that no signals originate at their upports; the MLs are not abnormal (i.e., functional), but there are no signals at their downports they could propagate, as for the BSs.) Therefore, on bootstrapping the interpretation of all atoms is set to false.

With the first observations, however, the picture may change. If, for example, a signal from bs2 is observed at the upport of ml1 but not at that of ml4, i.e., if

$$OBS = \{\{signalAtUpport(ml1, s2)\}, \{\neg signalAtUpport(ml4, s2)\}\},$$

the clause

$$\{ab\,(ml1)\,, \neg signalAtUpport\,(ml1, s2)\,, signalAtDownport\,(ml1, s2)\},$$

introduced through the completion of axiom $A3$ is invalidated, requiring the revision of the truth value of $signalAtDownport\,(ml1, s2)$ or $ab(ml1)$. DRUM–II chooses one of the two atoms, say the first (the other one will be considered upon backtracking), and flips its value to true, thereby backpropagating the signal from the upport to the downport of ml1 as shown in Figure 2 a). This in turn invalidates another clause,

$$\{\neg signalAtDownport\,(ml1, s2)\,, signalAtUpport\,(bs1, s2)\},$$

introduced through the completion of axiom $A1$, whose repair iteratively propagates the signal back to its source, the upport of bs2. Backpropagation terminates with setting the truth value of $active(bs2)$ in

$$\{\neg signalAtUpport\,(bs2, s2)\,, signalAtDownport\,(bs2, s2)\,, active(bs2)\}$$

to true, the situation of Figure 2 b). Following the same principles, but using clauses derived from the original (uncompleted) axioms, signal s2 is propagated forward along all other paths toward the BSC. On these paths, assuming all MLs to be functional leads to an unresolvable conflict with the observation $\neg signalAtUpport(ml4, s2)$ (Figure 2 c)), forcing DRUM–II to revise some of these assumptions, as indicated in Figure 2 d).

After backtracking has traversed all branches, DRUM-II terminates with the diagnoses

$$\{\{ml4\}, \{ml3, ml5\}\},$$

meaning that either ml4 is broken or, less obviously, ml3 and ml5 are defective. Because with the given information there is no way to decide which of the two alternatives is the correct diagnosis, two candidates (called hypotheses hereafter) are generated and new observations are awaited.

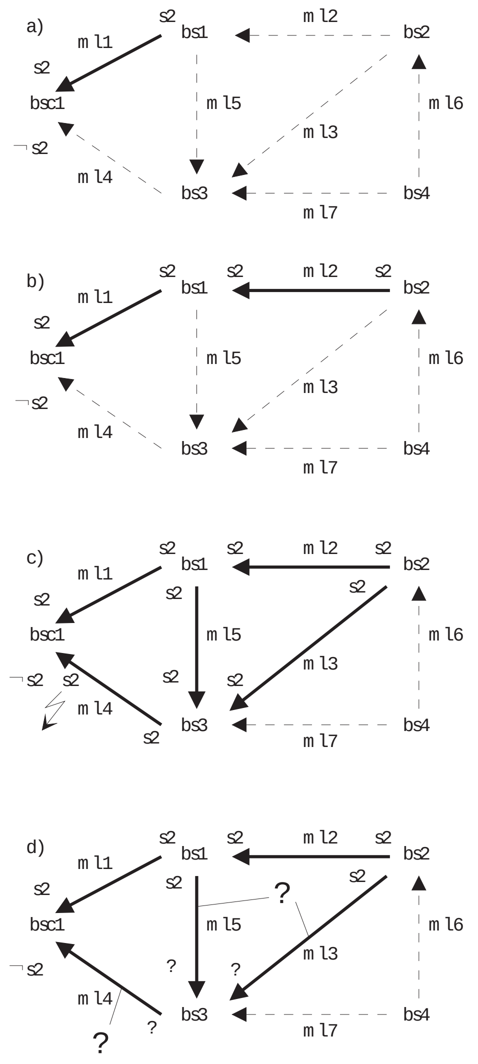

Figure 2: The propagation of signals
a) from the source of observation b) to the origin and
c) from the origin back along all other paths to the
source, where it leads to an inconsistency that can d)
only be resolved by assuming links abnormal

Different hypotheses are handled by maintaining different models, one for each hypothesis, in which the assumed faults, like the initial observations, are not subject to revision. New observations are then added to the models in parallel. Three situations may arise:

1. the hypothesis and the new observations are at unresolvable conflict, in which case the hypothesis must be dropped;
2. the hypothesis and the observations are compatible. In this case, the new observations either
   (a) confirm the fault assumptions of the hypothesis, or
   (b) add new ones that are independent of the others.

In any case, a hypothesis is proven if all its alternatives have been discarded. Returning to our example, if the next observation is

$$\{signalAtUpport(ml4, s3)\}$$

the first hypothesis must be dropped, while the second is confirmed, making the latter the only candidate for diagnosis. The inherent advantage of our procedure of repairing models is that it always maintains a valid model that has all previous observations and assumptions of fault incorporated, so that with each new step only the new observations need be integrated.

5.4. *Handling reconfiguration*

In large open systems, reconfiguration is a natural process that *per se* enjoys no special status. The fact that the configuration has changed is, like all other events, reported through notifications recorded in the MIB. Ideally, a reconfiguration of the system takes place without the whole or parts of it needing to be restarted. If model–based diagnosis is to be integrated seamlessly into the open system, it must react to configuration changes without undue delays.

We take this into account by allowing incremental instantiation of variables. Axioms constraining configuration parameters that change only seldom are compiled into the system description at an early stage. Those that change more frequently but nevertheless exhibit a certain persistence are compiled into the system description at later stages and can be removed by rolling it back to the state of an earlier instantiation. Finally, configuration data that change regularly may be considered as observations (completed under the closed world assumption) and get involved only at the model repair phase, albeit at the cost of a considerably larger system description.

6. Discussion

The information model chosen by the ISO for OSI management is object oriented. A class hierarchy and associated rules of inheritance provide for the definition of standards that are easily extended (specialized) to suit particular management, a new standard's, or a specific manufacturer's/operator's needs. It is this ease of extension that has lead to the widespread acceptance of the object–oriented paradigm.

Although model-based diagnosis is component ($\simeq$object)–oriented in principle, the formalizations of system descriptions in use today do not exploit the advantages offered by an object-oriented information model[5]. With our work we have defined a complete transformation from an object–oriented system description to a compact propositional clause set devoid of the overhead usually associated with inheritance. This representation allows a particularly efficient computation of diagnoses through our DRUM–II engine, which has shown excellent performance in popular model-based diagnosis benchmarks [11].

From the technical side, DRUM–II is more efficient than other algorithms because it continuously manipulates the same one interpretation, making copies only of the models found. This avoids the combinatorial explosion of data structures reported for ATMS systems [9]. Furthermore, requiring no sophisticated backtracking or heap maintenance procedures, DRUM–II is easily implemented as a set of portable Java classes, which is a prerequisite for deployment to practice.

A few practical problems remain. First and foremost, the dynamic nature of an open system as reflected by continuous instance creation and deletion is at odds with our requirement that all instances be known prior to the generation of the system description. Rerunning the instantiation phase of the diagnostic process however is costly and entails that all current models (the diagnostic hypotheses) are lost.

Secondly, all observations recorded in the MIB carry time–stamps. Because the logic on which our model–based diagnosis is based is inherently atemporal, this information is ignored. Also, although observations are processed as soon as they arrive, the diagnosis

[5]Among the few works we are aware of that do are the EXACT system for satellite testing [23] and an object-oriented approach to photo copier maintenance [24]. Both seem to take a rather pragmatic approach, but not enough detail is provided to allow a comparison with our work. Console et al. have worked on hierarchical diagnoses modelled by implication [6].

based on all currently available observations is independent of the order in which they were incorporated in the models. Although this loss of information has no consequences for our example, there are situations conceivable in which it may lead to inferior results.

Last but not least, it should be noted that predicate completion (and thus our implementation of propagation) works only for acyclic structures. This however is no real restriction to open systems management, since all cyclic connections must be resolved through a router the formal specification of which renders the system description acyclic, too.

7. Conclusion

The management of open systems requires effective procedures for fault localization. We have shown that the management information model established by the ISO blends well with the logical framework of model–based diagnosis. In fact, by defining an effective mapping from object–oriented system descriptions compliant with ISO standards to compact logical clause sets, we are able to exploit our model repair algorithm DRUM–II to efficiently localize faulty components of open systems in dynamic environments.

8. Acknowledgments

This work was made possible by a DFG grant under the “Schwerpunktprogramm Mobilkommunikation”. We thank our project partners Klaus Jobmann, Hermann Wietgrefe and Klaus–Dieter Tuchs and the people at e.plus for their cooperation.

References

[1] Philippe Besnard and Marie-Odile Cordier. Explanatory diagnoses and their characterization by circumscription. *Annals of Mathematics and Artificial Intelligence*, 11:75–96, 1994.

[2] K. H. Bläsius, U. Hedtstück, and C. R. Rollinger, editors. *Sorts and Types in Artificial Intelligence.* (*Lecture Notes in Computer Science 418*). Springer, 1989.

[3] T. S-C. Chou and M. Winslett. A model–based belief revision system. *Journal of Automated Reasoning*, 12:157–208, 1994.

[4] K. Clark. Negation as failure. In *Logic and Databases*, pages 293–322. Plenum, 1978.

[5] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. On the relationship between abduction and deduction. *Journal of Logic and Computation*, 1(5):661–690, 1991.

[6] Luca Console, Daniele Theseider Dupré, and Pietro Torasso. Towards the integration of different knowledge sources in model-based diagnosis. In E. Ardizzone, S. Gaglio, and F. Sorbello, editors, *Trends in Artificial Intelligence. 2nd Congress of the Italian Association for Artificial Intelligence*, pages 177–186. Springer, 1991.

[7] European Telecommunication Standards Institute. *Digital cellular telecommunications system* (*Phase 2*)*; Base Station System* (*BSS*) *Management Information* (*GSM 12.20*), 1996.

[8] European Telecommunications Standards Institute. *Digital telecommunications system* (*Phase 2+*)*; Fault Management for the Base Station Subsystem* (*BSS*) (*GSM 12.11*), 1997.

[9] P. Fröhlich. *DRUM-II: Efficient Model-based Diagnosis of Technical Systems*. PhD thesis, Uni. Hannover, 1998.

[10] P. Fröhlich and W. Nejdl. Efficient diagnosis based on incomplete system descriptions. In *Workshop on Non-Monotonic Reasoning*, Trento, Italy, 1998.

[11] Peter Fröhlich and Wolfgang Nejdl. A static model-based engine for model-based reasoning. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence* (*IJCAI-97*), Nagoya, Japan, August 1997.

[12] M. R. Genesereth and N. J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1987.

[13] W. Hamscher, L. Console, and J. de Kleer. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[14] ISO/IEC. *Information Processing Systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management Framework*, 1989. also: CCITT Recommendation X.700 (ITU).

[15] ISO/IEC. *Information Technology - Open Systems Interconnection - Structure of Management Information: Management Information Model*, 1993. also: CCITT Recommendation X.720 (ITU).

[16] ISO/IEC. *Information Technology - Open Systems Interconnection - Structure of Management Information: Guidelines for the Definition of Managed Objects*, 1993. also: CCITT Recommendation X.722 (ITU).

[17] ISO/IEC. *Information Technology - Open Systems Interconnection - Systems Management Overview, ISO/IEC 10040*, 1993. also: CCITT Recommendation X.701 (ITU).

[18] Kurt Konolige. Abduction versus closure in causal theories. *Artificial Intelligence*, 53:255–272, 1992.

[19] Richard E. Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artificial Intelligence*, 27(1):97–109, 1985.

[20] W. Lukaszewicz. *Non–monotonic reasoning: formalization of commonsense reasoning*. Ellis Horwood, 1990.

[21] Olivier Raiman. Diagnosis as trial - the alibi principle. In *International Model-Based Diagnosis Workshop*, Paris, July 1989.

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

[23] H. Smith-Meyer, R. Kuke, and K. Tangen. EXACT - model based diagnosis for satellite testing. In *5th Scandinavian Conference on Artificial Intelligence SCAI-95*, pages 453–457, Trondheim, 1995.

[24] Y. Umeda, T. Tomiyama, and H. Yoshikawa. Model based diagnosis using qualitative reasoning. In F. Kimura and A. Rolstadas, editors, *Proc. 3rd Int IFIP Conf. Computer Applications in Production and Engineering*, pages 443–450, Amsterdam, 1989. North-Holland.