MODEL-BASED RECONFIGURATION

MARKUS STUMPTNER AND FRANZ WOTAWA
{mst, wotawa}@dbai.tuwien.ac.at

Institut für Informationssysteme
Abteilung für Datenbanken und Expertensysteme
Technische Universität Wien
Paniglgasse 16, A-1040 Wien, Austria

Abstract.
Knowledge-based configuration is both a successful application domain for AI techniques and and an active research area. An open issue in many practical domains is that of reconfiguration, typically exhibited by legacy systems that are to be extended, upgraded or simply altered. Standard configuration techniques are not necessarily suited to this task. We discuss the use of a diagnosis approach to reconfiguration. We present the differing application and representation requirements, develop a representation that is suitable for expressing the information about the required and configurable functionalities from the diagnosis point of view, present an example and discuss our experiences.

1. Introduction

The automated configuration of technical systems is a successful application area for knowledge-based techniques. Initially, configuration research followed the mainstream of expert system research with the use of rule-based expert systems for configuration purposes (McD82), and rule-based tools advertised for intelligent configuration tasks can still be found on the market. Over time, the desirability of higher level representations more amenable to modelling the salient features of configuration problems was recognized (MF89), and representation were developed with the goal of making configuration knowledge more expressive, modular, and easier to maintain. The result was the creation of configuration tools that, while still generally shaped to deal with particular variants of configuration problems, are applicable to clearly delineated problem families instead of being adapted to one individual problem instance. These representations typically include a component hierarchy and a declarative formalism to describe more complex behavior. The latter can be based on description logics (WWB$^+$93), resources (HJ91), or constraint satisfaction problems (CSP), nowadays usually with extensions such as Dynamic (GW96) or Generative CSPs (SH93; SHF94). Most of these systems use problem-independent search methods, although additional strategic knowledge is usually necessary for effective search.

However, most of these approaches deal primarily with the notion of configurating systems from scratch (i.e., given a set of requirements). In practice, the use of configuration tools inevitably leads to the problem of being able to alter existing configurations to accommodate alterations to the knowledge base, extend or adapt legacy systems, and correct problems caused by errors during the original configuration process. This *reconfiguration* process can, but need not start from a complete requirements specification - it is quite possible that the only information that exists apart from the system that is to be reconfigured is the information about the desired properties which the system does not provide, i.e., why the reconfiguration is supposed to happen in the first place. The goal is then to arrive at a new system that corrects the problem (provides the desired new properties) while diverging as little as possible from the originally configured system.

The purpose of this paper is twofold. First, we present reconfiguration as a problem solving task with requirements different from conventional from-scratch configuration. We then propose an approach for adding improved reconfiguration capabilities to a constraint-based configuration tool. We present a representation for incorporating reconfiguration knowledge in the standard model-based diagnosis approach, give a simple example problem and discuss the results produced by our prototype implementation.

2. Reconfiguration

We use the term reconfiguration to refer to the situation where a configuration problem solver is confronted with an incorrect configuration, but not as an intermediate outcome of its search process (as would be the case immediately before application of a fix in the VT paradigm). There are two basic variations how this can happen:

Altered requirements. The customer notices that the system does not provide a particular desired function. This might be either be due to the requirements of the original configuration being incorrect or incomplete (e.g., through operator error or a misunderstanding between customer and vendor), or because the configuration was altered by the customer

Legacy systems. Some time after purchase, the customer wishes to alter the system's functionality (e.g., simply extend it because the small version he bought is no longer up to handling the increased workload), but while the requirements are sensible, the old system is now inconsistent with the new requirements, either because the knowledge base has changed (e.g., new regulations declare the old configuration unsafe) or because new components are not compatible with the components of the old system.

Given such a setting, the goal is to arrive at a new configuration that will satisfy the altered conditions. From a theoretical point of view, different criteria can be specified to identify preferable solutions (HHS93). From a practical point of view, the issue arises how such solutions can be found. A general control regime based on backtracking (chronological or dependency-directed) will obviously not be able to handle such a situation (there are no earlier states to backtrack to). Conversely, a revision-based approach will always be able to provide a heuristic proposal for such changes, but this will require that the knowledge base contains fixes for such things as user errors, fixes for cross-KB-version incompatibilities (any combination of versions) and so forth. Given that fixes are generally considered rules of thumb that are based on experience, their availability for a ”fresh” version of the knowledge base or arbitrary conflicting input constellations must be doubted. In addition, it should be noted that the purpose of a fix during a ”fresh” configuration process will be necessarily different from a reconfiguration situation, since a configuration that is altered by a fix during the initial configuration process is still purely hypothetical - only after the system produces an output will a real system be constructed. Here, though, the real system exists and as a default one can assume that alterations should be kept to a minimum.

In summary, what is required is a method that will propose alterations to the system in a general manner, based as far as possible on the original knowledge base, without making unreasonable choices. For this purpose we use the mechanisms of model-based diagnosis.

Described verbally, the notion of model-based diagnosis means that given a particular system (in this case the existing configuration) and a description of its behavior, we are looking for a set of components in the system whose misbehavior will explain why the system does not work as expected. For readers familiar with model-based diagnosis, note that several of these terms take on a different meaning in the context of reconfiguration.

First, the goal of diagnosis (identifying the ”misbehavior”) : in the ideal case we are looking for a set of aspects of the configuration whose alteration will yield a consistent configuration. In the general case, where such a diagnosis might refer to incorrectly chosen components, finding such a solution and guaranteeing its correctness would imply solving the whole configuration problem! Clearly, this is not realistic; all we can expect is for the diagnosis process to provide an indication of where to start. In a simple subcase, as shown in our example below where the behavior of the configuration is exclusively in terms of unsuitable parameter choices, providing a full diagnosis is actually possible.

Second, the notion of *misbehavior*. Unlike a malfunctioning hardware system (or a buggy program), where the intended output is distorted by the fault, a configurator does not deal with incorrect behavior. The configuration knowledge

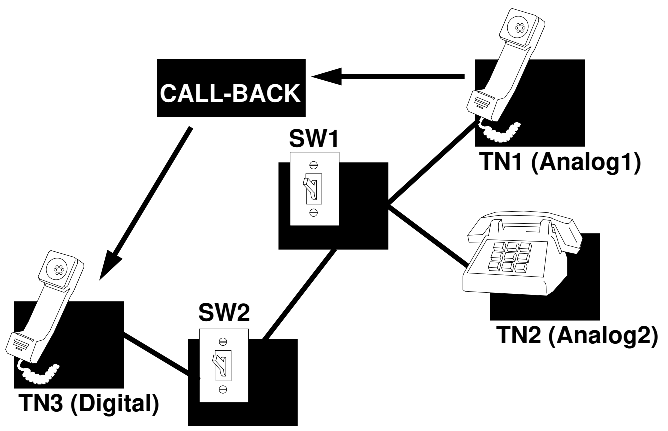

Figure 1. Example Telephone Network

merely specifies what conditions need to be satisfied, and it is the purpose of the configurator to provide the correct components and connections to satisfy these conditions for a given set of requirements. Thus, there is no a priori definition of correct and incorrect states, although we will continue to use these terms to distinguish between the choices reasoned about by the diagnosis system.

Finally, the notion of component changes when we pass the border between the configurator and the diagnostic system. From the point of view of diagnosis, a *component* is any part of the system that provide a certain functionality. If a value choice for a parameter can influence the validity of a configuration, then this parameter needs to be explicitly considered by the diagnosis system as a potential source of the misbehavior; from the point of view of the diagnosis system, it becomes a component.

2.1. AN EXAMPLE

Figure 1 shows a telephone network consisting of three phones and two switches. Two phones $tn1$ and $tn2$ are said to be internal phones since they are connected with the same switch $sw1$. $tn3$ is an external phone for $tn1$ and $tn2$. Several types of telephones exist. We distinguish between one digital and two analog types (analog1 and analog2). For brevity we assume that the basic functionality for every kind of phone is equally implemented.

Features of a telephone includes performing a call to an external phone set and the callback function among others. They are determined by the current parameter setting. For every function available there exists a parameter setting which must be fulfilled. This setting usually is done after installation of the telephone network but some parameters can be changed by the user. Especially settings of their own phone. To reset the phone to the initial state given by the desired features a reconfiguration of the parameter setting must be performed. As inputs the structure of the network, the desired functionality, and a model are given. The model is (verbally) described in handbooks and technical documentation. For example a model of the callback function is given such as follows.

A phone can establish a callback if the parameter `feasu` and `wabe` are set and if there exists a connection from this phone to the called phone with where all involved switches have the `rr` parameter set. For internal calls, the `vk` parameter of the internal switch must be set, too. The called phone set answers `free` if it is not used, and `notfree` otherwise.

Figure 2 depicts a typical view of a small partial configuration as a component-port model, such as used in the constraint-based COCOS configurator (SHF94). Connections between components are expressed via named ports, and parameters of components are named attributes. An object-oriented representation language is used to describe the port and attribute structure of components. The behavior of the components is expressed in terms of constraints which specify valid value assignments for attributes and ports. The callback function would be described as a set of constraints that restricts the involved parameters to valid settings if a setting for the callback parameter is made part of the requirements.

3. A Model for Telephone Networks

Model-based diagnosis (Rei87; dKW87) provides a basic framework for diagnosis. The existence of a description for a system (the model) and observations are assumed. The system itself must be composed out of components with a given behavior. The basic reasoning task in model-based diagnosis is, given a system and observations of that system's behavior, to find a set of components whose incorrect runtime behavior will explain the observations (a more formal definition follows later).

In model-based diagnosis usually the existence of a model based on components and connections is assumed. It consists of a description of the components' behavior and of the structure of the system. For every model for a particular type of component (also called *model fragment* below), the intended application and the required expressiveness will determine the level of abstraction at which the correct and faulty behavior of the component will be modeled. In one application area, the simply distinction between the correct component behavior and any kind of faulty behavior may be strong enough, while in other applications more explicit knowledge about the kind of fault may be necessary. In (SD89), fault models are used to eliminate diagnoses for analog circuits which could not occur in the real world, e.g., that a battery is empty but a connected bulb is still lit. Conversely, (FGN90) shows that in some cases knowledge about physical impossibilities used as a postcondition for diagnosis also eliminates impossible candidates,

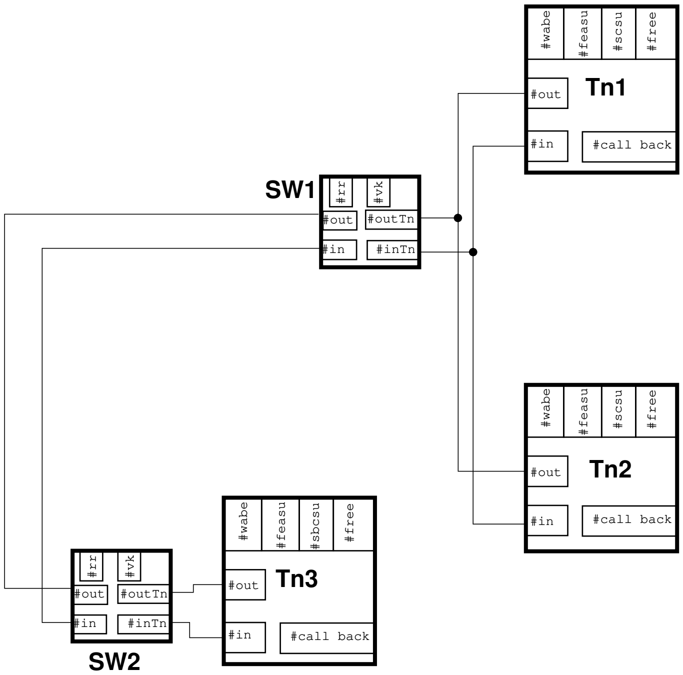

Figure 2. Configuration view of an example telephone network

but with less computational effort.

Note: As will become visible later, the configuration and diagnosis view of components are somewhat different. The diagnosis literature refers as a component to any entity in a system that possesses a separate behavior. In our case, this will extend to such things as parameters, which in the configuration view only exist as properties of the component objects. To make the context clear, we will explicitly use the term *diagnosis component* from here on, even though it is somewhat cumbersome.

Model fragments describe the functionality of a diagnosis component depending on the model's state or mode. A diagnosis component has at least two modes, the correct and the faulty behavior mode. It is possible that several correct behaviors or faulty behaviors exist. For example, a switch can be opened or closed or can be broken or fixed in the closed state. The distinction between the correct behavior (closed) and the faulty behavior (fixed) in the closed state is not always possible, but may be achievable if additional observations are added.

Definition 3.1 (System) *A system is a tuple* $(SD, COMP)$ *consisting of a system description* SD *and a set of diagnosis components* $COMP$.

The system description consists of two parts. The structure gives the interconnection between the diagnosis components and the behavior part specifies the diagnosis component behavior. Every diagnosis component has modes representing the state of the component. Therefore the functionality must be given for every mode.

Definition 3.2 (Diagnosis) *Let* $(SD, COMP)$ *be a system and* OBS *a set of observations for this system. A diagnosis is a function* Δ *associating a mode with every diagnosis component such that* $SD \cup OBS \cup \{\Delta(c) | c \in COMP\}$ *is not contradictory.*

Usually, correctly behaving diagnosis components are excluded when writing down a diagnosis, e.g., a diagnosis $\{\neg ab(c1) \wedge stuck\_at\_1(c2)\}$ is written as $\{stuck\_at\_1(c2)\}$. In the case where several modes for correct behavior exist, we assume that one mode is the default. Default modes are excluded when writing down a diagnosis. The function Γ answers a diagnosis ignoring default mode assignments ($\Gamma(\Delta) = \{\Delta(c) | \Delta(c) \neq DEFAULT\_MODE(c)\}$).

Diagnoses are usually ordered by their cardinality (default mode assignments are ignored). Therefore we define:

Definition 3.3 (Subsumption) *A diagnosis* Δ_1 *subsumes a diagnosis* Δ_2 *iff* $\forall x \in \Gamma(Delta_1) \rightarrow x \in \Gamma(Delta_2)$ *and they are not equal. We write* $\Delta_1 < \Delta_2$.

Definition 3.4 (Minimal Diagnosis) *A diagnosis* Δ *is a minimal diagnosis for* $(SD, COMP, OBS)$ *iff no other diagnosis subsumes* Δ.

Almost every diagnosis algorithm computes minimal diagnoses,. Note that in general not all supersets of minimal diagnoses are diagnoses (dKMR90). In the rest of this section we show how a model can be formulated (and used) for our example.

The model for the telephone network example consists of models for three different kinds of diagnosis components.

- **Parameters** are treated as separate diagnosis components, which allows us to talk about their state. Every parameter has one output and distinguishes between three modes `undef`, `on` and `off`. No fault behavior is given. The `undef` mode is the default mode meaning that the setting is not known or not important. The behavior of parameters is simple. In the `undef` mode the value `undef`, in the `on` the value `on`, and finally in the `off` mode the value `off` is given to the output.
- **Phones** provide specific functionality to the user depending on their parameter value settings. Since parameters are introduced as diagnosis components, a port exists for every parameter. In addition, one port exists for expressing user requests (and the answer returned by the network) and two ports for the input and output of values related to the invoked functionality. Since we are only interested in setting parameters, phones are assumed to always behave

correctly. Therefore, only the correct mode `nab` is used for phones. The behavior of the phone component uses requests (inputs) and parameter settings and associates outputs (answers) with them. A typical function of a phone is establishing a call. This can be described as follows.
Assume a user `a` wants to call user `b`. The call request `call(a,b)` is placed on the user port. The phone component checks its parameters. If they are set correctly, `call(a,b)` is propagated to the output port which is connected to switches or other internal phones. A phone `b` receiving `call(a,b)` checks whether it is free or not. In the free case a `free(b,a)` is sent back as output, otherwise a `notfree(b,a)` is sent back. If `a` receives a `free(b,a)` or `notfree(b,a)` value, it is propagated to the user port.
- **Switches** behave similarly to phones and, just as phones, are also assumed to always behave correctly. Switches have two ports connected to the input and outputs of internal phones and two ports connected to other switches. Again, for every parameter there is a port. Requests such as `call(a,b)` are propagated from the input to both outputs (the internal and the external) only if the parameters are set appropriately. Answers from other phones (or switches) are received also via one of the inputs and propagated to the output.

The example system from Figure 1 can be transformed into a component connection model, as shown in Figure 3. Note again that any part of the configuration that might require alteration is represented as a diagnosis component; in particular, the parameters subject to constraints are represented explicitly since the diagnosis system will reason about their state. The `request` port has been added to the phone components. This property was not present in the configuration model, it is used to introduce external observations (e.g., the assertion that a callback request was deposited, but no returning call was observed).

The verbal description of the call-back behavior of the telephone network can be directly transformed into a model.

- **Parameters:** $parameter(P) \rightarrow$

$$on(SW) \rightarrow out(SW) = on$$

$$off(SW) \rightarrow out(SW) = off$$

$$unknown(SW) \rightarrow out(SW) = unknown$$

- **Phones:** $phone(P) \Rightarrow$

$$callback(P, P') \in request(P) \wedge feasu(P) = on \wedge wabe(P) = on \\ \rightarrow callback(P, P') \in out(P)$$

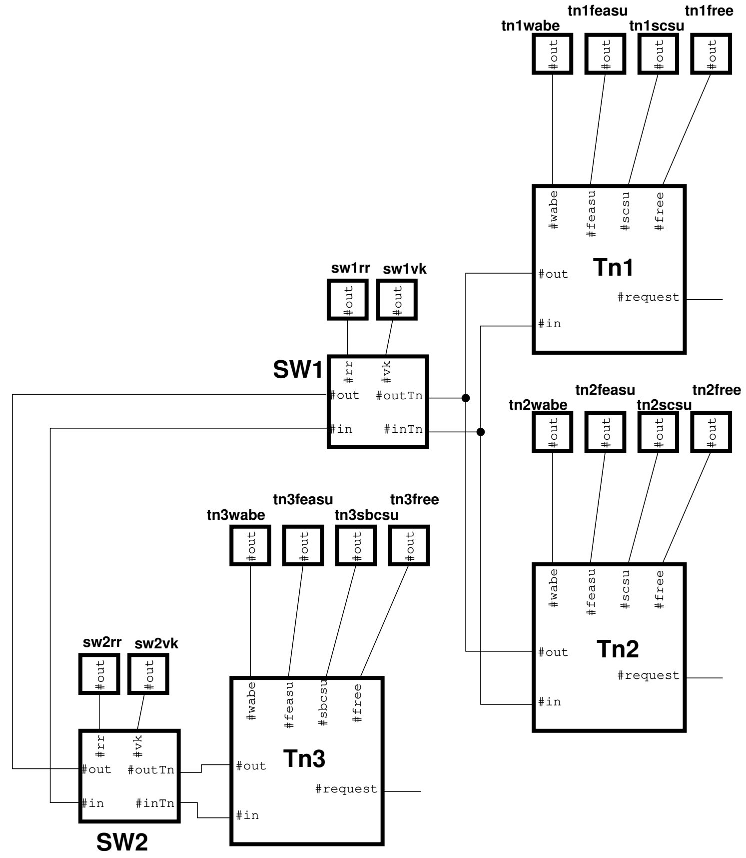

Figure 3. Diagnosis component oriented view of the example network

$$callback(P', P) \in in(P) \wedge scsu(P) = on \wedge free(P) = on$$
$$\rightarrow cb\_free(P, P') \in out(P)$$

$$callback(P', P) \in in(P) \wedge scsu(P) = on \wedge free(P) = off$$
$$\rightarrow cb\_notfree(P, P') \in out(P)$$

$$cb\_free(P', P) \in in(P) \rightarrow cb\_free(P', P) \in request(P)$$

$$cb\_notfree(P', P) \in in(P) \rightarrow cb\_notfree(P', P) \in request(P)$$

Because we want to use the component model for reconfiguration, we are interested in specifying all relevant parameters. Therefore the following rules are introduced:

$$callback(P, P') \in request(P) \wedge feasu(P) \neq false \wedge wabe(P) \neq false \\ \rightarrow feasu(P) = true \wedge wabe(P) = true$$

– **Switches:** $switch(SW) \Rightarrow$

$$callback(P, P') \in in(SW) \wedge rr(SW) = on \wedge vk(SW) = on \\ \rightarrow callback(P, P') \in out(SW)$$

$$callback(P, P') \in in(SW) \wedge rr(SW) = on \\ \rightarrow callback(P, P') \in outtoswitch(SW)$$

$$callback(P, P') \in infromswitch(SW) \wedge rr(SW) = on \\ \rightarrow callback(P, P') \in out(SW)$$

$$callback(P, P') \in infromswitch(SW) \wedge rr(SW) = on \\ \rightarrow callback(P, P') \in outtoswitch(SW)$$

$$cb\_free(P, P') \in in(SW) \rightarrow cb\_free(P, P') \in out(SW)$$

$$cb\_free(P, P') \in in(SW) \rightarrow cb\_free(P, P') \in outtoswitch(SW)$$

$$cb\_free(P, P') \in infromswitch(SW) \rightarrow cb\_free(P, P') \in out(SW)$$

$$cb\_free(P, P') \in infromswitch(SW) \\ \rightarrow cb\_free(P, P') \in outtoswitch(SW)$$

$$cb\_notfree(P, P') \in in(SW) \rightarrow cb\_notfree(P, P') \in out(SW)$$

$$cb\_notfree(P, P') \in in(SW) \rightarrow cb\_notfree(P, P') \in outtoswitch(SW)$$

$$cb\_notfree(P, P') \in infromswitch(SW) \\ \rightarrow cb\_notfree(P, P') \in out(SW)$$

$$cb\_notfree(P, P') \in infromswitch(SW) \\ \rightarrow cb\_notfree(P, P') \in outtoswitch(SW)$$

We explicitly add the following closure axiom:

$$callback(P, P') \in in(SW) \wedge rr(SW) \neq off \\ \rightarrow rr(SW) = on$$

This rule specifies that, if a callback request has been relayed by switch SW, the switch setting required for the relay must have been correct (or the relay action would not have taken place).

Finally, to complete the system description for the telephone network, we have to specify the diagnosis components that will be used (only shown partially for brevity):

$$switch(sw1) \wedge switch(sw2) \wedge phone(tn1) \wedge phone(tn2) \wedge \\ phone(tn3) \wedge parameter(tn1wabe) \wedge \ldots$$

and their interconnections

$$out(tn1wabe) = wabe(tn1) \wedge out(tn1feasu) = feasu(tn1) \wedge \ldots$$

Now we can apply the system description to the phone example. A call back request is stated as observation

$$callback(tn1, tn3) \in request(tn1)$$

We can easily prove that the mode assignment

$$\Delta = \{on(tn1feasu), on(tn1wabe), off(sw1rr)\}$$

is a (minimal) diagnosis wrt. SD and OBS. From $on(tn1feasu)$ and $on(tn1wabe)$ follows that $call(tn1, tn3, callback) \in out(tn1)$. Because $off(sw1rr)$ holds, $out(sw1rr) = off$ follows and nothing more can be derived. Therefore the assumptions about diagnosis component modes in Δ do not contradict the the conjunction of observations and system description, and Δ is a diagnosis. Minimality can be checked by removing an element and checking consistency.

4. Applying the Model to Reconfiguration

Continuing the telephone example, we can compute all minimal diagnoses using the system description and the observation introduced in section 3. This set is given by:

$$\begin{array}{l}
\{off(tn1wabe)\} \\
\{off(tn1feasu)\} \\
\{on(tn1wabe), on(tn1feasu), off(sw1rr)\} \\
\{on(tn1wabe), on(tn1feasu), on(sw1rr), off(sw2rr)\} \\
\{on(tn1wabe), on(tn1feasu), on(sw1rr), on(sw2rr), off(tn3free)\} \\
\{on(tn1wabe), on(tn1feasu), on(sw1rr), on(sw2rr), on(tn3free), off(tn3scsu)\} \\
\{on(tn1wabe), on(tn1feasu), on(sw1rr), on(sw2rr), on(tn3free), on(tn3scsu)\}
\end{array}$$

All these diagnoses represent possible configurations (i.e., in this restricted example, parameter choices) that do not contradict the observations. But most of the diagnoses do not actually establish the desired function (in our case the callback function). To get the correct configuration which ensures the desired functionality, we have to apply a filter condition. For the example, a filter criterion would be that the diagnosis together with the system description and the given observation should allow to derive $cb\_free(tn3, tn1) \in request(tn1)$.

In (WS97), configuration problems are divided into valid, suitable, and optimal configuration. Valid configurations satisfy all constraints, whereas suitable ones are valid and satisfy requirements, too. Optimal constraints satisfy optimality criteria. In our case, minimal diagnoses can be viewed as valid configurations. Those diagnoses which allow us to derive the observations which correspond to the desired function (e.g., a callback arriving at the original phone) are classified as suitable configurations.

Definition 4.1 (Suitable Configuration) *Let F be a (logical) sentence specifying the desired functionality. A suitable configuration Δ for $(SD, COMP, OBS)$ is a minimal diagnosis with the following property*

$$SD \cup OBS \cup \{\Delta(c) | c \in COMP\} \models F$$

A suitable configuration for the callback function is the minimal diagnosis

$$\{on(tn1wabe), on(tn1feasu), on(sw1rr), on(sw2rr), on(tn3free), on(tn3scsu)\}$$

The definition of suitable configurations works like a filter condition on the set of minimal diagnoses, i.e., potential configurations.

An alternative way of achieving the same effect would be to change the system description by the explicit specification of non-derivable behaviors. An example for such a change would be the following rule. If the rr parameter of switch $sw1$ is off, no callback functionality can be achieved:

$callback(P, P') \in in(SW) \wedge rr(SW) = off \Rightarrow \neg cb\_free(P', P) \in out(SW)$

The introduction of these rules would allow to derive only suitable configurations. However, explicitly representing impossible behavior increases the size of the system description, and thus increases the total computation time for diagnosis. Since checking suitability can be done very fast, it should be preferred. This runtime behavior is similar to the advantage of using physical impossibilities (FGN90) compared to the effects caused by introducing fault models (SD89).

In hardware diagnosis, an ordering between diagnoses is often defined in terms of their probabilities, based on the individual fault probabilities of the diagnosis components. In this context, these probabilities of course make no sense, but we expect it would make sense to examine the influence of such factors as

- obsolescence (e.g., how old is a component, or how many revisions behind current technology). This would obviously be relevant in case of a legacy system situation.
- how strongly is the component involved in the configuration (e.g., in terms of influenced constraints, connected components etc.). This is a notion that bears a strong similarity to concepts used in domain-independent CSP optimization.
- how easy was it for the customer (or the configured system's administrator, or any of the system's users, etc.) to alter that parameter, or change that particular diagnosis component? (A terminator plug would be easier to remove than a 5kg switching module.)

These considerations are topics for future research.

5. Background and Implementation

The presented model has been implemented using DTalk, an extension to Smalltalk providing objects and methods for diagnosis. Development of DTalk is part of the Dike project funded by Siemens Austria. The aim of this project is to provide a diagnosis kernel for diagnosis applications in conjunction with the COCOS configurator (SHF94).

DTalk consists of two parts, one for the specification of diagnosis component models, and the other for the structural description. To allow the specification for a wide range of models DTalk provides commands for specifying component ports, modes, fault probabilities for each component mode, information about possible repair steps, and finally for implementing the behavior for each mode. The behavior implementation must be expressed by DTalk methods using the underlying value propagation algorithm which is used to check consistency.

The structure-related part of DTalk includes commands for specifying diagnosis components, connections, and their interconnections using ports. Additionally, information about fault probabilities, repair information and costs, and costs of observation for every connection can be stated. Since informations about specific systems, if present, are assumed to be more important than generic data, they are preferred over the generic information given by DTalk component models.

The example presented in this paper, while somewhat simplified for presentation, corresponds closely to one of the real-world problems that prompted interest in the Dike project. So far, DTalk has been used for describing diagnosis problems in a variety of domains (including the usual digital and analog circuits). It should be noted that the reconfiguration model described here, while requiring certain special features such as set-valued connections, is mainly an extension at the modelling level, not at the level of the underlying diagnosis implementation. The representation has not been used so far to represent systems outside the telecommunications domain, but allows modelling a variety of other typical configuration domains such as computer systems. It should be noted that a major part of the current industrial use of automated configurators does indeed concern itself with computer and telecommunications equipment.

The main requirement for a direct re-use of the modelling approach without extensions is that the behavior of the systems involved must be representable in discrete terms, so that it is possible to map the system's behavior to propagated functionality expressions the way it is done in this paper, e.g., for the `callback` or `cb_notfree` properties. The use of the approach in more general design domains is a topic for future research.

Using Visualworks 2.5 on a Sun Ultra Sparc under Solaris, the valid configurations for the example are computed in about 0.9 seconds. Checking whether a configuration is suitable or not takes 12 ms. Therefore, searching for a suitable configuration for the desired functionality can be done in less than 1 second. Given this particular example feature (callback), the system is reasonably well-behaved in that feature propagation time will increase linearly with the system size. If, however, diagnoses indicate that a particular component (as opposed to a parameter) is at fault, then the system will naturally not be able to "finish" the configuration – it can only provide a pointer which parts of the configuration should be removed, so that the configurator can attempt to continue with the new, consistent, partial configuration.

6. Discussion

6.1. CONTRIBUTIONS AND RELATED WORK

The goal of this paper is to examine the use of diagnosis for the reconfiguration of an existing technical system. The assumptions typically made for a configuration domain include the existence of a fixed set of component types that can be connected via a well-defined set of ports and parameterized over various switches to provide certain functional requirements. It is these assumptions which allow a fairly direct mapping from a configuration representation (as in, e.g., (SHF94)) to the classic system description used in model-based diagnosis (Rei87), and the use of standard model-based diagnosis algorithms.

The example model included in this paper demonstrates the above requirements. It is a typical diagnosis system description insofar as ports in the system are explicit, and the functionality provided by components is exclusively propagated through the ports to other components connected along these ports. The effects caused by the propagation therefore depend entirely on the behavior of the receiving component and a system model can be composed from individual component models.

At the same time, there are two major differences. Reasoning about parameters of the configuration problem requires elevating the parameters themselves to the status of diagnosis components. Furthermore, the complex behavior of the individual parts of the phone example is expressed not by propagating low-level signal values along the connections, but instead high-level predicates that also carry global information (e.g., a `cb_notfree` message, even if it is not ultimately propagated correctly, implies that the connection from caller to called is available). The introduction of set-valued signals allows the use of one port to carry information about multiple propagated functionalities.

The model avoids explicit reasoning about topological properties of the system, since for example the reachability of a component (or the availability of alternate paths through the phone network) is handled by the propagation mechanism. The disadvantage is that if only a small part of the system is actually involved in the problem, propagating to remote parts of the system is both needless and computationally expensive. Earlier versions of the model required explicit information about topological properties and reachability, but might be more efficient for large networks. This is balanced by the fact that for large networks, explicitly describing and writing down all required properties will be a fairly onerous task.

We believe that the restricted nature of the configuration task as compared to a general, possibly innovative, design task, is the significant distinction that allows examining reconfiguration as a separate task independent from the original from-scratch configuration task, whereas systems aiming at more general design problems will generally deal with redesign as part of the original design or design optimization process (see the next sectoin for additional comments on this topic).

While it is certainly theoretically possible to directly use a diagnosis-based reconfigurator during the search for a solution to a from-scratch configuration problem, we are not sure whether the result would provide competitive performance. This is definitely an open avenue for extension.

The concept of using diagnosis for controlling redesign has been previously studied, e.g., in (GC89; GS96; ABBW91). The redesign approach of the KRITIK system family (GC89; GS96) is based on a functional representation. A device is described in terms of its internal and observable behavior. The internal behavior describes the causal processes that determine the output behaviors of the device. Those output behaviors which are intended by the designers (as opposed to potential side effects, desirable or not) are referred to as the *function* of the device. The system allows to represent the behavior of components over time by way of state transition information. Because of the interaction of behavior, function, and their mapping to the structure of the device, the representation is referred to as *Structure-Behavior-Function* (SBF) approach.

Unlike (GS96), we do not distinguish between diagnosis for missing functions or unexpected behaviors. The notion of misbehavior simply corresponds to the existence of an observation that is not consistent with the system description, and the diagnosis is not dependent upon whether the problem is due to missing functionality or spurious outputs. This distinction lies exclusively in the observations provided. The KRITIK systems are also applicable to generic, non-discrete design problems in a wider range of areas such as mechanical engineering which we have not studied yet due to our focus on configuration problems.

Unlike generic flow-based representations for diagnosis (CGTT93), there is no generic categorization of components in our approach, as the behavior of different components towards a particular functionality depends on the individual role of the components. It must be explicitly modelled as part of the system description, since a generic and exhaustive classification of the purpose of parameters, switches, and components in the types of systems we deal with is not forthcoming, as opposed to mechanical or physical domains.

An algorithm for model-based redesign was also introduced in (ABBW91). The paper focuses on parametric and component redesign of technical systems. It is assumed that the specification and model of the old system (which should be redesigned) and the specification of the intended system are available. The proposed diagnostic reasoning method uses a variant of focused Sherlock (dK91). Searching for a redesign is done by diagnosing the old system. Instead of a value propagation mechanism for searching for conflicts under specific assumptions, expression propagation is used. Expressions are computed according to the given component behavior. Since (ABBW91) does only use a subset of all (minimal) conflicts, computed diagnosis candidates can turn out not to be diagnoses. Therefore, the proposed algorithm iterates over the testing of diagnosis candidates and computing of new diagnoses using the obtained test result until a satisfying solution has been found.

The paper distinguishes between *parametric redesign, component redesign*, and *innovative redesign*, but explicitly limit themselves (as we do) to the first two classes.

The approach suffers from a number of restrictions. First, it assumes the existence of a separate specification for both the old and new versions of the system, so that more information is available compared to a situation where only the existing system and the required specification of the new system are given. In particular, the old specification is necessary to generate useful test cases (useful in the sense that they improve diagnosis performance). Second, the approach and algorithm are variants of their standard diagnosis counterparts without providing clear reasons for the variations. Furthermore, the algorithm is based on propagation of symbolic expressions, which leads to problems not only in terms of performance, but also in terms of decidability. It also requires the use of an external equation solver (e.g., MAPLE). The illustration of the approach is limited to the D74 circuit (a simple standard diagnosis example insufficient for judging practical relevance).

6.2. CONFIGURATION AND RECONFIGURATION

The previous section has compared our approach to other work in the general area of redesign. We still want to add some comments on more specific work in the configuration field.

In (WS97), a partial overview of the field of configuration research is given, ending with the question whether it is easier to arrive at a ”good” configuration (by whatever measure) by repairing an incorrect one, than to derive a correct configuration from scratch, i.e., from the initial problem requirements. The authors do not delve into the issue beyond noting that they incline towards the latter, but we believe there are a number of indications that the latter is indeed the case.

First, there is the question of knowledge representation. The remark was made in the context of a discussion of the propose-and-revise problem-solving method (PSM), based on the VT elevator design problem. This well-known problem, first presented in (MSM88), is probably the most widely used individual configuration design example, and its use as a reference example for knowledge acquisition methods (SB96). The VT representation partitions the knowledge base according to different kinds of knowledge used during the configuration process: constraints (for checking validity), formulae (for performing directed computation), and rules which either express conventional inferences – checking relationships between design parameters and deciding on actions to perform – or *fixes* (rules that express heuristic strategy decisions if parts of the design should turn out to be inconsistent).

However, while it provides an effective framework for studying various aspects of configuration problems, the propose-and-revise approach does not appear to be used widely in knowledge-based configuration tools. The newer systems listed above, and many others in the literature, are typically oriented towards configuration ”from scratch”, i.e., given a set of requirements and/or initial components, they will produce a solution under control of a general search algorithm. (The configurator generated by the DIDS system in (RBB94) uses the propose-revise method, but again in the context of the VT example; it is explicitly mentioned that the DIDS approach is not fixed to this particular PSM.) Focusing knowledge representation on the ”fixing” aspects of the problem seems to be relatively rare apart from the VT problem, and it is not quite clear if this is indeed to its full extent a necessary part of the problem and not the result of existing specifications of the problem already being formulated so as to enforce use of the propose-and-revise approach, a weakness that was noted, e.g., in (RGE^{+}96)[1] It should be noted that most of the systems mentioned in the first paragraph employ what are called *weak methods* in (WS97). They still use strategic knowledge

[1]The superficially disagreeing comments in (Yos96) criticize the complete omission of strategic knowledge, but does not discuss the possibility of overconstraining a specification to a particular PSM and thus does not touch the issue.

to improve performance, but the representation separates between problem and strategic knowledge. In this manner, the definition of the problem will focus on the description of a correct solution (e.g., a set of constraints to be satisfied), while the decision what alterations to make if an inconsistency is detected will be up to the control regime.

From the second, computational, point of view, a system based on repairing incorrect configurations is inherently nonmonotonic which again implies a disadvantage when compared to a system that constructs a solution in a monotonic manner.

Nonetheless, while it is hard to judge the effectiveness of the revising approach on problem classes different from the parametric design problem of the VT elevator, there exists a particular situation where this kind of nonmonotonicity cannot be avoided, not because it is necessarily part of the method of inference, but because it is imposed from the outside, namely in the case of *reconfiguration*. That is, the alteration of an existing (typically also already physically constructed and delivered) configuration to fit a changed set of requirements or a changed knowledge base. Note that this means that reconfiguration is a separate problem solving *task* from the "conventional" from-scratch configuration task. It is not a problem solving *method* (PSM) in the way that propose-and-revise is a PSM used in the VT domain for control of the steps internal to a single complete execution of the standard (as opposed to *re-*) design task.

7. Conclusion

In this paper we have presented the use of a model-based approach to reconfiguration, i.e., the configuration of a new system based on an inconsistent old one. Our goal is not to provide an internal control mechanism for the configurator, as is the case with the heuristic fix knowledge of the VT system, or the model-based approach of (BGP94), but rather an adjunct that will provide an indication of the necessary alterations for a configuration that is presented to the configurator by an "outside" event. Note that while the "user" receiving the results may be the configurator who uses the diagnoses as "repair" guidelines – which components to remove when starting to configure the new system that is supposed to arise from the presented, inconsistent one, but the diagnoses in the example, which refer to parameter settings because that is what the constraints refer to, may just as well be used as output given to a maintenance technician who was called by the customer because the system would not provide him with the expected callback function. This represents the second type of reconfiguration – the correction of errors created in the configuration process.

The approach we have presented is quite flexible and can be extended to cover additional functional features of the system without altering the representation of the callback function. If mutually inconsistent functions are required by the user, this case will also be handled correctly by the system (no diagnoses including both functions will be returned and filtering for suitable configurations will return an empty set). Finally, like the configuration specification, the system description is modular, i.e., the functionality of the system as a whole is derived from the composition of the individual components.

Acknowledgement

This work was partly supported by Siemens Austria under the Dike Project (project grant DDV GR 21/96106/4).

References

L.K. Alberts, R.R. Bakker, D. Beekman, and P.M. Wognum. Model-based redesign of technical systems. In *Proceedings on the Second International Workshop on Principles of Diagnosis*, 1991.

S. Bhatta, A. Goel, and S. Prabhakar. Innovation in analogical design: A model-based approach. In *Artificial Intelligence in Design*, pages 57–74. Kluwer, 1994.

Luca Chittaro, Giovanni Guida, Carlo Tasso, and Elio Toppano. Functional and telelogical knowledge in the multimodeling approach for reasoning about physical systems. *IEEE Transactions on Systems, Man and Cybernatics*, 23(3):1718–1751, 1993.

Johan de Kleer. Focusing on probable diagnoses. In *Proceedings AAAI*, pages 842–848, Anaheim, July 1991.

Johan de Kleer, Alan K. Mackworth, and Raymond Reiter. Characterizing diagnoses. In *Proceedings AAAI*, pages 324–330, Boston, August 1990. Morgan Kaufmann Publishers, Inc.

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

Gerhard Friedrich, Georg Gottlob, and Wolfgang Nejdl. Physical impossibility instead of fault models. In *Proceedings AAAI*, pages 331–336, Boston, August 1990. Also appears in Readings in Model-Based Diagnosis (Morgan Kaufmann, 1992).

Ashok Goel and B. Chandrasekaran. Functional representation of designs and redesign problem solving. In *Proc. IJCAI*, pages 1388–1394, Detroit, August 1989.

Ashok Goel and E. Stroulias. Functional device models and model-based diagnosis in adaptive design. *Artificial Intelligence for Engineering, Design, Analysis and Manufacturing (AI EDAM)*, 10:355–370, 1996.

E. Gelle and R. Weigel. Interactive configuration with constraint satisfaction. In *Proceedings of the 2nd International Conference on Practical Applications of Constraint Technology (PACT)*, August 1996.

Alois Haselböck, Thomas Havelka, and Markus Stumptner. Revising inconsistent variable assignments in constraint satisfaction problems. In *Proceedings CSAM'93 Workshop on Constraint Processing*, St. Petersburg, July 1993. Available as Tech. Report DBAI-CSP-TR 93/2.

M. Heinrich and E.W. Jüngst. A resource-based paradigm for the configuring of technical systems from modular components. In *Proceedings of the 7th IEEE Conference on AI Applications (CAIA)*, pages 257–264, 1991.

John McDermott. R1: A rule-based configurer of computer systems. *Artificial Intelligence*, 19:39–88, 1982.

Sanjay Mittal and Felix Frayman. Towards a generic model of configuration tasks. In *Proc.* 11^{th} *IJCAI*, pages 1395–1401. Morgan Kaufmann Publishers, Inc., 1989.

Sandra Marcus, Jeffrey Stout, and John McDermott. VT: An expert elevator designer that uses knowledge-based backtracking. *AI Magazine*, 9(2):95–111, 1988.

J.T. Runkel, A. Balkany, and W. P. Birmingham. Generating non-brittle configuration-design tools. In *Proceedings Artificial Intelligence in Design '94*, pages 183–200. Kluwer, 1994.

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

Thomas Rothenfluh, John Gennari, Henrik Eriksson, Angel Puerta, Samson Tu, and Mark Musen. Reusable ontologies, knowledge-acquisition tools, and performance systems: PROTEGE-II solutions to Sisyphus-2. *Int. J. Human-Computer Studies*, 44:303–332, 1996.

A. Th. Schreiber and W. P. Birmingham. The Sisyphus-VT initiative. *Special issue, Int. J. Human-Computer Studies*, 44, 1996.

Peter Struss and Oskar Dressler. Physical negation — Integrating fault models into the general diagnostic engine. In *Proceedings* 11^{th} *International Joint Conf. on Artificial Intelligence*, pages 1318–1323, Detroit, August 1989.

Markus Stumptner and Alois Haselböck. A generative constraint formalism for configuration problems. In *3rd Congress Italian Assoc. for AI*, volume 729 of *Lecture Notes in AI*, Torino, October 1993. Springer-Verlag.

Markus Stumptner, Alois Haselböck, and Gerhard Friedrich. COCOS - a tool for constraint-based, dynamic configuration. In *Proceedings of the 10th IEEE Conference on AI Applications (CAIA)*, San Antonio, March 1994.

Bob Wielinga and Guus Schreiber. Configuration-design problem solving. *IEEE Expert*, 12(2):49–56, 1997.

Jon R. Wright, Elia S. Weixelbaum, Karen Brown, Gregg T. Vesonder, Stephen R. Palmer, Jay I. Berman, and Harry G. Moore. A Knowledge-Based Configurator that Supports Sales, Engineering, and Manufacturing at AT&T Network Systems. In *Proceedings of the 5th Conference on Innovative Applications of AI*. AAAI Press, 1993.

Gregg Yost. Implementing the Sisyphus-93 task using Soar/TAQL. *Int. J. Human-Computer Studies*, 44:281–301, 1996.