Exploiting structural abstractions for consistency based diagnosis of large configurator knowledge bases

Alexander Felfernig[1], **Gerhard E. Friedrich**[1], **Dietmar Jannach**[1] and **Markus Stumptner**[2]

Abstract. Debugging, validation, and maintenance of configurator knowledge bases are important tasks for the successful deployment of product configuration systems, due to frequent changes (e.g., new component types, new regulations) in the configurable products. Model based diagnosis techniques have shown to be a promising approach to support the test engineer in identifying faulty parts in declarative knowledge bases. Given positive (existing configurations) and negative test cases, explanations for the unexpected behavior of the configuration systems can be calculated using a consistency based approach.

For the case of large and complex knowledge bases, we show how the usage of hierarchical abstractions can reduce the computation times for the explanations and in addition gives the possibility to iteratively and interactively refine diagnoses from abstract to more detailed levels. Starting from a logical definition of configuration and diagnosis of knowledge bases, we show how a basic diagnostic algorithm can be extended to support hierarchical abstractions in the configuration domain. Finally, experimental results from a prototypical implementation using an industrial constraint based configurator library are presented.

1 INTRODUCTION

Knowledge based product configuration systems are a successful application of AI technology and will still gain further importance due to the fact that more and more companies start to offer their products tailored to their specific customer's needs [4]. Many approaches for efficient problem solving and knowledge representation for configuration problems have been proposed, starting from rule-based systems [1] up to higher forms of representation and reasoning techniques, e.g., constraint based, case based, or functional reasoning ([14],[18]). However, the facts that real-world configuration knowledge bases tend to get large and complex and that the knowledge bases are subject to frequent changes (e.g., new component types, new regulations and restrictions) make the validation, maintenance, and debugging activities to important tasks during the lifecycle of the configurator. In fact, especially the maintenance problem was one of the most important problems already in the first industrial configuration systems [1] where changes of up to fourty percent of the knowlege base per year are not unusual.

Techniques from model based diagnosis (MBD) which were initially applied to diagnose faults in electronic circuits and other hardware devices have shown to be also applicable for the domain of software debugging, e.g., diagnosis of logic programs [5], repairing inconsistencies in databases [12], or VHDL programs [10].

In [7], it was shown, how these techniques can also be applied for error detection within knowledge based configuration systems. Speaking in MBD terms, a system is composed from a set of components with some predefined behavior. A diagnosis is then a set of components from the system, which if considered to be faulty, can explain discrepancies between the expected behavior of the system and the actual observations. When employing MBD to debug faulty knowledge bases, the role of the system's components is taken by the elements of the knowledge base (typically logical sentences or constraints). When provided some examples (test runs, observations) the diagnostic task is to identify these faulty parts from the knowledge base which explain an unexpected behavior of the configurator. For these task, *positive* and *negative* examples can be employed. Positive examples are (partial or complete) configurations, which should be accepted by the configurator or can be extended to a complete configuration. These positive examples can be former (supposedly valid) configurations, which should still be valid after some changes in the knowledge base. Negative examples are configurations which should be rejected by the configurator (e.g., because these constellations should not be available anymore). The behavior of the configurator is said to be *unexpected* if a positive example is rejected or if a negative example is falsely accepted.

However, when debugging large and complex knowledge bases, this diagnosis technique can be costly in terms of computation time, especially in cases, when multiple faults of higher cardinality should be detected and when the explanation facilities of the employed inference engine (e.g., a specialized constraint solver) are limited. The usage of *abstraction hierarchies* with different levels of detail has been applied successfully to increase the efficiency of model based diagnosis [16]. In this paper we want to show how these abstraction mechanisms can be employed for the diagnosis of configurator knowledge bases, where we can take advantage of the typical structure of these knowledge bases. Accordingly, no additional or artificial hierarchies must be defined to allow for a hierarchical approach, where we can start the diagnosis with at a high level (with a coarse granularity) which can be calculated very fast, and can then iteratively refine those diagnoses to a more detailed level.

The paper is organized as follows: After giving a motivating example (Section 2) we shortly review the definitions of consistency based diagnosis of configurator knowledge bases from [7] and show how a hierarchical extension can improve the efficiency of the algorithm (Section 3). After presenting experimental results using a simple algorithm with an indus-

[1] Institut für Wirtschaftsinformatik und Anwendungssysteme, University of Klagenfurt, 9020 Klagenfurt, Austria; email: {felfernig,friedrich,jannach}@ifi.uni-klu.ac.at

[2] Institut für Informationssysteme, Abteilung für Datenbanken und Expertensysteme, Paniglgasse 16, A-1040 Wien; email: mst@dbai.tuwien.ac.at

trial strength configurator library (Section 4) we discuss related work and conclusions.

2 MOTIVATING EXAMPLE

For the demonstration of our approach we use a small part of a configuration knowledge base from the domain of configurable personal computers. We employ first order logic as a representation mechanism to provide clear and precise semantics. As a conceptual model, a component-port based approach is chosen [15], the graphical depiction follows the approach from [6].

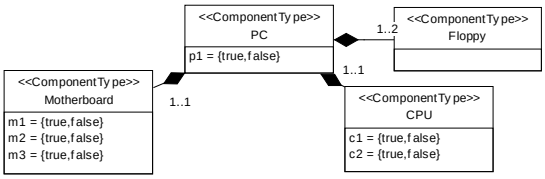

Figure 1 Example problem

We employ the following predicates for our logical representation: *types*, *ports*, *attributes*, and *dom* are functions to describe the component types of the domain with their connection points and attributes with possible values (domain). The facts *type/2*, *conn/4*, and *val/3* describe individual component type instances, their connections and the attribute valuations. We also use a logic programming notation, where variable names start with upper-case letters.

The knowledge base consists of the following definitions:

types={pc,motherboard, cpu, floppy}
ports(pc)={motherboard,cpu,floppy-1,floppy-2}.
ports(motherboard)={pc}.
ports(cpu)={pc}. ports(floppy)={pc}.
attributes(pc)={p1}. attributes(cpu)={c1,c2}. ...
dom(pc,p1)={true,false}. dom(pc,p2)={true,false} ....

In addition, the following constraints on attribute combinations of *PC, Motherboard*, and *CPU* have to hold:

Constraint PC_1: (p1 = m1 ∨ c1)

$type(P,pc) \wedge type(M,motherboard) \wedge type(C,cpu) \wedge conn(P,motherboard,M,pc) \wedge conn(P,cpu,C,pc) \wedge val(P,p1,X) \wedge val(M,m1,Y) \wedge val(C,c1,Z) \Rightarrow X = Y \vee Z.$

Constraint MB_1: (m1 = m2 ∧ m3)

$type(M,motherboard) \wedge val(M,m1,M1) \wedge val(M,m2,M2) \wedge val(M,m3,M3) \Rightarrow M1 = M2 \wedge M3.$

Constraint MB_2: (m2 = m3)

$type(M,motherboard) \wedge val(M,m2,M2) \wedge val(M,m3,M3) \Rightarrow M2 = M3.$

Constraint CPU_1: (c1 = c2)

$type(C,cpu) \wedge val(C,c1,C1) \wedge val(C,c2,C2) \Rightarrow C1 = C2.$

Note that additional constraints may be defined for the component types (e.g., for the *floppy* type), which are omitted in the example because they are not part of any minimal conflict set.

Let us assume, that constraint PC_1 was newly introduced to the knowledge base and contains an error and should be

$$"p1 = \neg (m1 \vee c1)"$$

If we consider the only positive example to be a configuration with connected instances of one *PC*, one *Motherboard*, and one *CPU*, where *p1=false*, *m2=true*, and *c2=true*, more formally,

CONF = {type(pc1,pc). type(cpu1,cpu).
type(mb1,motherboard}.
conn(pc1,motherboard, mb1,pc).
conn(pc1,cpu,cpu1,pc) .val(pc1,p1,false).
val(mb1,m2,true). val(cpu1,c2,true).},

this partial example cannot be completed to a working configuration. The constraints $\{PC_1,MB_1,MB_2,CPU_1\}$ cause a contradiction with the positive examples. When applying the diagnosis algorithm from [7], the following minimal diagnoses are returned expressed using *ab* (abnormal) literals.

$$ab(PC_1).\ \ ab(MB_1) \wedge ab(CPU_1).\ \ ab(MB_2) \wedge ab(CPU_1).$$

In other words, if we consider- for each diagnosis - the individual constraints to be faulty, i.e., abnormal (and ignore it in the search for solutions) a solution to the problem can be found. The resulting diagnoses serve as pointers for the test engineer on which constraints he/she should focus the debugging efforts. (For the treatment of negative examples, see [7]).

For the calculation of diagnoses, the notion of *conflict sets* is used. A *conflict set* can be seen as a set of constraints in our knowledge base which, together with the domain description and an example, causes a contradiction. A conflict set is *minimal*, if no subset of the conflict set is itself a conflict set. Although the diagnostic algorithm does not rely on the calculation of minimal conflict sets, the size of the conflict sets heavily influences the overall efficiency of the task. In cases, where we do not have minimal conflict sets, the diagnostic algorithm tries to find explanations that contain constraints which are definitely not relevant (e.g., some constraints on the floppy component type in our example.) The calculation of minimal conflict sets can also be costly, if the employed inference engine has only limited explanation facilities.

In the following we describe an approach, where we use *structural abstraction*, i.e., the system and its components is decomposed into a hierarchy of its subcomponents. The diagnosis process is started at an abstract level where the model is usually very simple and consists of only a few high level components. The calculation of solutions can be done very efficiently. When moving to the detailed level, the lower level details are taken into consideration, but the detailed diagnosis is done using the results from the previous levels. In technical systems, the hierarchical decomposition is often given through the system's physical structure. In some cases, diagnosis of the subcomponents is not even needed, if the smallest replaceable part is the aggregate component. When diagnosing configuration knowledge bases, the structural abstraction is given through the assignment of the individual constraints to component types. In other words, if we assume a component type definition to be faulty, we assume that all constraint types assigned to that component type are faulty. In the following we only consider such a two level hierarchy, although further abstractions (grouping of several related component types) may be applicable. When using this abstraction mechanism, the top level diagnoses for the problem are

$$ab(PC).\ ab(Motherboard) \wedge ab(CPU).$$

These initial diagnoses can be computed very efficiently since we have only four top level diagnosis components. Given this initial diagnosis the user can decide to focus on the *PC* component type (maybe because he/she only made changes for that component type) or can decide to iteratively refine the results through replacement of the component types with their individual constraints. In the following section we will treat this model more formally.

3 DIAGNOSIS OF CONFIGURATOR KNOWLEDGE BASES

First, we will shortly review the logical definition of configuration and configuration problems from [7]: Typically, configurable products are built from a predefined set of component types, which are characterized through attributes and can be interconnected via predefined connection points (ports). This information and the constraints on legal constellations can be expressed as a set of logical sentences *DD (domain description)*. Configuration problems have to be solved according to some specific requirements, again described through a set of sentences *SRS*. A configuration result is described by a set of ground literals containing information on the employed component instances, attribute valuations, and connections (*e.g., type, val,* and *conn* - literals).

Definition (Configuration Problem): *A configuration problem is described by a triple (DD,SRS,CONL), where DD and SRS are sets of logical sentences and CONL is a set of predicate symbols.*

DD represents the domain description, SRS an individual configuration problem instance. A configuration CONF is described by a set of ground literals whose predicate symbols are in CONL. ❑

Definition (Consistent Configuration): *Given a configuration problem (DD,SRS,CONL), a configuration CONF is consistent iff* $DD \cup SRS \cup CONF$ *is satisfiable.* ❑

To ensure the completeness of a configuration, additional formulae for each predicate symbol in *CONL* have to be introduced to *CONF*, e.g.,

$$type(X,Y) \Rightarrow type(X,Y) \in CONF.$$

We denote the configuration *CONF* extended by these axioms with $\overline{CONF}$. (For a detailed exposition, see [7]).

Definition (Valid and irreducible configuration): *Let (DD,SRS,CONL) be a configuration problem. A configuration CONF is valid iff* $DD \cup SRS \cup \overline{CONF}$ *is satisfiable. CONF is irreducible if there exists no other valid configuration* $CONF^{sub}$ *such that* $CONF^{sub} \subset CONF$. ❑

In MBD terms, a system consists of a set of components and a set of observations describing the actual behavior of the system. The role of the components is played by the elements of *DD*, the observations are given in terms of positive and negative examples.

Definition (CKB-Diagnosis Problem): *A CKB-Diagnosis Problem (Configuration Knowledge Base) is a triple* (DD, E^+, E^-) *where DD is a configuration knowledge base,* E^+ *is a set of positive and* E^- *a set of negative examples. The examples are given as sets of logical sentences. We assume each example on its own to be consistent.* ❑

Positive examples are (partial) configurations, which should be accepted by the configurator, whereas negative examples should be rejected.

Given these example sets and the domain description cause an inconsistency, a diagnosis corresponds to the removal of possibly faulty sentences restoring the consistency. In addition, if a negative example is consistent with the knowledge base, we have to find an extension to *DD* which restores inconsistency for all such negative examples.

Definition (CKB-Diagnosis): *A CKB-Diagnosis for a CKB-Diagnosis Prolem* $(DD, E+, E^-)$ *is a set* $S \subseteq DD$ *such that there exists an extension EX, where EX is a set of logical sentences, such that*

$$DD - S \cup EX \cup e^+ \text{ consistent } \forall e^+ \in E^+$$
$$DD - S \cup EX \cup e^- \text{ in consistent } \forall e^- \in E^- \text{ ❑}$$

Proposition: *Given a CKB-Diagnosis Problem* (DD, E^+, E^-), *a diagnosis S exists iff*

$$\forall e^+ \in E^+ : \cup \bigwedge_{e^- \in E^-} (\neg e^-) \text{ is consistent.}$$

From here on we refer to the conjuction of the negated negative examples as *NE, i.e.,* $NE = \bigwedge_{e^- \in E^-} (\neg e^-)$.

Proof. see [7].

In order to precisely define our approach of using abstraction hierarchies for the diagnosis task, we employ the concept of *Theory Diagnosis* from [9]. In our example, the abstraction of individual constraints from the knowledge base to component types comprising all assigned constraints can be expressed in terms of equivalences, e.g.,

$$ab(MB) \equiv ab(MB_1) \vee ab(MB_2).$$
$$ab(CPU) \equiv ab(CPU_1) \vee \ldots \vee ab(CPU_n).$$

This equivalences express that a component type definition is considered faulty, if at least one of the assigned constraints is faulty. We call this theory of hierarchical abstraction an *ab*-theory Σ.

Definition (Hierarchical Theory): *A hierarchical theory is a set of ab-clauses of the form*

$$ab(c_i) \equiv ab(c_j) \vee \ldots \vee ab(c_k).$$

where all literals contained in such an identity are unique. There are no two equivalences having the same literal on the left hand side. ❑

Note, that such a hierarchical theory defines a set of directed trees. Although in our example, only two level hierarchies are defined, further abstractions are possible, e.g., grouping of several component type definitions to a package, i.e., a more complex substructure of the configured system. For the hierarchical diagnosis task, the domain description *DD* is extended with this abstraction hierarchy Σ.

Let $LHS(\Sigma)$ be the set of literals that appear on the left hand side of a clause from Σ. In the following, given a set $C \subseteq LHS(\Sigma)$, we denote *succ(C)* as the union of the set of all direct and indirect successors and *leaves(C)* be the union of all leaf nodes of all $c_i \in C$ in the tree formed by the hierarchical theory. In the following, *leaf constraints* denote the set of all leaves of all trees and *abstract constraints* be the non-leaf sentences.

For hierarchical diagnosis we extend our notion of *CKB-Diagnosis* in a way that also constraints can appear in the diagnosis which are on the left hand side of a clause from the hierarchical theory.

Definition (Abstract CKB-Diagnosis): *An Abstract CKB-Diagnosis for a configuration problem* (DD,E^+,E^-) *and a hierarchical theory* Σ *is a set of sentences S from DD and a non-empty set C from LHS(*Σ*) such that:*

$DD - S - leaves(C) \cup EX \cup e^+$ *consistent* $\forall e^+ \in E^+$,
$DD - S - leaves(C) \cup EX \cup e^-$ *inconsistent* $\forall e^- \in E^-$ *where*
$S \cap leaves(C) = \emptyset$, $\forall c \in C: c \notin succ(C)$. ❑

Following this definition, such a diagnosis can therefor contain individual constraints from *DD* as well as *abstract* constraints. If an abstract constraint is considered faulty, all its sub-constraints are also considered faulty and must not be contained in *S*. In addition, if an abstract constraint is considered faulty, no abstract subconstraint according to the hierarchy may be contained in *C*.

Definition (Minimal Abstract CKB-Diagnosis): *An Abstract CKB-Diagnosis* $AD = \{S \cup C\}$ *for* (DD, E^+,E^-) *and* Σ *is said to be minimal, if no subset* $AD' \subset AD$ *is an Abstract CKB-Diagnosis.* ❑

Similar to *(Minimal) Theory Diagnoses* [9], these abstract diagnoses can be used as a generator for a set of minimal diagnoses. The abstraction theory Σ describes how the abstract diagnoses have to be extended in order to provide minimal diagnoses.

4 COMPUTING DIAGNOSES

4.1 Computing abstract diagnoses

Given the above definitions we can extend the standard algorithm for consistency-based diagnosis to calculate minimal (abstract) diagnoses. The basic algorithm is extended to fit the needs of our application domain and to support the notion of structural abstraction. In the standard algorithm ([13],[17]), the concept of *conflict sets* is used for focusing purposes:

Definition (Conflict Set): *A conflict set CS for* (DD, E^+,E^-) *is a set of elements from DD such that* $\exists e^+ \in E^+$: $CS \cup e^+ \cup NE$ *is inconsistent.* ❑

In order to support the calculation of minimal abstract diagnoses, we extend the definition of conflict sets to allow a conflict set to contain not only sentences from the knowledge base but also abstract constraints, which are then replaced by their subcomponents:

Definition (Abstract Conflict Set): *An abstract conflict set ACS for* (DD,E^+,E^-) *and a hierarchical theory* Σ *is a set of elements S from DD and a set C from LHS(*Σ*) such that* $\exists e^+ \in E^+$: $S \cup leaves(C) \cup e^+ \cup NE$ *is inconsistent, where* $S \cap leaves(C) = \emptyset$, $\forall c \in C: c \notin succ(C)$. ❑

In the algorithm, the basic HS-DAG algorithm from ([13],[17]) is extended as follows: a node *n* in the tree is labeled by a conflict set *CS(n)*; edges leading away are labeled by elements $s \in CS(n)$. The set of edge labels on the path from the root to a node *n* is referred to as *H(n).* In addition, for each node *n* a set *CE(n)* of consistent positive examples is stored, having in mind that once an example is already consistent it will not become inconsistent after further removal of constraints. Since a node can have multiple direct predecessors [13] - referred to as *preds(n)* - we combine the sets *CE* from all direct precedessors for such a node.

According to the idea of iteratively substantiating abstract diagnoses following the hierarchical structure of the problem, we will initally compute a set of high level diagnosis which can then be refined to a more detailed level. Consequently, the diagnostic algorithm has an additional input parameter beside the problem description and the examples, i.e., an abstract diagnosis that was already calculated on a higher abstraction level. During the calculation of a diagnosis the results from that higher level are reused and the diagnosis task is performed on the next detailed level, i.e., abstract constraints are replaced by the constraints from the next lower level.

Accordingly, given an abstract diagnosis *AD* as input, conflict sets returned by the call to the theorem prover must obey certain restrictions. Given a set *AC* of abstract constraints, let *sons(AC)* be the set of direct successors according to the hierarchical theory Σ.

- If $AD = \emptyset$, only sentences from the top level of the abstraction hierarchy can be contained in the returned abstract conflict set.
- If $AD \neq \emptyset$, only sentences from *sons(AD)*, constraints from the top level of the abstraction hierarchy which are not part of the hierarchy of elements of *AD*, and elements from *AD* itself, which are already at the lowest level of the abstraction theory, can be contained in the returned abstract conflict set.
- The returned (abstract) conflict set cannot contain any elements from the path *H(n)* from the root node of the HS-DAG to the current node.

Algorithm: (schema)
In: (DD,E^+,E^-), Σ, an abstract Diagnosis *AD*
Out: a set of refined diagnoses *RD*

(1) Use the hitting set algorithm to generate a pruned HS-DAG *D* for the collection *F* of abstract conflict sets for $((DD,E^+,E^-), \Sigma)$. Compute the DAG in breadth-first manner in order to generate diagnoses in order of their cardinality.

(a) Every theorem prover call $TP(DD - H(n), E^+ - CE(preds(n)), E^-, \Sigma, AD)$ at a node *n* tests whether there exists an $e^+ \in E^+$ such that there is an inconsistency. In this case an (abstract) conflict set is returned, otherwise it returns ok.

(b) Set *CE(n)* to be the set of examples found to be consistent in the call to *TP* union the examples that were already consistent at the direct precedessors of *n*.

(2) Return {H(n) | n is a node of D labeled by ok

4.2 Computing all minimal diagnoses

As described in [9], the calculated abstract diagnoses can be used to describe all minimal diagnoses and serve as a generator for all these diagnoses. Following the explanations from Section 2, we propose an iterative approach where one starts with a high level diagnosis which can be computed very efficiently and decide afterwards either to stop at the current level and focus on some package of sentences from the knowledge base or to refine the diagnosis to a more concise level. In addition, the hierarchical approach can also be applied if the

user only is interested in the most detailed level, because the hierarchical approach outperforms a flat approach in many cases. The recalculation of diagnoses at the different levels causes additional costs but the resulting HS-DAG's size is much smaller depending on the structure of the knowledge base. In addition, in cases where the employed inference engine lacks sufficient explanation facilities (see experimental results) and the calculation of minimal conflict sets would be costly, the algorithm performs better than a flat approach.

The following algorithm sketch shows how the hierarchical approach can be utilized to calculate all minimal diagnosis. In the algorithm a tree of sets of diagnoses with refinement at each level is generated. For ease of presentation, we use a *list* data structure (*DiagsList*) to hold the individual nodes (diagnoses) of the tree.

Algorithm (sketch)
DiagsList = [];
D = diagnose(DD,E^+,E^-,Σ, $\emptyset$)
/ Compute a set D of initial diagnoses on the most abstract level */*
Append all d $\in$ D to DiagsList;
index = begin of DiagsList;
set $E^+ = E^+ - \{e^+ \in E^+ \mid e^+$ consistent with DD}
/ remove examples which are always consistent with DD */*
***while** not at end of DiagsList*
  set d = diagnosis of current node;
  ***if** d is expandable*
   newD = diagnose(DD,E^+,E^-, Σ,d);
   ***for all** nd $\in$ newD*
    ***if** nd is not already in list;*
     append nd to DiagsList;
    endif
   end for
  move index to next element in DiagsList;
end while

After the calculation of a diagnosis on the most abstract level, for each resulting diagnosis a node is generated and added to the list of nodes to be refined. Remove all positive examples which are definitely consistent with the domain description. A node is *expandable*, if the diagnosis of that contains sentences which are not leaf nodes of the hierarchical theory. A node is expanded by calculating a new set of diagnoses in the context of an abstract diagnosis according to the algorithm in Section 4.1. For each of the refined diagnoses a new node is generated if that diagnosis is not already somewhere in the tree (list). The algorithm ends, if no more node can be further expanded. The algorithm prunes out elements from the tree which were already computed in another part of the tree.

The following example illustrates the algorithm for our simplified example. As a first step, an initial diagnosis on the top level is performed, yielding in the minimal diagnosis *{PC}* and *{MB,CPU}* for the abstract conflict sets *{PC,CPU}* and *{MB,CPU}*. Given the user wants to proceed to the next level, these diagnosis, which are at the top of the tree of diagnoses, are refined calling *diagnose* with the additional parameter of the more abstract diagnosis. The call to *diagnose* with *(CPU, MB)* results in the detailed diagnoses *{MB_1, CPU_1}*, *{MB_2, CPU_1}*, and *{PC}*. Note, that the component *PC* was not refined and in addition is redundant, because it is already in the tree of diagnoses (*DiagList*). The *diagnose* call with *{PC}* returns *{PC1}* and *{MB, CPU}*, where the latter is also redundant. At this stage, the tree of diagnoses cannot be refined anymore, since we already reached the lowest level of detail. As a result, the lowest level diagnoses *{PC_1}*, *{MB_1,CPU_1}*, and *{MB_2, CPU_2}* are returned.

After the computation of all detailed diagnoses, the final tree *(DiagList)* of refined diagnoses is:

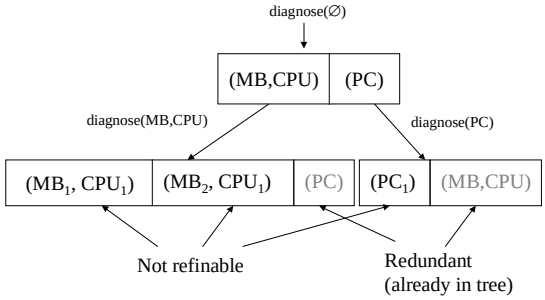

Figure 2 Tree of diagnoses for example problem

5 EXPERIMENTAL RESULTS

In order to test the applicability of our approaches, we implemented a prototype using the industrial strength software library *ILOG Configurator* [14]. Using this package of C++ libraries, a configuration problem is formulated in terms of a *Generative Constraint Satisfaction Problem* (CSP) [8]. This enhancement of the basic CSP mechanism allows the number of variables of the problem to be dynamically changed, i.e., the number of employed components may not be known beforehand. The conceptual model of this library strongly corresponds to the notion of the component port model from Section 3. Both the domain description (types, attributes, and ports), the additional constraints, and the examples can be stated using calls to the library.

In the context of that CSP, a conflict set is a set of constraints from the knowledge base, which, if canceled, makes the configuration problem satisfiable, i.e., a solution can be found. Using this library, the search for an arbitrary solution for a configuration problem can be done very efficient, especially in cases when the problem is underconstrained which is not unusual for configuration problems. However, the employed library does not offer adequate explanation mechanisms which would be helpful for the calculation of (minimal) conflict sets.

We implemented the diagnostic algorithm both for the flat approach from [7] and the extended hierarchical algorithm presented in this paper. The computation time for the diagnostis task depends on several factors such as number of constraint types, cardinality of the diagnoses or the time to test one individual example for consistency. Diagnosis of the simple example problem can be done nearly instantaneously with both algorithms; the identification of two triple faults in a setting with about twenty types of constraints and about hundred constraint instances is done in a few seconds on a standard Pentium-II PC running Windows NT with both algorithms. For these tests we employed our unoptimized prototype which does not calculate minimal conflict sets nor utilizes any special heuristics. However, for larger knowledge bases (containing about hundred types of constraints and component types), the performance of the diagnostic task using the flat approach degrades strongly when calculating diagnoses of higher cardinality. In these cases, the usage of the

hierarchical (interactive) approach with refinement of the diagnoses leverages this problem, if we assume that the given constraints are (uniformly) distributed among the component types. When considering our simple example, the *Floppy* component will only be considered as one single element of the conflict sets and will never be expanded to a more detailed level. With this approach, even larger and more complex knowledge bases remain diagnosable within an acceptable computation time, because additional constraints within the correct parts of the knowledge base only influence the costs of the consistency checks but not those of the diagnostic algorithm. In addition, we conducted experiments with real-world examples from the domain of private telecommunication systems. The tests showed that the results from the diagnostic process are suitable for debugging and validation purposes.

6 RELATED WORK

Model based diagnosis techniques were initially developed for the identification of faults in physical devices, e.g., electronic circuits. Later, these techniques were adopted for diagnosis and debugging of software, e.g., logic programs [5] or relational database consistency constraints [12]. Currently, work is underway to employ MBD techniques to debug other types of software with non-declarative semantics (e.g., hardware designs specified in VHDL [10]). [11] use model based diagnosis to find solutions for overconstrained Constraint Satisfaction Problems, which can also be achieved using our approach through the assignment of weights of importance to the individual constraints. The use of MBD techniques to solve overconstrained CSP's has also been proposed in [1].

The usage of hierarchies for the diagnosis task has been discussed in various application areas of model based diagnosis [16]. Our approach mostly corresponds to what is called *structural abstraction* (vs. *behavioral abstraction*). One of the important problems is to have the information on the hierarchy available at each abstraction level (causing additional modeling effort). For the case of debugging of constraint knowledge bases, however, the hierarchical abstraction has a good correspondence to the configurable artifact, whereas in other domains of (software) debugging, this abstraction may be more complicated to define. Furthermore, we found our logical model of configuration and our configuration language to be expressive enough to cover a wide range of configuration problems. In addition, in [6] framework is defined, where configuration knowledge bases of this type can be automatically generated from high-level conceptual product models.

7 CONCLUSIONS

The demand for AI-based product configuration technology is steadily increasing, due to the growing relevance of mass-customized production and the increasing complexity of the resulting knowledge bases. For the validation and maintenance tasks, only limited support can be found in nowadays systems. For these tasks, we show, how techniques from model based diagnosis can be utilized in supporting the knowledge engineer in validating the knowledge base. Given positive and negative examples (maybe from former configuration runs), the knowledge base is diagnosed and possible explanations for the unexpected behavior are generated. These results can serve as a pointer for the knowledge engineer, which of the constraints may have to be revised to obtain a correct knowledge base. For large and complex knowledge bases, the usage of hierarchical structural abstraction which is given by the problem structure can improve the efficiency of these diagnostic algorithms. We have sketched a basic algorithm which allows for iterative (and interactive) refinement of diagnoses, where further improvements, e.g., through reuse of conflict sets, are part of our future work.

The usage of an industrial strength configurator library alleviates the further integration of AI technology in standard software environments.

8 REFERENCES

[1] R.R. Bakker and F. Dikker and F. Tempelman and P.M. Wognum. Diagnosing and solving over-determined constraint satisfaction problems. Proc: *IJCAI'93*, p. 276–281, Chambery, Morgan Kaufmann, 1993.

[2] Virginia E. Barker and Dennis E. O'Connor. Expert systems for configuration at Digital: XCON and beyond. Comm. ACM, 32 (3): 298-318, 1989

[3] G. W. Bond. Top-down consistency based diagnosis. In Proceedings *DX'96 Workshop*, Val Morin, Canada, 1996

[4] D. Brady, K. Kerwin, D. Welch et al.: Customizing for the masses, Business Week, No. 3673, March 2000.

[5] L. Console, G. Friedrich, and D.T. Dupré: Model-based diagnosis meets error diagnosis in logic programs. In Proc. *IJCAI'93*, Chambery, Morgan Kaufmann, 1993.

[6] A. Felfernig, G. Friedrich, and D. Jannach:, UML as domain specific language for the construction of knowledge based configuration systems, in Proceedings: SEKE'99, 1999.

[7] A. Felfernig, G.Friedrich, D. Jannach, and M. Stumptner, Consistency based diagnosis of configuration knowledge-bases. In Proceedings: ECAI'2000, Berlin, August, 2000.

[8] G. Fleischanderl, G. Friedrich, A. Haselboeck, H. Schreiner and M. Stumptner, Configuring Large Systems Using Generative Constraint Satisfaction, IEEE Intelligent Systems, July/August, 1998.

[9] G. Friedrich, Theory Diagnosis: A Concise Characterization of Faulty Systems, in: Proc. IJCAI'93, Chambery, France,1993.

[10] G. Friedrich, M. Stumptner, and F. Wotawa: Model-Based Diagnosis of Hardware Designs, in Artificial Intelligence, Vol. 111, Num. 2, 1999.

[11] E. Freuder, R. J. Wallace: Partial Constraint Satisfaction, Artificial Intelligence (58), 1992.

[12] M. Gertz, U. Lipeck. A Diagnostic Approach to Repairing Constraint Violations in Databases. In Proceedings *DX'95 Workshop*, Goslar, 1995.

[13] R. Greiner, B.A. Smith, and R.W. Wilkerson: A correction to the algorithm in Reiter's theory of diagnosis. *Artificial Intelligence*, 41(1), 1989.

[14] D. Mailharro: A Classification and Constraint-based Framework for Configuration, AI EDAM, Vol 12 (1998), Cambridge University Press, 1998.

[15] S. Mittal and F. Frayman, Towards a generic model of configuration tasks, in Proceedings: IJCAI'89, 1989.

[16] I. Mozetic: Hierarchical Model Based Diagnosis, in: Harmscher et al.: Readings in Model Based Diagnosis, Morgan Kaufmann, 1992.

[17] R. Reiter: A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1), 1987.

[18] M. Stumptner, An overview of knowledge-based configuration, AI Communications 10(2), pages 111-126, 1997.