Hierarchical diagnosis of large configurator knowledge bases

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

[1]University Klagenfurt, A-9020 Klagenfurt

{felfernig, friedrich, jannach}@ifit.uni-klu.ac.at

[2]Technical University Vienna, A-1040 Wien

mst@dbai.tuwien.ac.at

Abstract

Debugging, validation, and maintenance of configurator knowledge-bases are important tasks for the successful deployment of product configuration systems. Consistency-based diagnosis has shown to be a promising approach for detecting faulty parts in the knowledge bases and explaining unexpected behavior of the configurator, whereby (partial) configurations are used as test cases.

In this paper we show how hierarchical diagnosis can be employed to cope with the complexity of debugging large configurator knowledge bases. A framework for hierarchical diagnosis on different levels of abstraction is presented as well as an algorithm for the calculation of diagnoses for different levels. The presented approach aims at the reuse of existing special purpose configuration systems. We show that the exploitation of hierarchies in such problem domains leads to significant efficiency enhancement thus broadening the applicability of consistency-based diagnosis.

Introduction

Knowledge-based configuration is an important application field of AI technology [19] due to increasing demand for configurable and mass-customized products [4]. The increased complexity and high change rates of the products (and the corresponding knowledge bases) made adequate debugging and testing support a prerequisite for successful deployment of such tools.

Consistency-based diagnosis techniques have shown to be applicable not only for diagnosing electronic circuits or other hardware devices, but also for debugging of software systems like logic programs [5], repairing inconsistencies in databases [10] or VHDL programs [9].

In [6] it was shown how model-based diagnosis (*MBD*) can also be employed for error detection within knowledge bases for configuration systems. A framework is described where "positive" and "negative" examples can be provided, where (partial or complete) "positive" examples should be accepted or completed by the configurator and "negative" ones should be rejected. The behavior of the configurator is unexpected, if a positive example can not be completed or causes a contradiction or an intended negative example is falsely accepted. For localizing possible explanations, the problem is mapped to a diagnosis problem, where the sentences (constraints) of the knowledge base play the part of *components* from the MBD terminology.

The success of AI techniques in the configuration area is based on highly specialized configurators ([7],[13]) able to deal with large configuration problems. In addition such systems use a restricted first-order logical language. Our goal is to extend such specialized systems with diagnosis capabilities for debugging. The work of [16] shows how such an integration of special consistency-checking systems into a general diagnosis problem solver can be achieved. However, the efficiency of the diagnosis computation depends heavily on the minimality of the conflict sets. Typically, specialized configurators offer no generation of conflict sets or provide non-minimal conflicts including many additional elements. One reason for this is that in configuration we have to deal with general clauses and many dependencies between these constraints. Therefore, dependency tracing on top of constraint propagation will not reduce the conflicts significantly in practice.

In this paper we show that hierarchical abstraction ([1],[12],[15],[17]) significantly enhances the efficiency of diagnosing configuration knowledge bases. Typical hierarchical structures in configurator knowledge bases can be employed for diagnosis at different levels of granularity.

The paper is organized as follows. After giving a motivating example, we shortly review the framework of consistency-based diagnosis of configurator knowledge bases. We present the extension of this approach with structural abstractions and show soundness and completeness results for diagnosis at different levels. After the description of an algorithm and results from a prototypical implementation, we discuss related work and present our conclusions.

Motivating Example

For demonstration purposes we use a small fragment for a configuration problem. After inserting a typical failure, we show how consistency-based diagnosis can help to detect this error. We use first order sentences to ensure clear representation with precise semantics. In the example, we have a frame, where cards of different types (CPU's and switching modules) can be inserted on existing named slots. The knowledge-base consists of the following definitions, where *types* describes the available component types and *ports* describes the predefined connection points for the components [14]:

types = {frame, cpu-1, cpu-2, sm-1, sm-2};
ports(frame) = {cp1,cp2,smp1,smp2}.
ports(cpu-1) = {framep}. ports(cpu-2) = {framep}.
ports(sm-1) = {framep}. ports(sm-2) = {framep}.

We use the following predicates for describing configuration knowledge: *type(c,t)* associates a component identifier with one of the predefined types, *conn(c1,p1,c2,p2)* describes that component instance *c1* is connected on port *p1* with another component *c2* on *p2*. Attribute valuations of components are described by a predicate *val/3* which is omitted here (see [6] for an example). In addition, definitions are contained describing which components can be connected via which ports in general and other domain-independent constraints, like that one port can be connected to exactly one other port and connections are symmetric.

In addition, the following constraints for the individual component types have to hold:

"CtF_1: If there is a cpu-1 on cp2, there must be a sm-1 on one of the switching module ports.", more formally

$\forall C,F : type(F,frame) \wedge type(C,cpu\text{-}1) \wedge conn(F,cp2,C,framep) \Rightarrow \exists (S,P): type(S,sm\text{-}1) \wedge conn(F,P,S,framep) \wedge P \in \{smp1, smp2\}.$

For sake of brevity, we only describe the other constraints without formal representation.

"CtS_1: If there is a switching-module sm-2 on smp1 there must be also one sm-2 on smp2."

"CtC_1: If there is a CPU of any type connected to any cpu-port, at least one switching module of type sm-1 or sm-2 must be connected to smp1 or smp2."

"CtC_2: A CPU of type cpu-2 on port cp1 requires switching modules of type sm-2 on both ports smp1, smp2."

"CtC_3:A CPU of type cpu-1 on cp2 requires a CPU of the type cpu-2 on cp1."

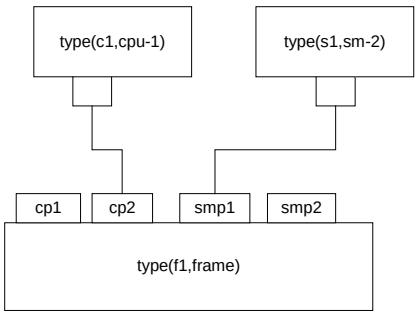

Figure 1 Partial example configuration

Let us assume, that CtF_1 is faulty and too restrictive, because also switching modules of type *sm-2* should be allowed. This situation came about because *sm-2* was a type newly introduced to the knowledge base and CtF_1 was not maintained correctly.

The user provides a positive example with one *cpu-1* and a switching module *sm-2*:

$e^+ = \{\exists F,S,C: type(F,frame) \wedge type(S,sm\text{-}2) \wedge type(C,cpu\text{-}1) \wedge conn(F,cp\text{-}2,C,framep) \wedge conn(F, smp\text{-}1, S, framep)\}.$

Note, that the partial example cannot be completed to a correct configuration (See [6] for the usage of negative examples). Following the consistency-based approach from [6] the minimal conflicts (sets of constraints causing a contradiction with the example)

$\{CtF_1, CtS_1\}$ and $\{CtF_1, CtC_3, CtC_2\}$

induce the minimal diagnoses explaining the unexpected behaviour

$\{ab(CtF_1).\}$
$\{ab(CtS_1). ab(CtC_3).\}\ \{ab(CtS_1). ab(CtC_2).\}$

In other words, if these sets of constraints (diagnoses) are considered abnormal and are canceled, the partial configuration can be completed by the configurator.

Note, that CtC_1 is not contained in any minimal conflict set and that the assumption of having minimal conflict sets available for the HS-DAG generation is very strong for the configuration domain for different reasons: First, there are many interdependencies between the constraints, so a dependency analysis or tracing will not suffice. Second, for the configuration domain, tools (e.g., [12]) with specialized inference mechanisms (*Generative Constraint Satisfaction* [7]) and limited explanation facilities (and costly conflict minimization) are employed for effective calculation of configurations. In addition, the constraint language must be expressive enough for the domain, i.e., the language's constructs exceed the expressiveness of *Horn*-clauses. Consequently, the conflicts returned by the specialized *theorem prover* are large and non-minimal, leading to high search complexity during the HS-DAG generation, because overlaps in the conflict sets minimize benefits from techniques like conflict reuse. One could argue that we should minimize the conflicts before using them during HS-DAG generation. However, note that the computation of additional minimal conflicts is costly, because conflicts with many redundant elements may contain several minimal conflicts, such that minimization requires a number of consistency checks which is not linear in the number of elements of the non-minimal conflict.

Following a hierarchical approach, we will try to analyze the system on a coarse level with smaller complexity, i.e., we do not diagnose individual constraints but whole groups of related constraints. If we group the constraints (indicated by the names in the example) and assume all contained constraints to be

faulty if a group is "abnormal", the minimal diagnoses will be

{ab(frame).} {ab(sm). ab(cpu).}

A faulty constraint is simply not considered for consistency checking, i.e., we don't consider fault models.

Given this result which can be calculated fast having only 3 diagnosable components, the user can decide to use the result as a pointer to faulty group or refine the diagnoses to the next level. When refining one diagnosis to the next level, the contained faulty groups are replaced by their elements; groups not assumed faulty are still treated as one component which reduces the number of diagnosable components.

If we extend the example to have five component types (groups), each containing 10 constraints (diagnosable components *n=50*) and the diagnoses as before, the number of theoretically possible combinations for double-faults is $\binom{50}{2} = 1.225$ which is not reduced significantly when having only large non-minimal conflicts. Using the hierarchical approach, for the top-level diagnosis there are only 10 possible two-element combinations for 5 groups. Refining the first abstract diagnosis with replacement of the faulty group *frame* leads to *n=10+4* diagnosable components and *n=20+3* for the second diagnosis. So, the theoretical upper limit for hierarchical approach for double faults without regarding conflicts is

$$\binom{5}{2} + \binom{14}{2} + \binom{23}{2} = 344.$$

In the following, we will treat the approach more formally.

Diagnosing configurator knowledge bases

In our general framework, a configurator knowledge base consists of a set of logical sentences *DD* describing available component types, their attributes and connection points as well as constraints on legal product constellations [14]. Configuration problems are solved according to specific user requirements *SRS*. A configuration result can be described by means of a set of ground literals containing information on component instances, attribute values and connections. The set of possible literals is contained in a set *CONL*.

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 the user requirements for a 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 onsistent iff DD ∪ SRS ∪ CONF is satisfiable.* ❑
To ensure the completeness of a configuration, additional formulae for each 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 [6]).

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

Definition (CKB-Diagnosis Problem): *A CKB (Configuration Knowledge Base) Diagnosis Problem 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 *EX* to *DD* which restores inconsistency for all such negative examples.

Note that a debugging approach based on test examples is feasible, because positive examples can correspond to previous solutions that should still be accepted using the altered knowledge base. For the formulation of the negative examples, typically only partial configurations will be manually defined by the test engineer. Having altered or added a constraint, the test engineer will provide a focused example in order to test the restrictiveness of the knowledge base, i.e., the test engineer typically has in mind which product constellations should be forbidden by the altered constraint.

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*

$DD - S \cup EX \cup e^+$ *consistent* $\forall e^+ \in E^+$
$DD - S \cup EX \cup e^-$ *inconsistent* $\forall e^- \in E^-$ ❑

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

$\forall e^+ \in E^+ : e^+ \cup \bigwedge_{e^- \in E^-} (\neg e^-)$ *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 [6].

Corollary: *S is a diagnosis iff* $\forall e^+ \in E^+$: *DD-S* $\cup$ e^+ $\cup$ *NE is consistent.*

The following remark relates configuration and diagnosis for configurator knowledge bases [6].

Remark: Let e^+ be partitioned in two disjoint sets e^+_{CONF} and e^+_{SRS} where e^+_{CONF} is a set of positive ground literals whose predicate symbols are in the set of *CONL* and e^+_{SRS} represents system requirements (if some are specified in conjunction with the positive example).

S is a diagnosis (DD, E^+, E^-*) iff* $\forall e^+ \in E^+$: e^+_{CONF} *is a consistent configuration for* $(NE \cup DD - S, e^+_{SRS}, , CONL)$.

Note that if the completeness axioms have been added to e^+_{CONF} then e^+_{CONF} is a valid configuration for $(NE \cup DD - S, e^+_{SRS}, CONL)$.

Note that we defined a very general framework for configuration and diagnosis in order to stay independent from e.g., specific problem solving mechanisms.

Hierarchies in the knowledge base

We will now show how hierarchies in the knowledge base can be used for calculation of diagnoses on the different levels of abstraction. Therefore, we assume that the individual constraints from the knowledge base are arranged into named groups which can be again grouped, such that the structure forms a tree. This hierarchical structure *T* can be expressed using a function *sons(n)*, which returns the direct successors of a node *n* in the tree, i.e., the elements of a named group *n* (and ∅ if *n* is a leaf node). We assume a group *root* to exist in the tree representing the root of the tree. The leaf nodes of the tree are the individual sentences from *DD*. A function *leaves(n)* returns all leaf nodes for a given node *n* which are under *n* (and *n* itself if it is already a leaf node). Finally, all diagnosable constraints from *DD* have to be contained in the tree. Note, that the idea for the following framework is, that we consider all constraints of a group to be potentially faulty, if at least one constraint of the group is faulty.

Definition (Hierarchy tree): A hierarchy tree T for a configuration knowledge base DD is a tree, where

- *the leaf nodes are named elements from DD,*
- *a node "root" represents the root element of the tree,*
- *inner nodes represent named constraint groups from the knowledge base,*
- *the names all leaf nodes and inner nodes appear exactly once in the tree.* ❑

For hierarchical diagnosis we extend our notion of *CKB-Diagnosis* in a way that also constraint group names can appear in the diagnosis. We define a function *successors(n)* to be returning the set of all direct and indirect successors of a node *n* in the tree (and ∅, if *n* is a leaf node). The function *allLeaves(N)* defined on a set of nodes returns the union of *leaves(n)* applied to every $n \in N$.

Definition (Abstract CKB-Diagnosis): *An Abstract CKB-Diagnosis for a configuration problem (DD,*E^+,E^-*) and a hierarchy tree T is a set S of nodes of T such that:*

$DD - allLeaves(S) \cup EX \cup e^+$ *consistent* $\forall e^+ \in E^+$,

$DD - allLeaves(S) \cup EX \cup e^-$ *inconsistent* $\forall e^- \in E^-$ ❑

Definition (Minimal Abstract CKB-Diagnosis): *An Abstract CKB-Diagnosis S for (DD,*E^+,E^-*) and T is said to be minimal, if no subset S'* $\subset$ *S is an Abstract CKB-Diagnosis.*❑

In order to ensure that by using this form of abstraction for different levels no diagnostic information is lost, we have to show that every abstract level diagnosis has a corresponding diagnosis at a more detailed level. Given these definitions, soundness and completeness of the diagnoses at the different levels is ensured.

Soundness of diagnosis: Let *ABSTR* be an Abstract CKB-Diagnosis for a configuration problem *(DD,*E^+,E^-*)* and a hierarchy tree *T* then there exists a *CKB*-Diagnosis *DIAG* such that *DIAG* is a subset of *allLeaves(ABSTR)*. Note, that *allLeaves(ABSTR)* will be a (non-minimal) diagnosis.

Completeness of diagnosis: Let *DIAG* be a *CKB*-Diagnosis for a configuration problem *(DD,*E^+,E^-*)* and a hierarchy tree *T* then there exists an Abstract *CKB*-Diagnosis *ABSTR* such that *DIAG* is a subset of *allLeaves(ABSTR)*. Note, that *{ab(root).}* will be an Abstract Diagnosis. Furthermore, if we construct a set *S'* by replacing any element *s* from *DIAG* with a direct or indirect precedessor of *s*, then *S'* will also be an Abstract Diagnosis.

Computing diagnoses at different levels

Given the above definitions, we can extend the standard hitting-set algorithm for model-based diagnosis to calculate (minimal) diagnoses at the different levels. In the standard algorithm ([11],[16]), *conflict sets* are used for focusing purposes. For the domain of diagnosis of knowledge bases [6], a conflict set is defined as follows:

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 calculation of minimal diagnosis at different levels of abstraction, we extend the definition, s.t. also constraint groups can appear in a conflict set.

Definition (Abstract Conflict Set): *An abstract conflict set for (DD,E^+,E^-) and a hierarchy tree T is a set ACS of elements from T such that*

$\exists e^+ \in E^+$: allLeaves(ACS) $\cup e^+ \cup$ NE is inconsistent. ❑

For the computation of minimal diagnoses for configurator knowledge bases, the basic HS-DAG algorithm from ([11],[16]) is adapted as follows [6]: a node *n* in the DAG is labeled by a conflict set *ACS(n)*; edges leading away are labeled by elements $s \in ACS(n)$. 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 [11] - referred to as *preds(n)* - we combine the sets *CE* from all direct predecessors for such a node.

According to the idea of iteratively substantiating abstract diagnoses following the hierarchical structure of the problem, we will initially 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 (context) beside the problem description and the examples, i.e., an abstract diagnosis that was already calculated on a higher abstraction level. For the calculation of diagnoses on the next level of detail, the constraint groups from the higher-level diagnosis are replaced by their successor nodes according to the hierarchy.

Accordingly, given an abstract diagnosis *AD* as context, the diagnosable components (in terms of model-based diagnosis typically denoted as *COMPS*) for the refined diagnoses are given as follows:

- If $AD = \emptyset$, only elements from *sons(root)* can be contained in the diagnoses.
- If $AD \neq \emptyset$, only elements can be contained in the refined diagnoses, which are from *sons(root)* that are not in *AD*, direct successors from elements from *AD*, and leaf nodes contained in *AD*.

Note, that we have to take *sons(root)* into account for the detail-level diagnosis, because additional groups leading to minimal diagnoses can appear in the detail-level diagnoses, which were hidden by the minimality criterion at the abstract level.

Algorithm: Diagnosis in abstraction context (schema)

In: (*DD*,E^+,E^-), *T*, 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^-), *T*, *AD*). 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^-, T, 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}

Computing all minimal diagnoses

We propose an iterative approach where we start with a high-level diagnosis, which can be computed efficiently. The user can decide to stop at this level and focus on some group(s) of constraints or can refine these results to a more concise level. In the following, an algorithm is presented where a tree with nodes labeled with sets of diagnoses is generated, where at each successor node one of the diagnoses of the parent is refined. First, an initial set of top-level diagnoses (in context *root* of hierarchy tree *T*) is generated. Then, the tree is generated in breadth-first manner, where for each diagnosis of the parent still containing a constraint group, node a child node is generated and diagnosis is performed in the context of that diagnosis. Note that the node is only refined if the considered diagnosis is not already somewhere else in the tree. The algorithm ends, if no more nodes can be refined. Furthermore, if we are only interested in leading diagnoses, the search can be limited, e.g., to a given cardinality.

The usage of the standard diagnosis algorithm guarantees that the computed diagnoses are correct and minimal. Furthermore, the result of every refinement step characterizes the candidate space. These candidate spaces include all minimal diagnoses. It follows, that no minimal diagnosis is excluded during refinement.

Algorithm *(sketch):*
rootnode_diagnoses = diagnose(DD,E^+,E^-, T, ∅)
set $E^+ = E^+ - \{e^+ \in E^+ \mid e^+$ consistent with DD}
*:**label** refine*
refinable = set of diagnoses from current leaf nodes containing constraint groups.
if** refinable = ∅ **goto** :end; **endif
***forall** d ∈ refinable*
calculate diagnosis d' = diagnose(DD,E^+,E^-, T,d)
***if** d' not already in tree*
create child node for d labeled with d'
endif
endfor
***goto** :refine*
*:**label** end*

The following figure shows the tree of diagnoses for the example of Section 2.

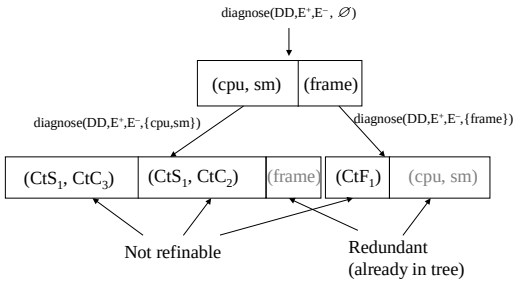

Figure 2 Tree of diagnosis for example problem

Experimental results

In order to test the applicability of our approaches, we implemented a prototype using the industrial strength software library *ILOG Configurator* [13]. Using this package of C++ libraries, a configuration problem is formulated in terms of a *Generative Constraint Satisfaction Problem* (GCSP) [7]. This enhancement of the basic CSP mechanism allows the number of variables of the problem to be dynamically changed, and the number of employed components may not be known beforehand. The domain description with the additional constraints and the examples are expressed 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. Using this library, the search for an arbitrary solution for a configuration problem can be done very efficiently, depending on the problem domain. However, no adequate explanation mechanisms which would be helpful for the calculation of (minimal) conflict sets are provided.

We implemented the diagnostic algorithm both for the flat approach from [6] and the extended hierarchical algorithm presented in this paper. Note, that when using this library, a basic two-level hierarchy, i.e., assignment of constraint types to component types, is already given with the problem formulation and does not cause additional modeling efforts.

The effective computation time for the diagnosis task depends on several factors, e.g., number of constraint types, cardinality of the diagnoses or the time to test one individual example for satisfiability. 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 PC running Windows NT with both algorithms, whereby our unoptimized prototype does not calculate minimal conflict sets nor utilizes domain dependent heuristics. However, for larger knowledge bases (containing about hundred types of constraints and component types and hundreds of constraint instances), the usage of the hierarchical (interactive) approach with refinement of the diagnoses leverages the problem of computational complexity. When considering our simple example from Section 2, components will only be considered as one single diagnosable element and will never be expanded to a more detailed level if they are not in any abstract diagnosis. Therefore, even larger and complex knowledge bases remain diagnosable within an acceptable computation time of a few seconds, 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 process.

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.

Complexity issues

For a simple analysis of the reduction of the computational complexity when using a hierarchical approach, let us consider the search space for diagnosis candidates for both approaches. Given a set of n diagnosable components, i.e., configuration constraints, where we want to find diagnoses of cardinality k, the computational complexity is $O(n^k)$.

If we are given a two-level hierarchy where the n diagnosable components are distributed over g groups, the number of groups determines the computational complexity $O(g^k)$ for the first level.

If we want to refine one of the diagnoses, the constraint groups are replaced with the contained elements from the next level and the average number of elements for two levels will be n/g. Given a number s, $s \leq k \leq g$, describing the number of constraint groups in the abstract diagnosis, the number of diagnosable components for the next level is $(n/g)*s + (g-s)$. The number of remaining constraint groups, which were not in the abstract diagnosis and are therefore not refined, is $(g - s)$. This finally leads to the complexity of $O((n/g)*s + (g-s))^k)$.

The possible achievable benefits from the hierarchical abstraction depend on several factors: First, the number of needed detailization steps depends on the number of existing diagnoses (the upper bound) and the distribution of elements from the detailed diagnoses among constraint groups. In the best case, all detail-level diagnoses are included in one (or a few) abstract diagnosis, whereas in the worst case all detailed diagnoses correspond to different abstract diagnoses. However, it can be reasonably assumed that only small fractions of the knowledge base are faulty after maintenance. Another factor is how the constraints are grouped, i.e., how many constraints for an individual group exist. In the worst case, each group contains one element leading to additional overhead when using the

hierarchies. In good cases, only groups of small size are contained in the abstract diagnoses.

Finally, the assumption that only large, non-minimal conflicts are available following the argumentation of Section 2 (dependencies, expressiveness of configuration language, and specialized inference mechanisms) is an important factor. In cases, minimal conflicts can be easily computed, the hierarchical approach will lead to additional overheads when diagnoses at the detailed levels are needed. Conversely, the more the conflicts contain irrelevant elements, the better the hierarchical approach reduces the complexity. Note, that conflicts are also used for focusing on the abstract level by mapping the detail-level conflicts to the current abstraction level.

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], relational database consistency constraints [10], hardware designs specified in VHDL [9], and overconstrained Constraint Satisfaction Problems [2]. Our work extends the work of [6] by exploiting hierarchies for consistency based diagnosis of configuration knowledge-bases.

The usage of hierarchies for the diagnosis task has been discussed in various application areas of model based diagnosis (e.g., [8],[12],[15],[17]). Our approach mostly corresponds to what is called *structural abstraction* (vs. *behavioral abstraction*) and aims at a more efficient diagnosis process. 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. Changes to the product catalog are usually applied to sets of modules (configuration components) leading to a small set of effectively affected components.

In [1], it was shown that when modeling a system at different levels of abstraction (independently) for general diagnosis problems there may be situations where diagnoses at a detailed level do not have a correspondence to a diagnosis on a more abstract level such that diagnostic information may be lost. This phenomenon cannot appear in our approach, because at each level, the system's "behaviour" (consistency checks) is always analyzed on the most detailed level.

[12] describes hierarchical diagnosis based on value propagation and XDE an extension of the ATMS approach. Our approach is similar in the way structural decomposition is applied. However, our goal was to integrate configuration engines (e.g., based on generative constraint satisfaction) and diagnosis. The approach of [16] offers an appealing way for this integration which was extend by our work in order to employ hierarchies.

[15] uses also hierarchies for improving the efficiency of diagnosis but applies a different notion of diagnosis by defining a diagnosis as a logical consequence of a theory.

Different approaches to diagnosis which avoid the computation of conflict sets were proposed by [3] and [17]. They improve the underlying theorem proving algorithms such that diagnoses can be computed efficiently. Note, that our goal was to reuse specialized problem solvers which are optimized to solve complex configuration problems. The incorporation of these diagnosis techniques without degrading the performance of the configurators remains an interesting open issue.

Conclusions

The demand for AI-based product configuration technology is steadily increasing. For the validation and maintenance tasks, only limited support can be found in nowadays systems. For these tasks, it was shown in [6], how techniques from model based diagnosis can support the knowledge engineer in validating the knowledge base.

Currently, for the configuration of large and complex products, special problem solving mechanisms must be applied. In addition, the domain requires general clauses for expressing configuration constraints. Due to these facts, only limited explanation of conflicts is provided. We showed how the exploitation of hierarchical structures can significantly improve the efficiency of diagnosing configuration knowledge bases. This was achieved by employing specialized configuration systems for consistency checking and extending the approach from [16] in order to cope with hierarchies. We have presented an sound and complete algorithm relying on iterative refinement of diagnoses and validated our approach in a prototype implementation.

References

[1] K. Autio and R. Reiter. Structural abstraction in model-based diagnosis, Proc: ECAI'98, Munich, 1998.

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

[3] P. Baumgartner, P. Fröhlich, U. Furbach and W. Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. Proc: IJCAI'97, Nagoya, August 1997.

[4] D. Brady, K. Kerwin, D. Welch et al. Customizing for the masses, Business Week, No. 3673, 03/2000.

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

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

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

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

[9] G. Friedrich, M. Stumptner, and F. Wotawa. Model-Based Diagnosis of Hardware Designs, Artificial Intelligence (111) 2, 1999.

[10] M. Gertz, U. Lipeck. A Diagnostic Approach to Repairing Constraint Violations in Databases. Proc: DX'95 Workshop, Goslar, 1995.

[11] 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.

[12] W. Harmscher. Modeling Digital Circuits for Troubleshooting, Artificial Intelligence 51(1-3), 1991.

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

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

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

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

[17] P. Struss. What's in SD? Towards a theory of Modeling of Diagnosis. In: Harmscher et al.: Readings in Model Based Diagnosis, Morgan Kaufmann, 1992.

[18] M. Stumptner, F. Wotawa. Diagnosing tree-structured systems, Proc: IJCAI'97, Nagoya, 1997.

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