Iterative focusing for finding faults in large configurator knowledge bases

A. Felfernig[1], G. E. Friedrich[1], D. Jannach[1], M. Stumptner[2], and M. Zanker[1]

[1] Computer Science and Manufacturing Research Group, University of Klagenfurt, Austria
{felfernig, friedrich, jannach, zanker}@ifit.uni-klu.ac.at
[2] Advanced Computing Research Centre, University of South Australia,
SA 5095 Adelaide, Australia
mst@cs.unisa.edu.au

Abstract. Adequate maintenance and debugging support is a key pre-requisite for successful industrial application of AI technology. Consistency-based diagnosis techniques have shown to be a promising approach for detecting faults in declarative knowledge bases in the domain of product configuration. We show how the efficiency of the basic diagnostic reasoning approach that uses (partial) configurations as test cases can be enhanced by exploiting given structures in these knowledge bases and by iteratively focusing on potentially faulty parts of the knowledge base.

1 Introduction

Knowledge-based product configuration has been a prominent application field for declarative knowledge representation for the last decades and with the new sales channels of Electronic Commerce this technology is gaining even more importance due the increasing demand for configurable and customizable products that are marketed over the Web. From the perspective of configuration technology, declarative approaches, e.g., based on Constraint Satisfaction, have shown to be superior to early procedural or rule-based approaches and these techniques are also employed in commercial product configuration systems ([7],[13]). Nonetheless, despite the declarative way of describing the configuration knowledge like the available components and legal product constellations, it shows that the major costs of running a product configurator lie in the validation and maintenance phase; these costs are typically caused by the increased complexity of the configurable products (and the corresponding knowledge bases) as well as by the frequent change rates in the structure and parameters of the products. Moreover, commercial configuration systems are typically optimized for solving the configuration problem itself efficiently but have their limitations when it comes to debugging, i.e., finding the real cause of an unexpected behavior of the configurator.

In the approach described in [5] it was shown how techniques from the field model-based diagnosis can be employed to detect faults in declarative knowledge bases by providing partial configurations as test cases. When using this framework, the knowledge engineer validating a recently changed knowledge base provides *positive* and *negative* test cases, whereby the positive examples should be consistent with a correct knowledge base and negative examples should be rejected by the configurator. Note that the positive examples can correspond to valid configurations from previous configuration runs before the maintenance activities and should still be consistent with the updated knowledge base. In cases where the configurator with an updated knowledge base behaves in an unexpected way by, e.g., rejecting an configuration example that should be consistent according to the intention of the knowledge engineer, model-based diagnosis techniques are used to find diagnoses, i.e., sets of assumedly faulty statements from the knowledge base that explain the discrepancy between the expected behavior and the observed one.

One of the key influencing factors for the diagnostic reasoning task from [5] lies in the size of the *conflict sets* which have to be provided by the *theorem prover* and are used to focus the search process. However, if we want to apply the diagnosis framework in real-world applications and therefore use specialized industrial configurators as core reasoner, we face the fact that their explanation facilities are limited and no conflict generation is available.

In this paper we show how we can utilize given hierarchical abstractions that are typical for configuration knowledge bases to significantly enhance the efficiency of the diagnostic reasoning process: this is achieved by an iterative process, where the knowledge-base is diagnosed at different levels of granularity. After giving a short example and reviewing the basic concepts from [5], we will describe the extensions needed for hierarchical diagnosis and discuss related work in the area.

2 Example

We will use a small but typical configuration example fragment from the telecommunication equipment domain for demonstration purposes, where the problem is to plug cards of different types onto a frame (see [6]).

Using first order logic sentences for knowledge representation [5] the description of the available components is as follows:

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}.

For the constraints and the configuration results we use the predicates *type(c,t)* that associates a component instance with one of the given types and *conn(c1,p2,c2,p2)* for describing connections between components c1 and c2 over the ports p1 and p2, respectively. Note that connections are symmetric and one port can be connected to exactly one other port. In the example, the following constraints 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

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

*"**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."*

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 [5] for the usage of negative examples). Following the consistency-based approach from [5] the minimal conflicts [16] (sets of constraints causing a contradiction with the example)

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

induce the minimal diagnoses for the unexpected behavior:

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

Note, that CtC_1 is not contained in any minimal conflict set and that the assumption of having minimal conflict sets available for HS-DAG generation ([10],[16]) is very strong when using commercial configurators, based on techniques like e.g., Constraint Satisfaction. Unfortunately, the complexity of finding diagnosis without the focusing effect of (minimal) conflict set is high, because overlaps in the conflict sets minimize benefits from techniques like conflict reuse.

In this paper we propose adopting a hierarchical approach where we diagnose the knowledge base first on a coarse level with smaller search complexity by examining whole groups of constraints together. A *natural* grouping for our example is given by the components the constraints relate to (indicated by the constraint names). If we assume all constraints of a group to be faulty if the group is considered faulty, the minimal diagnosis for the example will be

{frame} and *{sm, cpu}*

With only three diagnosable components at the most abstract level (i.e., the constraint groups *frame, sm,* and *cpu*), the search complexity during HS-DAG generation is extremely low. In a next step, the knowledge engineer, can decide to use the result as a pointer to a 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 that were not assumed to be faulty are still treated as one component, which reduces the number of diagnosable components.

3 Consistency-based diagnosis of configuration knowledge bases

This section shortly recapitulates the basic definitions from [5]. In their 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. o

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 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 [5]).

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* $CONF^{sub} \subset CONF$. ❑

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 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 Problem* (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*

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

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

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

$\forall e^+ \in E^+ : e^+ \cup NE$ *is consistent.*

Proof. see [5].

Corollary: *S is a diagnosis iff*

$\forall e^+ \in E^+ : DD - S \cup e^+ \cup NE$ *is consistent.* ❑

4 Hierarchical structures in the knowledge base

In this section we will show how we can impose a hierarchical (tree) structure on the contents, i.e., the individual sentences, of a typical configurator knowledge base. The inner nodes of the tree correspond to named groups of constraints; the leaf nodes are the individual (original) constraints. Such a hierarchical structure T can be expressed using the following functions: *sons(n)* 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, and
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 there exists an extension EX, where EX is a set of logical sentences, 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. Soundness and completeness properties of the hierarchical approach can be found in [12].

5 Diagnosing the knowledge base with abstraction context

Given the above definitions, we now extend the standard hitting-set algorithm for model-based diagnosis to calculate (minimal) diagnoses at the different levels. In the standard algorithm ([[10],[[16]), *conflict sets* are used for focusing purposes. For the domain of diagnosis of knowledge bases [[5], 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, such that 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 HS-DAG algorithm from ([[10], [[16]) is adapted as follows: 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, knowing 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 [[10] - 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 diagnoses, which can then be refined to a more detailed level. Consequently, the diagnostic algorithm has an additional input parameter (context) besides the problem description and the examples, i.e., an abstract diagnosis that was already computed 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* $\neq \emptyset$, we have to take a special set of nodes into account for the next refinement step: a) the leaf nodes from *AD*, and b) for each constraint group in *AD*, we have to compute the path of that element to the root of the hierarchy tree.

Given the set of nodes that are contained in one of these paths, we have to compute the union of all direct successors of these nodes.

Note, that we have to take these direct successors along the abstraction hierarchy into account for the next-level diagnosis, since additional constraint groups leading to minimal diagnoses can appear in the detail-level diagnoses, which were hidden by the minimality criterion at some abstract level. These special cases of hidden diagnoses are explained in more detail in [12].

Algorithm 1: 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 already consistent examples at the direct predecessors of $n$.
(2) Return {H(n) | n is a node of D labeled by ok}.
```

6 Computing all minimal diagnoses

We propose an iterative approach starting with a high-level diagnosis that 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, 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 2: Iterative refinement of diagnoses*(sketch):*

```
rootnode_diagnoses = diagnose($DD,E^+,E^-, T, \emptyset$)
set $E^+ = E^+ - \{e^+ \in E^+ | e^+$ 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
```

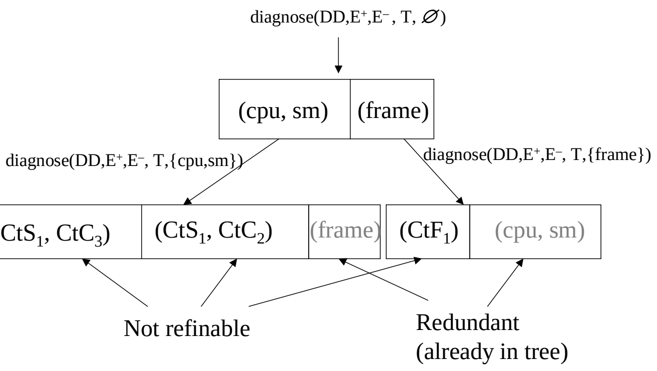

Fig. 1 Tree of diagnoses for example problem

Implementation: A prototype diagnosis system was implemented on top of ILOG's C+ *Configurator* [13] libraries whereby the used library version does not support the generation of (minimal) conflict sets. Our first results showed that the hierarchical approach outperforms the flat approach from [5] in most test cases with larger knowledge bases (with about hundred types of *generic* constraints and component types and hundreds of constraint instances). As a result, even these more complex problems can be solved within an acceptable time frame of a few seconds. For more details on the implementation and an analysis of complexity issues, see [12].

7 Related work

Model-based diagnosis techniques were initially developed for the identification of faults in physical devices, e.g., electronic circuits. Later on, these techniques were adopted for diagnosis and debugging of software, e.g., logic programs [4], hardware designs specified in VHDL [9], and overconstrained Constraint Satisfaction Problems [2]. Our work extends the work of [5] 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],[11],[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 configuration 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 "behavior" (consistency checks) is always analyzed on the most detailed level.

[11] describes hierarchical diagnosis based on value propagation and with 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] also uses 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 [18]. 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 and not to provide conflict sets of explanations. The incorporation of diagnosis techniques from [3] and [18] without degrading the performance of the configurators remains an interesting open issue.

8 Conclusion

The demand for (AI-based) product configuration technology is steadily increasing, but for the validation and maintenance task we can find only limited support in nowadays systems. For these tasks it was shown how techniques from model-based diagnosis can support the knowledge engineer in validating even more complex knowledge bases using an approach that exploits hierarchical structure in the knowledge base itself.

References

[1] K. Autio and R. Reiter. Structural abstraction in model-based diagnosis, Proc: ECAI'98, Brighton, UK, John Wiley and Sons, 1998, pp. 269-273.

[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, pp. 276-281.

[3] P. Baumgartner, P. Fröhlich, U. Furbach and W. Nejdl. Semantically Guided Theorem Proving for Diagnosis Applications. Proc: IJCAI'97, Nagoya, Morgan Kaufmann, 1997, pp. 460-465.

[4] L. Console, G. Friedrich, and D.T. Dupré. Model-based diagnosis meets error diagnosis in logic programs. Proc: IJCAI'93, Chambery, Morgan Kaufmann, 1993, pp. 1494-1501.

[5] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Consistency based diagnosis of configuration knowledge bases. Proc: ECAI'2000, Berlin, IOS Press, 2000, pp. 146-150.

[6] A. Felfernig, G. Friedrich, D. Jannach, and M. Stumptner. Hierarchical Diagnosis of large configurator knowledge bases}. Proc. 24th German / 9th Austrian Conference on Artificial Intelligence (KI-2001), Lecture Notes in Artificial Intelligence 2174, Vienna, Austria, September 2001, pp. 185-197.

[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, Morgan Kaufmann, 1993, pp. 1466-1473.

[9] G. Friedrich, M. Stumptner, and F. Wotawa. Model-Based Diagnosis of Hardware Designs, Artificial Intelligence (111) 2, Elsevier, 1999, pp. 3-39.

[10] R. Greiner, B.A. Smith, R.W. Wilkerson. A correction to the algorithm in Reiter's theory of diagnosis. Artificial Intelligence, 41(1), Elsevier, 1989, pp. 79-88.

[11] W. Harmscher. Modeling Digital Circuits for Troubleshooting, Artificial Intelligence 51(1-3), Elsevier, 1991, pp. 223-271.

[12] D. Jannach. Integration of consistency-based diagnosis and configuration, PhD thesis, University Klagenfurt, 2001.

[13] D. Mailharro. A Classification and Constraint-based Framework for Configuration, AI EDAM, Vol 12(98), Cambridge University Press, 1998.

[14] S. Mittal and F. Frayman. Towards a generic model of configuration tasks, Proc: IJCAI'89, 1989, pp. 1395-1401.

[15] I. Mozetic. Hierarchical Model-based Diagnosis, in: W. Harmscher et al.: Readings in Model-based Diagnosis, Morgan Kaufmann, 1992, pp. 354-372.

[16] R. Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32(1), Elsevier, 1987, pp. 57-95.

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

[18] M. Stumptner, F. Wotawa. Diagnosing tree-structured systems, Artificial Intelligence, 127(1), Elsevier, 2001, pp. 1-29.