Verifying the On-Line Help System of SIEMENS Magnetic Resonance Tomographs using SAT (Extended Abstract)

Carsten Sinz and Wolfgang Küchlin

Symbolic Computation Group, WSI for Computer Science, University of Tübingen and
Steinbeis Technology Transfer Center OIT, 72076 Tübingen, Germany
http://www-sr.informatik.uni-tuebingen.de

Abstract. Large-scale medical systems—like magnetic resonance tomographs—are manufactured with a steadily growing number of product options. Different model lines can be equipped with large numbers of supplementary equipment options like (gradient) coils, amplifiers, magnets or imaging devices. The diversity in service and maintenance procedures, which may be different for each of the many product instances, grows accordingly. Therefore, instead of having one common on-line service handbook for all medical devices, SIEMENS fragments the on-line documentation into small (help) packages, out of which a suitable subset is selected for each individual product instance. Selection of the packages is controlled by XML documents, but the conditions can be translated into Boolean Boolean formulae. To check whether the existing set of help packages is sufficient for all possible devices and service cases, we developed the *HelpChecker* tool. *HelpChecker* uses both SAT- and BDD-based methods to check the consistency and completeness of the on-line documentation and to generate small (counter-)examples for cases where verification conditions are violated.

1 Introduction

There is a persistent trend towards products that are individually adaptable to each customer's needs (*mass customization* [1]). This trend, while offering considerable advantages for the customer, at the same time demands special efforts by the manufacturer, as he now must make arrangements to cope with myriads of different product instances. Questions arising in this respect include: How can such a large set of product variants be represented and maintained concisely and uniquely? How can the parts that are required to manufacture a given product instance be determined? Is a certain requested product variant manufacturable at all? And, last but not least, how can the documentation—both for internal purposes and for the customer—be prepared adequately?

Triggered, among other reasons, by an increased product complexity, SIEMENS Medical Solutions recently introduced a formal description for their magnetic resonance tomographs (MR) based on XML. Thus, not only individual product instances, but also the set of all possible (*valid, correct*) product configurations can now be described by an XML term. This formal *product documentation* allows for an automatic checking of incoming customer orders for compliance with the product specification. Besides checking an individual customer order for correctness, further tests are possible. These may include cross-checks between the set of valid product instances and the parts list (in order to find superfluous parts) or other product attributes (e.g., the product's help documentation).

In this paper we show how a formal semantics can be assigned to the SIEMENS XML representation of their MR systems. This is an indispensable precondition for applying automatic theorem proving methods like SAT-checking. We further show how to translate different consistency properties of the on-line help system (help package overlaps, missing help packages) into propositional formulae. Thus we are able to apply SAT-checkers to find defects in the package assignment of the on-line help system. Situations in which such a defect occurs are computed and simplified using BDD-based abstraction techniques.

2 Product Documentation using XML

Product Structure. Many different formalisms have been proposed in the literature to model the structure of complex products [2–6]. The method used by SIEMENS for the configuration of their MR systems resembles the approach presented by Soininen *et al.* [4]. Structural information is explicitly represented as

a tree. This tree serves two purposes: first, it reflects the hierarchical assembly of the device, i.e. it shows the constituent components of larger (sub-)assemblies; and, second, it gathers all available, functionally equivalent configuration possibilities for a certain functionality. These two distinct purposes are reflected by two different kinds of nodes in the tree, as can be seen from the example in Fig. 1. *Type Nodes* are

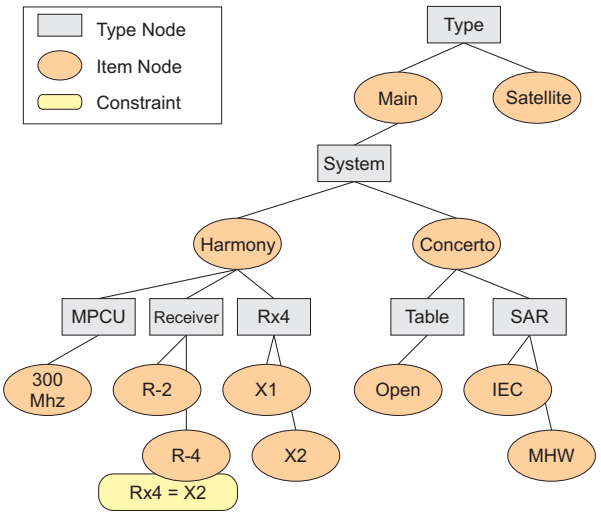

Fig. 1. Product Structure of Magnetic Resonance Tomograps (Simplified Example).

employed to reflect the hierarchical structure, whereas *Item Nodes* mirror possible configuration options with common functionality. From the example tree shown in Fig. 1 we may, e.g., conclude that there are two different possibilities for choosing a *System* from: *Harmony* and *Concerto*. A *Harmony* system possesses three configurable (direct) subcomponents, of type *MPCU*, *Receiver*, and *Rx4*, respectively. The receiver, in turn, may be selected from the two alternatives *R-2* and *R-4*. Choosing the latter option puts an additional restriction on the configurable component *Rx4*: this has to be selected in its form *X2*. Each type node possesses additional attributes *MinOccurs* and *MaxOccurs* to bound the number of subitems of that type to admissible values.

Within the SIEMENS system, the tree describing all product configurations is represented as an XML term. XML terms are checked for well-formedness using XML Schemas.

Structure of On-Line Help. The on-line help pages that are presented to the service personnel of an MR system may depend on the configuration of the system. For example, help pages should only be offered for components that are in fact present in the system configuration. Moreover, for certain service procedures (e.g. tune up, quality assurance), the pages depend not only on the system configuration at hand, but also on the (workflow) steps that the service personnel already has executed. Thus, the help system is both configuration and workflow state dependent.

To avoid writing the complete on-line help from scratch for each possible system configuration and all possible workflow states, the whole help system is broken down into small *Help Packages*. A help package contains documents (texts, pictures, demonstration videos) on a specialized topic. The authors of the help packages decide autonomously about how to break down the whole help into smaller packages. So it is their own decision whether to write a whole bunch of smaller packages—one for each system configuration—or to integrate similar packages into one. Therefore, a list of *dependencies* must be attached to each help package, in which the author lists the system configurations and workflow states for which his package is suitable.

The situations for which a help package must be available are specified by the engineering department using so-called *Help Contexts*. A help context determines system parameters and workflow steps for which a help package must be present. Currently, almost a thousand help contexts are defined for eleven MR systems, each with millions of different configuration possibilities. So, in spite of in-depth product knowledge, it is a difficult and time consuming task for the authors of help packages to find gaps (missing

packages) or overlaps (ambiguities in package assignment) in the help system. To assist the authors, we therefore developed the *HelpChecker* tool, which is able to perform cross-checks between the set of valid system configurations, the situations for which help may be requested (determined by the contexts) and the situations for which help packages are available (determined by the packages' dependencies).

3 Logical Translation of Product Structure and Help System

To check the completeness and consistency of the on-line help system we need a translation into a logical formalism. We have chosen propositional logic for this purpose because of its relative simplicity and the presence of fast and elaborate decision procedures (SAT, BDD). We now lay down precisely what constitutes a consistent help system. Of course, for each situation in which help may be requested there should be a unique suitable help package. Therefore, we have to find out which situations and product configurations can actually occur. In a first step, we therefore develop a formalization of the product structure by building a configuration validity formula (ValidConf). The validity formula can automatically be derived from the XML data of the product structure and consists of consistency criteria for each of the structure's tree nodes. For a type node the following three validity conditions have to hold:

T1. The number of sub-items of the node must match the number restrictions given by the MinOccurs and MaxOccurs attributes.
T2. All selected sub-items must fulfill the validity conditions for item nodes.
T3. No sub-items may be selected that were not explicitly listed as admissible for this type.

For an item node the following three validity conditions have to hold:

I1. All sub-type nodes must fulfill the validity conditions for type nodes.
I2. The item's constraint, if present, has to be fulfilled.
I3. Unreferenced types and their items must not be used in the configuration. Types are considered unreferenced, if they do not appear as a subnode of the item.

We now informally define completeness and consistency of the on-line help system.

Definition 1. *The on-line help system is complete, if for each help context a matching help package exists. Only valid system configurations have to be considered.*

Definition 2. *There is an overlap between two help packages ("ambiguity"), if there is a help context and a valid system configuration for which both help packages match. The on-line help system is consistent, if there are no overlaps between help packages.*

These properties can be translated into propositional criteria to be checked. To build the link between XML terms and propositional logic, sub-elements and attributes from the XML product and help package description are extracted.

Besides the validity formula ValidConf, we define a further formula HelpReq and a set of formulae HelpProv for each help package. HelpReq defines all situations, i.e. configurations and workflows, for which a help package is required, whereas HelpProv(p) determines the situations for which package p provides help. Now, completeness of the help system is equivalent to the validity of the propositional logic formula

$$\text{HelpReq} \wedge \text{ValidConf} \Rightarrow \bigvee_{p \in \text{HelpPackages}} \text{HelpProv}(p) \ . \qquad (*)$$

There is an overlap between help packages p_1 and p_2 if and only if

$$\text{HelpReq} \wedge \text{ValidConf} \Rightarrow \text{HelpProv}(p_1) \wedge \text{HelpProv}(p_2) \qquad (**)$$

is satisfiable. Thus, the help system is consistent, if the latter formula is unsatisfiable for all help packages p_1 and p_2 with $p_1 \neq p_2$.

4 Technical Realization and Experimental Results

Our implementation *HelpChecker* is a C++ program that builds on Apache's Xerces XML parser to read the SIEMENS product and help system descriptions. From these data, it generates Formulae (*) and (**). After having generated these formulae, it checks their satisfiability (in case of (*), it checks satisfiability of the negation). Prior conversion to CNF is done using the well-known technique due to Tseitin [7]. In case of an error condition, a formula is generated describing the set of situations in which this error occurs. This formula is simplified by existential abstraction over irrelevant variables using BDD techniques. *HelpChecker* is embedded into a larger system for the authors of help packages at SIEMENS.

First experiments and timing measurements with the *HelpChecker* were conducted on a data set containing eleven different lines of MR systems, 964 help contexts and twelve (dummy) help packages. To check the completeness and consistency of this data set, 35 SAT instances were generated (we use a fast approximative pre-test for package overlaps that filters out trivial cases). These SAT instances contained 1425 different propositional variables and between 11008 and 11018 clauses. Ten of them were satisfiable, 25 unsatisfiable. One satisfiable instance corresponded to a missing help package, the other nine were due to package overlaps. To check satisfiability we used a sequential version of our parallel SAT-checker PaSAT [8]. Unsatisfiability could always be determined by unit propagation alone, the maximal search time for a satisfiable instance amounted to 15.90 ms (80 branches in the Davis-Putnam algorithm, search heuristics MAX OCC).

5 Conclusion

In this paper we presented an encoding of the configuration and on-line help system of SIEMENS MR devices in propositional logic. Consistency properties of the on-line help system are expressed as Boolean logic formulae and checked by a SAT solver.

Although we demonstrated the feasibility of our method only for the MR systems of SIEMENS Medical Solutions, we suppose that the presented techniques are also usable for other complex products. More generally, we expect that a wide range of cross-checks between XML documents can be computed efficiently using SAT-solvers.

References

1. Davis, S.M.: Future Perfect. Addison-Wesley (1987)
2. Mittal, S., Frayman, F.: Towards a generic model of configuration tasks. In: Proc. of the 11th Intl. Joint Conf. on Artificial Intelligence, Detroit, MI (1989) 1395–1401
3. Sabin, D., Weigel, R.: Product configuration frameworks – a survey. IEEE Intelligent Systems **13** (1998) 42–49
4. Soininen, T., Tiihonen, J., Männistö, T., Sulonen, R.: Towards a general ontology of configuration. AI EDAM **12** (1998) 357–372
5. McGuiness, D., Wright, J.: Conceptual modelling for configuration: A description logic-based approach. AI EDAM **12** (1998) 333–344
6. Küchlin, W., Sinz, C.: Proving consistency assertions for automotive product data management. J. Automated Reasoning **24** (2000) 145–163
7. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In Silenko, A.O., ed.: Studies in Constructive Mathematics and Mathematical Logic. (1970) 115–125
8. Sinz, C., Blochinger, W., Küchlin, W.: PaSAT - parallel SAT-checking with lemma exchange: Implementation and applications. In Kautz, H., Selman, B., eds.: LICS'2001 Workshop on Theory and Applications of Satisfiability Testing (SAT'2001). Volume 9 of Electronic Notes in Discrete Mathematics., Boston, MA, Elsevier Science Publishers (2001)