Computer-Aided Design 48 (2014) 1–16

Contents lists available at ScienceDirect

Computer-Aided Design

journal homepage: www.elsevier.com/locate/cad

Constructing a meta-model for assembly tolerance types with a description logic based approach

Yanru Zhong[a], Yuchu Qin[b], Meifa Huang[c,*], Wenlong Lu[b], Liang Chang[a]

[a] *Guangxi Key Laboratory of Trusted Software, Guilin University of Electronic Technology, Guilin 541004, PR China*

[b] *The State Key Laboratory of Digital Manufacturing Equipment and Technology, School of Mechanical Science and Engineering, Huazhong University of Science and Technology, Wuhan 430074, PR China*

[c] *School of Mechanical and Electrical Engineering, Guilin University of Electronic Technology, Guilin 541004, PR China*

HIGHLIGHTS

- Constraint relations are formalized by assertional axioms.
- Assembly tolerance types are defined with terminological axioms.
- Assembly tolerance types are generated automatically in the meta-model.
- The meta-model has rigorous logic-based semantics.
- The meta-model lays a good foundation for realizing semantic interoperability.

ARTICLE INFO

Article history:
Received 29 April 2013
Accepted 23 October 2013

Keywords:
Meta-model
Assembly tolerance types
Semantic interoperability
Description logics

ABSTRACT

There is a critical requirement for semantic interoperability among heterogeneous computer-aided tolerancing (CAT) systems with the sustainable growing demand of collaborative product design. But current data exchange standard for exchanging tolerance information among these systems can only exchange syntaxes and cannot exchange semantics. Semantic interoperability among heterogeneous CAT systems is difficult to be implemented only with this standard. To address this problem, some meta-models of tolerance information supporting semantic interoperability and an interoperability platform based on these meta-models should be constructed and developed, respectively. This paper mainly focuses on the construction of a meta-model for assembly tolerance types with a description logic $\mathcal{ALC}(\mathcal{D})$ based approach. Description logics, a family of knowledge representation languages for authoring ontologies, are well-known for having rigorous logic-based semantics which supports semantic interoperability. $\mathcal{ALC}(\mathcal{D})$ can provide a formal method to describe the research objects and the relations among them. In this formal method, constraint relations among parts, assembly feature surfaces and geometrical features are defined with some $\mathcal{ALC}(\mathcal{D})$ assertional axioms, and the meta-model of assembly tolerance types is constructed through describing the spatial relations between geometrical features with some $\mathcal{ALC}(\mathcal{D})$ terminological axioms. Besides, $\mathcal{ALC}(\mathcal{D})$ can also provide a highly efficient reasoning algorithm to automatically detect the inconsistency of the knowledge base, a finite set of assertional and terminological axioms. With this reasoning algorithm, assembly tolerance types for each pair of geometrical features are generated automatically through detecting the inconsistencies of the knowledge base. An application example is provided to illustrate the process of generating assembly tolerance types.

1. Introduction

Semantic interoperability refers to the ability of computer systems to transmit data with unambiguous, shared meaning [1]. With the sustainable growing demand of collaborative product design, there is a critical requirement for semantic interoperability among heterogeneous CAT systems [2,3]. To implement semantic interoperability, heterogeneous CAT systems should have the ability to transmit tolerance information with unambiguous, shared meaning. However, CAT systems are developed separately and independently by different vendors. The data storage format and information transmission mode are all different in these systems. So CAT systems must use the standard for the exchange of product model data (STEP) [4] as an intermediary to realize the exchange of tolerance information among them. From the perspective of data exchange, STEP has the following drawbacks [5]: (1) STEP uses

* Corresponding author.
E-mail address: hmhmf@guet.edu.cn (M. Huang).

0010-4485/$ – see front matter Crown Copyright © 2013 Published by Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cad.2013.10.009

neutral files to realize data exchange. Because these neutral files always occupy a large amount of space, it is difficult to implement the real-time data exchange. (2) STEP attempts to use the EXPRESS modeling language to construct consistent semantic models. But because concepts are always expressed with ambiguous meaning in different application domains, it is difficult to construct a general semantic model. (3) STEP provides the mapping mechanisms from EXPRESS language to extensible markup language (XML). The uncontrollable nestifications in XML will lead to extremely complex structure and lacking data constraint of XML documents. Data exchange in STEP still remains on the level of syntax but not on the level of semantics. Thus, tolerance information is difficult to be transmitted unambiguously and semantic interoperability among heterogeneous CAT systems is difficult to be implemented only with STEP [6].

To accomplish semantic interoperability among heterogeneous CAT systems, some meta-models of tolerance information supporting semantic interoperability should be constructed and then an interoperability platform based on these meta-models should be developed. This paper mainly focuses on constructing a meta-model of assembly tolerance types with a description logic $\mathcal{ALC}(\mathcal{D})$ based approach. The meta-models of other tolerance specifications will be constructed and the interoperability platform based on all these constructed meta-models will be developed in the upcoming works.

Description logics, a family of knowledge representation languages for authoring ontologies, are well-known for having rigorous logic-based semantics supporting interoperability. They have been widely applied to the field of industrial information systems to realize semantic interoperability [7]. The main strength of description logics is they provide considerable expressive power going far beyond propositional logic, while reasoning is still decidable. Furthermore, they offer highly efficient reasoning algorithms [8]. As a kind of knowledge representation languages, the intuition of description logics is to define concepts of a domain and use these concepts to specify the properties of individuals occurring in this domain. The primitive symbols of description logics are a set of role names, a set of concept names and a set of individual names. Starting from these symbols, each kind of description logics provides a set of constructors to form complex roles and concepts. Attributive language with complements ($\mathcal{ALC}$) [9] is one of the most influential description logics. It provides negation ($\neg$), conjunction ($\sqcap$), disjunction ($\sqcup$), existential restriction ($\exists$) and value restriction ($\forall$) constructors for construction of concepts. With these constructors, some knowledge on abstract logical level can be described. For example, the concept *Computer-aided-tolerancing* can be specified as follows:

$$\mathit{Computer\text{-}aided\text{-}tolerancing} \equiv \mathit{Tolerance\text{-}design} \sqcap \exists \mathit{use}.\mathit{Computer}.$$

Which represents that computer-aided tolerancing is concerned with all aspects of using a computer to do tolerance design. $\mathcal{ALC}$ not only provides certain expressive power for describing the knowledge on abstract logical level, but also desirable computational properties such as decidability, soundness and completeness of deduction procedures. However, a drawback of $\mathcal{ALC}$ is that all the knowledge has to be defined on abstract logical level. For example, coaxiality tolerance can be defined as a geometric variation caused by two coincident lines. This example refers to the knowledge defined on concrete logical level. $\mathcal{ALC}$ is not sufficient for describing the knowledge in the example.

To meet the requirements of applications in which the knowledge can be defined on a concrete logical level, $\mathcal{ALC}$ is extended by adding a concept-forming predicate constructor and $\mathcal{ALC}(\mathcal{D})$ [10] is therefore achieved. $\mathcal{ALC}(\mathcal{D})$ divides the set of logical objects into two disjoint sets, the abstract and the concrete objects. Abstract objects can be related to concrete objects via attributes. Relations between concrete objects are described with a set of domain specific predicates. Referring to these predicates, properties of abstract objects can also be described through using the concept-forming predicate constructor. In $\mathcal{ALC}(\mathcal{D})$, the pair consisting of a set of concrete objects and a set of predicates defined over these objects is called a concrete domain. With a concrete domain, knowledge in a concrete application domain can be described and reasoned conveniently. So $\mathcal{ALC}(\mathcal{D})$ is quite sufficient to model assembly tolerance types. To make this paper self-contained, a technical background about $\mathcal{ALC}(\mathcal{D})$ is sketched in Section 2.

The rest of the paper is organized as follows. An overview of related works is given in Section 3. Constraint relations in the spatial relations based assembly tolerance representation model [11] are defined with some $\mathcal{ALC}(\mathcal{D})$ assertional axioms in Section 4. Section 5 defines a concrete domain $\mathcal{D}_{\mathcal{AT}}$ and constructs a meta-model of assembly tolerance types with some $\mathcal{ALC}(\mathcal{D}_{\mathcal{AT}})$ terminological axioms. Section 6 provides an application example to illustrate the process of generating assembly tolerance types in the constructed meta-model. Finally, discussions are carried out and conclusions are drawn in Sections 7 and 8, respectively.

2. Preliminaries

In this section, a brief introduction to the concrete domain and the syntax, semantics and reasoning algorithm of description logic $\mathcal{ALC}(\mathcal{D})$ [10] are given.

2.1. Concrete domain $\mathcal{D}$

As mentioned in the introduction, the pair which consists of a set of concrete objects and a set of predicates defined over these objects is called a concrete domain. It can be formally defined as follows:

Definition 1. A concrete domain $\mathcal{D}$ consists of a set $\mathrm{dom}(\mathcal{D})$, the domain of $\mathcal{D}$, and a set $\mathrm{pred}(\mathcal{D})$, the predicate names in $\mathcal{D}$. Each predicate name P is associated with an arity n, and an n-ary predicate $P^{\mathcal{D}} \subseteq \mathrm{dom}(\mathcal{D})^n$.

2.2. Syntax and semantics of $\mathcal{ALC}(\mathcal{D})$

Primitive symbols of description logic $\mathcal{ALC}(\mathcal{D})$ contain: a set of concept names N_C; a set of role names N_R; a set of abstract domain individual names N_{AI}; a set of concrete domain individual names N_{CI}; a set of attribute names N_A; a set of predicate names $\mathrm{pred}(\mathcal{D})$; the constructors $\neg, \sqcap, \sqcup, \exists, \forall$, and $C\exists P$; other symbols, including subsumed symbol $\sqsubseteq$, defined symbol $\equiv$, parenthesis (), colon :, comma , and period ..

Starting with these symbols, the concept terms of $\mathcal{ALC}(\mathcal{D})$ are inductively defined with a set of constructors.

Definition 2. The concept terms of $\mathcal{ALC}(\mathcal{D})$ is generated by the following syntax rule:

$$C, D \rightarrow C_i \mid \neg C \mid C \sqcup D \mid \forall R.C \mid P(u_1, u_2, \ldots, u_n)$$

where $C_i \in N_C$, $R \in (N_R \cup N_A)$, $P \in \mathrm{pred}(\mathcal{D})$, $u_1, u_2, \ldots, u_n \in N_A$.

The expressions of the forms C_i, $\neg C$, $C \sqcup D$, $\forall R.C$ and $P(u_1, u_2, \ldots, u_n)$ are atomic concept, concept negation, concept disjunction, value restriction and predicate restriction, respectively. The concept terms satisfy DeMorgan law. Thus the expressions of the forms *Top* (universal concept), *Bot* (bottom concept), $C \sqcap D$ (concept conjunction) and $\exists R.C$ (existential restriction) are introduced as the abbreviations of the expressions $C \sqcup \neg C$, $\neg Top$, $\neg(\neg C \sqcup \neg D)$ and $\neg(\forall R.\neg C)$.