The Increasing Role of Semantics in Object Modeling

Willem F. Bronsvoort[1], Rafael Bidarra[2], Hilderick A. van der Meiden[3] and Tim Tutenel[4]

[1]Delft University of Technology, W.F.Bronsvoort@tudelft.nl
[2]Delft University of Technology, R.Bidarra@tudelft.nl
[3]Delft University of Technology, RickvanderMeiden@gmail.com
[4]Delft University of Technology, T.Tutenel@tudelft.nl

ABSTRACT

Object modeling for applications like CAD/CAM, simulation and computer games, has traditionally been limited to the shape of objects. Currently, a trend can be observed to add several types of semantics to object models.
This observation is discussed in some detail for three modeling approaches worked on in our research group: feature modeling, including recent advances for freeform features, modeling families of objects, and modeling virtual worlds. The use of semantics in some other modeling approaches is briefly discussed.
Adding semantics can ease specification and modification of an object model, help to guarantee its validity, and be useful for applications which use the model, such as process planning for manufacturing or gameplay. Much work remains to be done to determine the types of semantics most useful in practice.

Keywords: object modeling, constraints, feature modeling, families of objects.
DOI: 10.3722/cadaps.2010.431-440

1 INTRODUCTION

In many applications, including computer-aided design and manufacturing, computer simulations, and computer games, the geometry of objects plays a predominant role. The geometry being considered nowadays can be very complex. An object can consist of many parts, possibly arranged in some pattern, and the parts can have freeform geometry. In virtual worlds, many objects can be included.

Traditionally, only the geometry of an object is represented in its object model, typically by mathematical equations of its bounding surfaces. Such equations do not contain any semantics of the object. Semantics represents certain invariant properties of the object, typically by annotations or constraints. It can be very helpful in building object models, but also in, for example, manufacturing planning and gameplay. Although the designer or engineer who specifies an object model often has certain semantics in mind, there is no way to store it in geometric representations. Currently, however, a trend can be observed to add semantics to object models. The major product modeling approach in which this is done is feature modeling, but the type of semantics included in feature models is very diverse. In particular the use of semantics in freeform feature modeling is a new development. In other modeling approaches, such as modeling families of objects and modeling virtual worlds, the role of semantics is also increasing.

In this paper, the role of semantics is discussed in some detail for the modeling approaches mentioned above. This discussion is based on research performed in our group. Some other approaches in which semantics is important are only briefly discussed. The overview we thus give of the role of semantics in modeling is far from complete. For example, semantics already plays a role in current commercial feature modeling systems and game development tools too, although usually in a less advanced way than discussed here. The goal of this paper, however, is not to give a complete survey of the role of semantics in object modeling, but rather to show that semantics can be very helpful for initial specification and subsequent modification of models, for guaranteeing the validity of models, and for applications making use of object models.

Section 2 gives some background information on modeling and semantics. Sections 3 to 5 describe the role of semantics in the modeling approaches worked on in our research group: feature modeling, modeling families of objects, and modeling virtual worlds. Section 6 shortly indicates this role for some other modeling approaches not worked on in our research group. Section 7 enumerates some conclusions and future developments.

2 BACKGROUND

In traditional object modeling, only information about the geometry of an object is stored, mainly in the form of mathematical equations of surfaces. A distinction can be made between surface modeling and solid modeling.

In surface modeling, only the geometry of surfaces is represented, by mathematical equations, nowadays usually Non-Uniform Rational B-Splines (NURBS) [14]. NURBS offer much modeling freedom, i.e. surfaces modeled with them can be arbitrarily curved or freeformed. Surface modeling has many applications in, for example, styling and aesthetic design.

In solid modeling, the geometry of solid objects is represented by a boundary representation or by a constructive solid geometry representation. A boundary representation consists of an enumeration of the faces, edges and vertices bounding the object; the faces and edges in turn are described by their mathematical equation, and the vertices by their coordinates. The relations between the faces, edges and vertices are stored in a graph structure; this aspect of the model is called its topology. A constructive solid geometry representation consists of a number of relatively simple primitive objects combined by set operations; the primitives are again represented by mathematical equations. Traditionally, in solid modeling only a set of relatively simple surfaces, in particular algebraic surfaces, could be used to represent faces of an object, but nowadays surfaces like NURBS are also available to represent freeform faces of an object. Solid modeling has many applications in, for example, mechanical engineering.

In many applications, e.g. modeling virtual worlds, surfaces and faces of objects are represented by an approximating mesh with small planar faces, in particular triangles.

The above representations are purely geometric, and contain no information on why certain geometry is there in an object model or how it can be used. Stated differently, there is no semantic information in the model. However, as mentioned in Section 1, there is a trend in object modeling to add more semantics to models. Semantic information represents certain invariant properties of an object, and is useful for many purposes. For example, it can help in creating only valid object models, and in using object models for engineering applications and for changing virtual worlds during gameplay. Semantics typically occurs in two types: annotations and constraints.

Annotations are concerned with descriptive semantics, i.e. semantic information is expressed in, usually short, descriptions or attributes attached to aspects of an object model. Such semantics can be helpful in, for example, interpretation of a model by another engineer and in searching a database or the internet for objects, or parts of an object, with particular semantics; the latter is shortly illustrated in Section 6.

Descriptive semantics is, however, less suitable to automatically check object models for validity and generate object models. For these purposes well-defined, or formal, semantics is needed. Such semantics is usually expressed with constraints, or relations, between certain entities in an object model [3]. Examples of this are that two side faces of a slot in a mechanical part should be parallel, and, at a much higher modeling level, that two buildings should be adjacent to a road and next to each other. Many more examples of such constraints are given in Sections 3 to 5.

To determine the shape of an object specified with constraints, some types of constraints need to be solved. Several constraint solving algorithms are available [5]. Typically, these compute configurations of the geometry that satisfy all the constraints. Constraints that specify properties such as distances, perpendicularity, parallelism and coplanarity of faces, can be automatically handled in this way. The solver can either compute one configuration of the geometry, or indicate that no configuration or more than one configuration of the geometry exist for the specified constraints. A configuration can, for example, be the set of all positioned faces of a mechanical object, or the set of all positioned buildings in a virtual world. After the constraint solving, the geometry can be generated. Other types of constraints, e.g. limitations on the volume of an object, cannot be solved, but can only be checked after the geometry has been generated. If such a constraint turns out to be violated, the model has to be adapted by the user of the modeling system.

Specifying an object model by a set of constraints comes down to a declarative way of modeling: only properties of the object model have to be specified, not how the geometry has to be generated [10]. The counterpart of declarative modeling is procedural modeling, in which in fact a procedure to generate the geometry has to be specified. In practice, many modeling systems use a combination of declarative and procedural techniques. As semantics becomes more important, declarative modeling also becomes more important.

In the next sections, it will be shown what the role of semantics is, and how it is handled, in several modeling approaches.

3 FEATURE MODELING

Feature modeling is nowadays the predominant approach for product modeling [16],[4]. It enables the addition of semantics to the geometric information in a product model. A feature model consists of a set of features that are related to each other.

A feature can be defined as a representation of shape aspects of a product that are mappable to a generic shape and are significant for some product life cycle phase. Semantic information can be associated with the shape information, e.g. on the use of the shape aspects for the end-user, but also engineering information, e.g. on the way the shape aspects can be manufactured. Typical examples of features are protrusions, holes, slots and pockets, but many other features can be thought of, dependent on the specific application, and it should therefore be possible to easily introduce new types of features in a feature modeling system.

All properties of a feature type are specified in the corresponding feature class, including the generic shape of the feature, and a number of parameters and constraints that characterize this shape. Examples of parameters are the two radii and the two depths of a stepped through hole; examples of constraints are that the axes of the two cylindrical holes that comprise the stepped through hole are collinear, and that the radius of the upper hole should be larger than the radius of the lower hole (see Fig. 1(a)). The properties define the basic semantics of the feature class and all its instances.

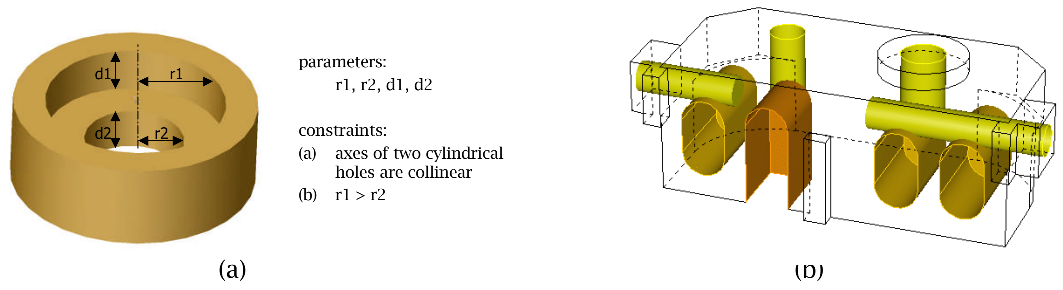

Fig. 1: Feature modeling: (a) stepped through hole feature class, (b) complete feature model.

A user of a feature modeling system usually specifies a feature model via a graphical user interface. He can create instances from feature classes in a library, by specifying values for the parameters, and add them to the model. An instance is normally attached to other features in the model, i.e. some of its faces are coupled to faces of other features. Additional model constraints can be used to relate an instance to other feature instances in the model, and specify other semantics for the feature model. Examples of such constraints are that the axes of two stepped through hole instances should be parallel, at a certain distance, and that the radius of the upper hole of one particular stepped through hole instance should be twice as large as the radius of the lower hole. Feature instances can also be modified, by changing their parameters, or be removed from the model. So a feature model essentially consists of a set of feature instances (see Fig. 1(b)), including all the constraints from their respective class, and a set of additional model constraints between feature instances.

Current feature modeling systems are quite advanced, but fall short in one or more of the following respects: shape domain, i.e. the types of shapes that can be modeled, facilities to specify the semantics of features, and validity maintenance of a feature model.

The shape domain of current feature modeling systems is usually limited to regular-shaped features and simple freeform features, e.g. arising from extrusions of freeform curves. We have recently developed approaches for more general freeform feature modeling, both for freeform surface feature models [13] and for freeform volumetric feature models [2].

The types of constraints that can be used in current feature modeling systems to specify semantics are usually limited to geometric, algebraic and dimension constraints. We have developed several other types of constraints, to be able to more extensively and precisely specify the semantics of features and feature models. These include several constraint types that are specific for freeform features. An overview of these constraint types and what they specify is given in Tab. 1.

constraint type	**constraint specifies**
geometric	geometric properties, such as that two faces should be parallel at a certain distance or be perpendicular
algebraic	arithmetic relations between values of parameters
dimension	range of values allowed for a parameter
boundary	whether a feature face should be present on the model boundary, e.g. that the top face of a hole should not be on the model boundary
interaction	whether certain interactions between features are allowed, e.g. whether a slot may be split into two slots
surface area	minimum or maximum area of face or of all faces of a feature
volume	minimum or maximum volume of feature
continuity	minimum degree of continuity within feature face or between feature faces
curvature	minimum or maximum curvature within feature face
self-intersection	whether self-intersection of feature face or of feature is allowed

Tab. 1: Constraint types.

Validity maintenance in current feature modeling systems is usually limited, i.e. the semantics of a model is not adequately maintained during the modeling process. In many systems ‘features’ are in fact only a kind of shape macros, i.e. only the geometry resulting at the time of creation of the feature is stored in the product model. Other systems do store some information about features in the product model, but this information is not enough to check that the semantics of all features is maintained during the modeling process. For example, a through hole can be turned into a blind hole by blocking one of the openings of the hole, without the system even notifying this change. Although the change in the model is geometrically correct, it is incorrect in the sense that the semantics of the feature is changed from a through hole into a blind hole. Stated differently, the model has become invalid.

We have developed the semantic feature modeling approach in which all semantics in a feature model is checked by the system after each modeling operation [3]. This comes down to solving and checking all constraints in the model. A feature model is valid if all constraints are still satisfied, and invalid otherwise. If a model has become invalid, e.g. if one of the openings of a through hole is blocked, this is notified by the system, and the user is assisted in creating a valid model again. The latter is called validity recovery, and includes reporting constraint violations to the user, documenting their scope and causes, and, whenever possible, providing context-sensitive corrective hints to create a valid model again.

The semantic feature modeling approach has now been implemented for simple and freeform features, and can guarantee that the user can create only valid models [3],[2]. This can significantly improve the modeling process, both initial specification and subsequent modification of a model. In addition, semantics relevant to downstream applications, such as finite element analysis [18] and process planning for manufacturing [16], can be included in feature models and effectively support such applications. Thus, semantics in feature modeling can support the entire product life cycle.

4 MODELING FAMILIES OF OBJECTS

For many applications, what is needed is not just a geometric model of a single object, but a generic model that represents a set of similar objects. Such a set of objects is called a family of objects, and such models are called family models. A family model can be used to represent a product line, e.g. a mechanical part with different sizes, or a customizable product, which can be easily adjusted to specific customer specifications. Also, a family model can represent a design that is not yet completely specified; this model can then be used to explore possible design variations, or can be adjusted to fit as a part of a larger model [10].

Typically, families of objects are represented by parametric models, which characterize all possible shape variations by a set of parameters. By specifying a value for each parameter, a member of the family is selected. Feature models, discussed in Section 3, are also parametric models, where the parameters are the dimensions of the features in the model. However, there are two main difficulties in modeling families of objects with current feature models.

Firstly, we need to specify exactly which objects are members of the family, i.e. the semantics of the family. Current feature models do not allow the semantics of a family to be properly specified. In particular, topological properties of families cannot be specified, and as a result, objects with undesirable topology may be part of the modeled family.

Secondly, a family of objects is an abstract concept that is not always easy to work with. In particular, it is often not clear to the user for which parameter values family members exist, and how parameters affect the topology of the model. Current feature modeling systems provide little support for working with families of objects.

In [9] we have presented a model for families of objects, the Declarative Family of Objects Model (DFOM), which is based on the semantic feature model discussed in Section 3. In the DFOM, the geometry and topology of a model do not have to be fully specified. Thus, a DFOM represents, in general, a family of objects. A DFOM is specified completely declaratively, using geometric and topological variables and constraints. To determine family members, the geometric and topological constraints are solved. Note that in the semantic feature modeling approach discussed in Section 3, topological constraints are only checked, whereas here these constraints are solved.

The building blocks of a DFOM are geometric variables, called carriers, and topological variables, called constructs. Carriers define surfaces that partition space, e.g. a planar carrier defines a planar surface and the two half-spaces on either side of the surface. Constructs basically represent point sets, i.e. volumes, surfaces, curves and individual points, constructed by intersections of subspaces defined by carriers. Carriers and constructs are related via so-called subspace constraints.

To further specify the semantics of a family, geometric constraints are imposed on carriers, and topological constraints on constructs. Geometric constraints include distances and angles between carriers, which can be used as dimension parameters. Topological constraints include nature constraints, which specify whether material should be added or removed by a construct, boundary constraints, which specify relations between constructs and the boundary of the model, and interaction constraints, which specify how constructs may interact with each other (see Tab. 1).

A DFOM may have zero, one, a finite or an infinite number of realizations. A realization is an object that satisfies all constraints. Realizations are determined by solving first the geometric constraints and then the topological constraints [9]. A DFOM that has one realization represents a single family member.

A user has a lot of flexibility for modeling families or single objects. A DFOM can be built using basic variables and constraints, or by combining features, which are essentially other DFOMS that are added as a whole. By adding constraints, the number of realizations of a DFOM can be reduced until only one realization remains, and the DFOM thus represents a single family member.

In Fig. 2, two different realizations of a DFOM are shown. The model does not completely specify how the blind hole may interact with the other features in the model. Therefore, the blind hole may be partially obstructed by a block protrusion feature, as in realization (a). If an interaction constraint is added that specifies that the hole may not be obstructed, then only realization (b) will be found.

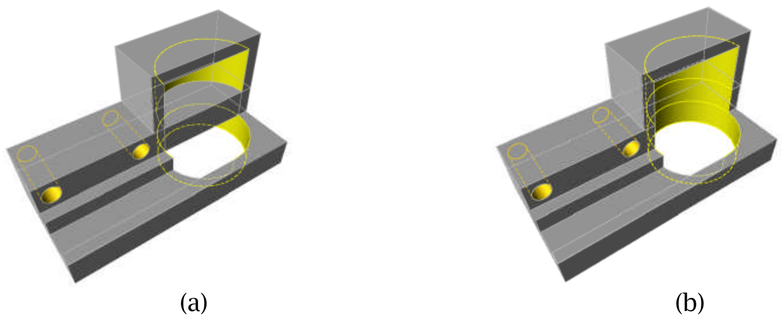

Fig. 2: Two realizations of a DFOM: (a) blind hole partially obstructed, (b) blind hole not obstructed.

A family may not have members for all combinations of parameter values. To help users instantiate family members, we have developed methods to compute parameter ranges, i.e. the range of allowable values for any parameter [8]. Also, we have developed methods to determine the critical parameter values, i.e. the parameter values for which topological changes occur [11]. The latter method allows users to more easily explore the topological variations in a family.

The declarative modeling approach used to specify DFOMs, makes it possible to exactly specify the semantics of families of objects. Such declarative models allow much more flexibility during the modeling process, because incomplete specifications are allowed. Furthermore, critical parameter values and parameter ranges can be computed for declarative models, which can help users with instantiating family members and with exploring and understanding families of objects.

5 MODELING VIRTUAL WORLDS

While designing virtual worlds, whether for games, simulations or other applications, semantics can play an important role as well [20]. Usually, in virtual worlds, semantics is added to objects on a higher level than in CAD applications: the global shape and relative position of objects is more important than their detailed shape.

In our methodology, semantics is defined for object classes using features, which have a shape and a type. The features are defined once for each object class, and are instantiated automatically for every geometric model that is assigned to this class. A simple example is a four-legged table class. We can define a tabletop feature at the top and four leg features at the corners of the bounding box. These features are instantiated for each geometric model of a four-legged table that is assigned to the class. We chose this method, because when designing a virtual world, geometric models for objects have often already been created, typically without any semantics at all. By assigning such a model to an object class in the library, it is enriched with the semantics of this class. The classes in the library also contain attributes. These attributes need to be instantiated for each individual model, either manually or in some cases automatically. For example, the volume of a 3D model can be computed and used to determine the weight of the model.

The object class library further contains formal semantics in the form of relationships, or constraints, between classes. These relationships can involve features to, for example, specify that we want a plate or a mug to be placed on the tabletop feature of an instance of the table class. This means that by assigning a model to a class, the model is automatically enriched with, mostly geometric, relationships. In the methodology used in our research group, we associate these relationships with specific object classes instead of with the feature types directly. The reason for this is that most relationships used in virtual environments are specific for classes, and usually even specific for a given context. For example, plates need to be placed differently when they are put on a dinner table, on the kitchen sink before the washing up, or when stored in a cupboard. Some predefined feature types, however, do have embedded placement rules. For example, so-called off-limits features cannot overlap with any other features, and clearance features can only overlap with other clearance features. This last type is used to, for example, guarantee some free space in front of a cupboard.

The semantic layout solving approach, explained in more detail in [21], accommodates both user-assisted design and fully automated procedural generation of virtual worlds. Given an initial layout, the layout solver can position a new object in that layout, complying with the relationships defined in the classes. This initial layout can be empty or, for example, contain walls when creating a layout for the objects inside a room. By iteratively providing a set of objects to the solver, a valid layout is created. In a manual design application, the user adds the new object. The locations deemed valid by the solver can be shown as guidance to the user, or the application can place the new object immediately on a valid location.

By procedurally adding objects to a layout with this solver, complete virtual worlds can be created automatically. The procedure defines which objects need to be placed in a particular scene and in which order. These objects are step by step added to the solver, which in turn generates a new valid layout. In these procedures, semantic attributes can be used to create specific results. For example, the procedure can contain an instruction to keep adding cabinets to the kitchen until the sum of the storage volume attributes of the placed cabinets exceeds a certain value. In Fig. 3, we show a living room scene generated twice with the same procedure, with and without taking into account some essential object class relationships, thus exemplifying the importance of these relationships in creating realistic scenes.

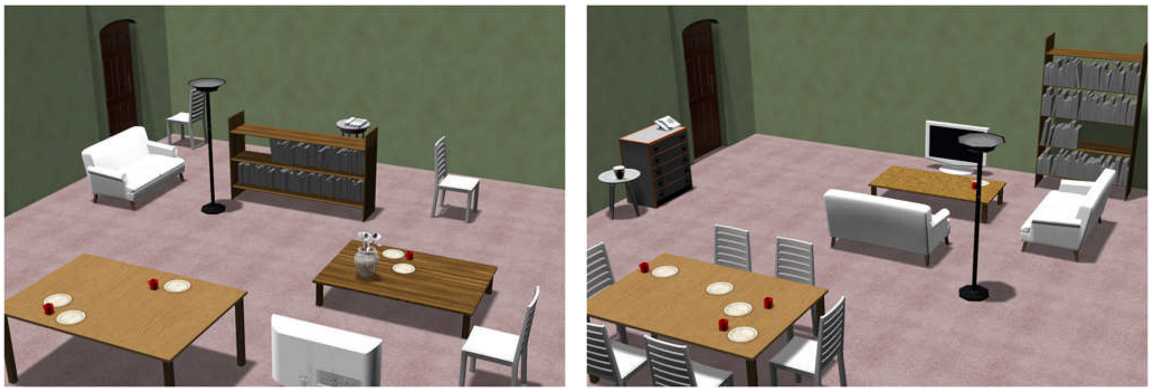
(a) (b)

Fig. 3: A procedurally generated living room: (a) without and (b) with some essential class relationships.

When a game world has been built up and the game is being played, semantics can improve many aspects of the gameplay as well. The AI component of the game, which controls all the non-player characters, can benefit from semantic knowledge. For example, instead of scripting that a particular character walks to a specific vending machine instance every so often, the behavior of this character could include the desire to go to an object that can quench thirst. This information, which we call semantic services, is included in the object classes and is used to support the interaction of the player with the objects as well, as is explained in [6]. These services specify both the actions the player can perform on objects and what the effects of these actions are.

Semantics can thus improve modeling of worlds for virtual environments or games in many different ways, either by aiding the designer while manually editing or by extending procedural modeling techniques. Semantic data can also provide knowledge for the AI engine or on how to handle interaction of a player with the objects in a game.

6 OTHER MODELING APPROACHES

There are several other modeling approaches in which semantics is being used as well. This section shortly discusses some of these approaches, to further illustrate the increasing role of semantics in object modeling.

In [1] an approach is described to characterize objects with semantic annotations. The surface mesh of any object can be automatically segmented into surface features. The user can add annotations to features that express their semantics; for this, an ontology of relevant concepts can be used, and tools are provided to browse this ontology. Dependent on the concept assigned, specific properties of a feature, such as certain dimensions, can be automatically computed and added to the semantic information. Two applications are described: virtual character design, in which a human head having certain characteristics can be easily retrieved from a repository, and product design via internet, in which parts with certain properties can be retrieved from distributed data bases, adapted as needed, and combined into a new design.

In [15] the concept of knowledge-guided NURBS is introduced. The basic idea is that semantics is added to the mathematical equations traditionally used to represent NURBS. The semantics, or knowledge as it is called here, includes for all modeling entities information such as:

- the type of the entity, e.g. a circle or a spline;
- how it was created, e.g. by import or offsetting;
- why it was created and what it may be used for, e.g. for intersection or styling;
- irregularities such as cusps;
- properties such as maximum curvature(s), surface area and volume;
- relations between modeling entities, e.g. to relate a set of points to the curve fitted to these points, or to indicate parallelism between two entities; and
- design intent, e.g. information about the modeling history of the entity.

All this information can be used to support, in particular, more robust computation of NURBS curves and surfaces. More reliable data transfer between modeling systems becomes also feasible, because modeling entities can be recomputed from their origins in the receiving system.

Semantics is of major importance in design automation and knowledge-based engineering systems. Design and engineering knowledge is encoded such that object models can be automatically synthesized from requirements. Such knowledge is usually described using a procedural or rule-based approach. A procedural specification is mostly a program in a specialised programming language for geometric modeling, often tied to a particular CAD system. A rule-based specification is a set of rules, i.e. modeling operations that are conditionally executed, often in random order, e.g. an L-system or a shape grammar [19]. Procedural and rule-based techniques are often used for architectural design [12],[7].

An approach to synthesize mechanical models from functional requirements is presented in [17]. Here, semantics is encoded using a formalism called the Design Exemplar, which can be used to specify objects with geometric, topological and functional properties. For example, the Design Exemplar can be used to describe the concept of a gear, and the concept of a gear box. From these descriptions it is possible to synthesize gear boxes, with various configurations of gears, such that the complete system has a given input-output speed ratio.

So, besides in the modeling approaches discussed in Sections 3 to 5, specific types of semantics are being exploited in several other modeling approaches too, always to improve the modeling process in one way or another.

7 CONCLUSIONS AND FUTURE DEVELOPMENTS

The use of semantics in object models has been discussed in some detail for several modeling approaches, in particular feature modeling, modeling families of objects, and modeling virtual worlds. However, semantics is being used more and more in many other modeling approaches too, and some examples of this have also been given.

The goals of using semantic information in object modeling are to increase the efficiency of the modeling process, both the initial model specification and subsequent modifications, to improve the quality of the resulting models, in the sense that these always have certain required properties, and to better support applications of the models. The results reported in various papers support that these goals are feasible.

Which types of semantics are useful depends on the application at hand, but it is obvious that much more research should be done in this respect. For example, in computer-aided design new types of semantics may be introduced to fully characterize the properties of products, in particular products with freeform features or with patterns, and in computer games new types of semantics can be useful to improve the behavior of object models during gameplay. The desirable types of semantics should therefore be determined in close cooperation with designers and end users of an application.

For the implementation of semantics, in particular formal semantics, constraints are being used. New types of semantics may well require new types of constraints, and possibly new constraint solving and checking algorithms. Developing such algorithms is therefore another important research area for the future, but also improving the efficiency of the whole constraint solving and checking process, because otherwise model validity maintenance might become a bottleneck in the design process.

One particularly interesting research topic is to find a good way to specify semantics for models with repetitive structures, or patterns. What is needed is a declarative way to specify complex patterns in families of objects, such that these patterns can be reasoned about. One possibility is to extend the declarative formalism with a set of pattern types with certain invariant properties.

Besides the applications discussed in this paper, many other applications can benefit from a semantic modeling approach. We envision a declarative framework for geometric modeling, which can be used to formally define semantics for a wide variety of applications, ranging from computer-aided design and modeling virtual worlds, to multi-scale modeling and scientific/medical applications. Such a framework would consist of a declarative language with basic geometric and topological variables and geometric and topological constraints. Using this language, application-specific semantics, e.g. features and modeling operations, can be defined. The framework should provide generic constraint solving functionality, which can be extended with specific solving strategies to improve performance for certain applications.

ACKNOWLEDGEMENTS

The work of H.A. van der Meiden has been supported by the Netherlands Organization for Scientific Research (NWO), and the work of T. Tutenel by the GATE project, funded by the Netherlands Organization for Scientific Research (NWO) and the Netherlands ICT Research and Innovation Authority (ICT Regie).

REFERENCES

[1] Attene, M.; Robbiano, F.; Spagnuolo, M.; Falcidieno, B.: Characterization of 3D shape parts for semantic annotation, Computer-Aided Design, 41(10), 2009, 756-763.

[2] van den Berg, E.; Bronsvoort, W.F.: Validity maintenance for freeform feature modeling, Journal of Computing and Information Science in Engineering, 10(1), 2010, 011006/1-14.

[3] Bidarra, R.; Bronsvoort, W.F.: Semantic feature modelling, Computer-Aided Design, 32(3), 2000, 201-225.

[4] Bronsvoort, W.F.; Bidarra, R.; Nyirenda, P.J.: Developments in feature modelling, Computer-Aided Design & Applications, 3(5), 2006, 655-664.

[5] Hoffmann, C.M.; Lomonosov, A.; Sitharam, M.: Decomposition plans for geometric constraint solving, Part I: performance measures for CAD, Journal of Symbolic Computation, 31(4), 2001, 367-408.

[6] Kessing, J.; Tutenel, T.; Bidarra, R.: Services in game worlds: a semantic approach to improve object interaction, Entertainment Computing – ICEC 2009, Proceedings 8th International Conference, September 3-5, Paris, France, Natkin, S.; Dupire, J. (eds), Lecture Notes in Computer Science 5709, Springer, Berlin, Germany, 2009, 276-281.

[7] Makris, D.; Ravani, I.; Miaoulis, G.; Skourlas, C.; Fribault, P.; Plemenos, D.: Towards a domain-specific knowledge intelligent information system for Computer-Aided Architectural Design, Proceedings 3IA'2003 International Conference on Computer Graphics and Artificial Intelligence, May 14-15, Limoges, France, 2003.

[8] van der Meiden, H.A.; Bronsvoort, W.F.: A constructive approach to calculate parameter ranges for systems of geometric constraints, Computer-Aided Design, 38(4), 2006, 275–283.

[9] van der Meiden, H.A.; Bronsvoort, W.F.: Solving topological constraints for declarative families of objects, Computer-Aided Design, 39(8), 2007, 652–662.

[10] van der Meiden, H.A.; Bronsvoort, W.F.: Modeling families of objects: review and research directions, Computer-Aided Design & Applications, 6(3), 2009, 291-306.

[11] van der Meiden, H.A.; Bronsvoort, W.F.: Tracking topological changes in feature models, Computer Aided Geometric Design, 27(3), 2010, 281-293.

[12] Müller, P.; Wonka, P.; Haegler, S.; Ulmer, A.; Van Gool, L.: Procedural modeling of buildings, ACM Transactions on Graphics, 25(3), 2006, 614-623.

[13] Nyirenda, P.; Bronsvoort, W.F.: A framework for extendable freeform feature modelling, Computers in Industry, 60(1), 2009, 35-47.

[14] Piegl, L.; Tiller, W.: The NURBS Book, 2nd Edition, Springer-Verlag, New York, NY, 1997.

[15] Piegl, L.A.: Knowledge-guided NURBS: principles and architecture, Computer-Aided Design & Applications, 3(6), 2006, 719-729.

[16] Shah, J.J.; Mäntylä, M.: Parametric and Feature-based CAD/CAM; Concepts, Techniques and Applications, Wiley, New York, NY, 1995.

[17] Summers, J.D.; Bettig, B.; Shah, J.: The Design Exemplar: a new data structure for embodiment design automation, Journal of Mechanical Design, 126(5), 2004, 775-787.

[18] Sypkens Smit, M.; Bronsvoort, W.F.: Efficient tetrahedral remeshing of feature models for finite element analysis. Engineering with Computers, 25(4), 2009, 327-344.

[19] Tapia, M.: A visual implementation of a shape grammar system, Planning and Design, 26(1), 1999, 59-73.

[20] Tutenel, T.; Bidarra, R.; Smelik, R.M.; de Kraker, K.J.: The role of semantics in games and simulations. ACM Computers in Entertainment, 6(4), 2008, a57.

[21] Tutenel, T.; Bidarra, R.; Smelik, R.M.; de Kraker, K.J.: Using semantics to improve the design of game worlds, Proceedings of the Fifth Artificial Intelligence and Interactive Digital Entertainment Conference, October 14-16, Stanford, CA, Darken, C.J.; Youngblood, G.M. (eds), AAAI Press, Menlo Park, CA, 2009, 100-105.