Žiga Turk*

Object Oriented Modelling Techniques and Integrated CAD

Submitted to the CIB W78 & CBS meeting in Montreal, Quebec, Canada, May 12 - 16, 1992.

Abstract: *In integrated CAD different techniques are used to model various aspects of the problem domain - information, activities, data and data flow. Finally all models developed using different techniques must at least co-operate which is not encouraged by different modelling techniques. We propose a modelling approach borrowed from object oriented software design introduced by Booch that offers methods and notations for both the static and the dynamic aspects of state and behaviour of the models and offers smoother implementations with object-oriented languages and databases. The paper discusses the motives, the details of the proposed OO modelling approach and an example from the domain of standards representation.*

1. Introduction

If we use different modelling approaches, how difficult will it be to integrate the implementations?

In efforts to develop integrated CAD solutions for AEC industries researchers and developers come across a variety of modelling techniques, notations and data model types. Different approaches are suggested for the modelling of data (EXPRESS), information(E-R, IDEF1X, NIAM, EXPRESS-G), data flow (DFD) and activities (SADT, IDEF0) /FOWL91/. In the end, the solutions based on these distinct models need to be integrated.

Are traditional modelling approaches best suited for object oriented implementations?

Over the past few years there have been clear indications that OO technology (data bases, languages, development shells, operating systems, user interfaces) will eventually be used for the implementation of integrated CAD systems of this decade.

unified approach through OO modelling methods

We agree with arguments /BJOR91/ that a unified approach is needed. We will explore the possibility to implement an OO software design method in information modelling as the basic factor of unification. This should give us the advantage of object orientation from the very start of the modelling process. The approach is complete and includes methods and notations suitable for both static and dynamic features of objects. This should help us to answer the question:

*M.Sc., University of Ljubljana, Civ.Eng.Dept., Institute of Structural and Earthquake Engineering; mail: FAGG, Jamova 2, 61000 Ljubljana, Slovenia; fax: +38 61 268 572; e-mail: ziga.turk@uni-lj.ac.mail.yu

Is it possible to replace traditional modelling methods with OO design methods?

and provide some data to answer the two questions in the sidebar above.

Chapter 2 discusses the modelling process, the object model[1], modelling method and notation. Chapter 3 presents an example.

2. Object Oriented Modelling[2]

First, the model of a modelling process will be shown and the role of the OO technique marked. Key elements of object models will be discussed. Finally the method of OO modelling and the corresponding technique will be presented.

2.1 The Modelling Process

We understand the modelling process as shown in Fig 1.

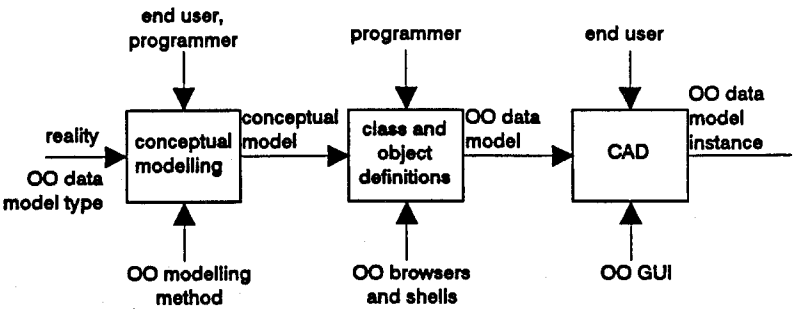


Fig.1: The modelling process. Object oriented techniques and tools are used throughout. The control comes from developers and designers. The tools provide the mechanisms.

2.2 The Model

The object oriented paradigm has become well known and publicised over last few years. This chapter will only point out some of the key ideas that distinguish object models from traditional models that are derived from the entity-relationship data model type. The key concepts of object models (abstraction, encapsulation, modularity, hierarchy, concurrency and persistence) will not be discussed at length.

Objects vs. Entities

entity < object

In the context of traditional modelling methods, the terms object and entity were used interchangeably to describe real world concepts we wish to have information about. Entities would be associated with

[1] As we do not use the adjective "oriented" in other models (relational, network) the term "object model" is preferred to "object oriented model".
[2] Key ideas of this chapter are based on OO design approach as proposed by Booch in /BOOC91/ and /BOOC86/.

attributes and related to other entities with relations. The E-R, IDEF1X and NIAM data types are all versions of this understanding of an entity.

role of objects is that of entities

In the context of human cognition an object can be informally defined as a *tangible entity that exhibits some well-defined behaviour* /BOOC91/. In the context of object oriented modelling an object is (like entity) a basic building block of object models which has three kinds of properties:

three kinds of properties that objects have

- **State** that includes static and dynamic properties of the object. The static properties of objects roughly correspond to relations in E-R models. The fact that a beam is supported by columns may be considered such a static property. By which actual column the beam is supported, how long, wide and deep it is, are dynamic properties which correspond to attributes in E-R models.
- **Behaviour** includes information on how the object acts upon other objects and how it reacts to messages from other objects. The reactions may include state changes or actions (messages). In a physical model of a structure, a reaction to the message "compute" may result in state changes of internal forces inside the beam. A reaction to the message "check" in a conformance checking context would result in a message with "yes" or "no" being returned to the sender. Messages can be divided into five groups: modifiers (1) that change object's state, selectors (2) that query object's state, iterators (3), constructors (4) and destructors(5). The last three become important in the implementation.
- **Identity** is a property that distinguishes the object from all other objects /KHOS86/. Implementations of object data models could add special ID's to object's state (common in object data bases) or resolve identity from object's address in memory.

things you do something to

Static state and behaviour of similar objects can be defined in their class. Particularly the behaviour property of objects introduces the difference between objects and entities. Objects also become *things you do something to* and not only *things you know something about.*

mechanism

A collection of objects that exchange messages exibit behaviour and is called a mechanism. We could understand a mechanism as an extension of the system/subsystem concept which is used in AEC information models.

Relationships

The relationships can be divided into the following categories:

- object to object;
- object to class;

- class to class;

Models are built of objects that may be related to each other in two ways:

- an object **uses** another object;
- an object **contains** another object.

roles of two objects in a "using" relation

The using relation gives an object one of the following three roles. An object may be an **actor** (1) which means that it uses the other object but is never used. An object is a **server** (2) if it is only used by the other object and is an **agent** (3) if it can act both as an actor or as a server.

synchronisation of objects

From the synchronisation's point of view an object may be **sequential, blocking** or **concurrent.**

to contain or to use - no clear boundary between objects and attributes

The relation **"contains"** creates aggregate objects where an object becomes a part of another object. As there is a dilemma in E-R type modelling, whether something is an attribute or a related separated entity, there is a dilemma between understanding some information as a *using* or as a *containing* relation. The first results in more loosely, the second in tightly coupled systems.

class to class relations

Class to class relations involves **single** or **multiple inheritance** (the well-known a_kind_of relation), **using** (a class may use another class in its interface or its implementation), **instantiation** (used in container classes) and **metaclass** (used if a class itself is considered an object). For information modelling only the first two relations seem useful.

2.3 The Approach

Object oriented design is a software development method that is based on the waterfall software development life cycle extended with backloops (Fig. 2). We are interested in the first two phases:

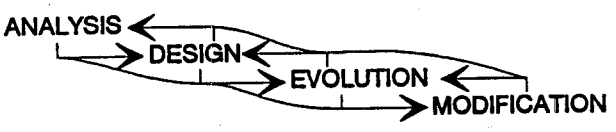
analysis = vocabulary

- object oriented analysis should identify the objects and classes that form the vocabulary of the universe of discourse;

design = behaviour

- object oriented design should invent abstractions and mechanisms that provide the behaviour of the model.

Fig. 2: The software development life cycle.



The attributes of quality design should be followed and include coupling, cohesion, sufficiency, completeness and primitiveness. One of the key tools to achieve that is classification.

Classification

classification = discovery + invention

Classification provides means to organise objects and classes. It can be broken into two sub problems - the discovery of key abstractions and mechanisms and the invention of generalised abstractions. It is considered an incremental and iterative process where it is rarely possible to find the "right" or the "best" solution. The suggested methods for the classification of objects are:

how to find classes

- classical classification (use properties a a criteria for sameness among objects);
- conceptual clustering (formulate conceptual descriptions of classes then classify);
- prototype theory (defining concepts by examples),
- existing classification systems (like the NBC classification system - /VANI91/) in areas where they exist.

making new classes from existiong ones

As the name implies, the result of classification are classes. These can be used to create new classes by **derivation** (derive new specialised classes from existing ones) with its inverse **abstraction** (group classes with a commonalty into super classes) and **factorisation** (split a class into several classes) with its inverse **composition** (join several classes into one class).

OO Analysis

Analysis can either be performed on the problem in question or on the whole domain of problems (domain analysis). Product modelling approach corresponds the second. Some authors /SHLA88/ suggest things that are candidate classes. Other possible analysis methods include the Informal English Description and the Structured Analysis methods /YOUR89/, which are well supported by existing CASE tools.

OO Design

OO Design process is a fuzzy, iterative process (contrary to top-down or bottom up classical designs) which develops solutions through refinement. The process can be broken into the following steps (applied at different levels of abstraction):

the four steps of OO design

- On the basis of the discovered classes and objects (from the analysis phase) we identify classes and objects on a certain level of abstraction. We **discover key abstractions and mechanisms.**
- **Identification of the semantics** (meaning) of classes and objects. The particular interest should be in the interface of the object/class that define the properties that are seen and can be used by other objects/classes.
- **Identification of the relationships** among classes and objects that can be of static and of dynamic nature.

- **Implementation** is concentrating on the representation of classes and objects, grouping them into modules and processes.

2.4 The Notation

one diagram type is not enough

The table below shows the six diagrams needed to capture OO design.

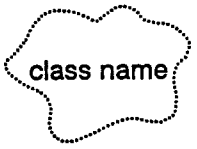
Fig 3: The six types of diagrams used in OO design. In modelling applications we will not be interested in the two showing physical allocation.

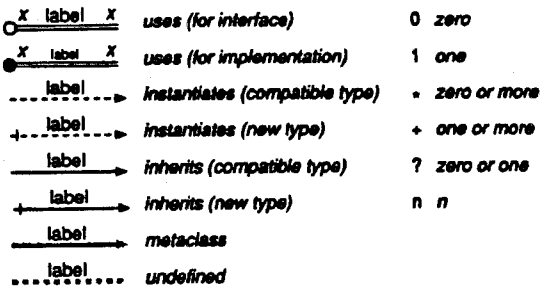
	static features	dynamic features
logical	**class diagrams** show relations between classes	**state transition diagrams** show the state-space of an object, events that cause the transitions and actions that result from a state change
	object diagrams show objects and mechanisms they participate in	
physical	**module diagrams** show the allocation of class and object definitions in modules	**timing diagrams** show dynamic interactions among various objects in object diagrams
	process diagrams show the allocation of processes to processors	

Class diagrams

The graphic vocabulary of the class diagrams consists of classes and various kinds of relationships:

Fig. 4: Right: icons used in class diagrams (from /BOOC91/). Below - class icon.





The shape of class should indicate that it is of irregular form, but has clear boundaries. The boundary however, is dashed, because most operations are performed on the instances of the class. Large projects would require many class diagrams. Booch introduces a concept of class categories to group and relate similar classes. In-depth explanation of classes in the diagrams is provided in class templates - forms with several fields describing several aspects of the class (example in Fig.5).

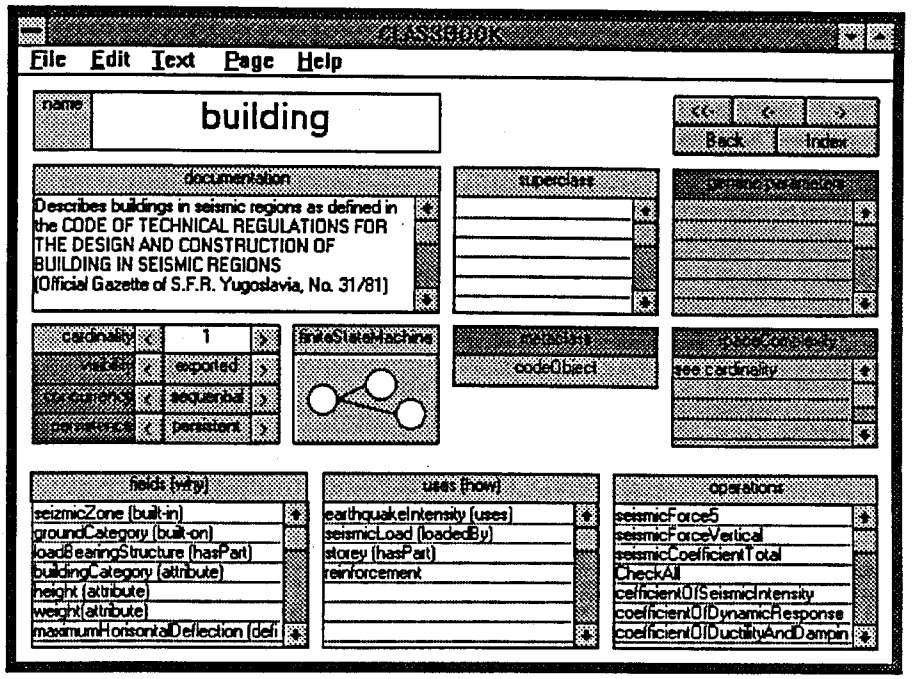
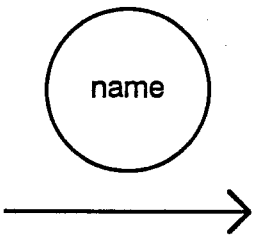


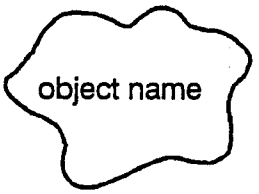
Fig. 5: A page from a Class Book (Developed with Toolbook hypertext system) showing the fields that describe a class.
Darker fields are not important for the modelling stage.
The braces in the "field" and "uses" lists are an extension and show the relations between an object of this class and other classes and are also hyper-linked to those classes. The distinction between "field" and "uses" becomes important during the implementation.

State Transition Diagrams



Since state transitions of objects are caused by actions (messages), the finite state machine of Mealy type is used to capture state transitions. Icons include circles to represent states (double line for start state and thick line for end state), and arrows to represent actions. The details of actions could be explained in PDL (program design language) or in object diagrams.

Object Diagrams



Object diagrams should capture the model of the mechanisms. Its primary goal is to capture the message passing between the objects. The shape of the object icon is the same as of the class icon, except that the line is not dashed. Notation is provided for different kinds of message synchronisation (simple, synchronous, balking, time-out, asynchronous) and object visibility. The letter is of interest in detailed design.

Timing Diagrams

Both object and state transition diagrams do not capture the dynamic nature of collaborating objects. The firsts are static, the second capture only state transitions within a single object. We could add dynamics to object diagrams by adding ordinal numbers to messages. These would then show the sequence in which messages are sent. The second suggested approach is using a PDL for each object diagram and the third the timing diagrams borrowed from hardware design and similar to electronics timing charts. The

method also captures multi-threaded events that will become common programming practice on operating systems like OS/2.

3. Example

The OO design method has been used for a more detailed design of the layered standards representation /TURK91/.

3.1 The Layered Representation of Standards

bridging the gap between the standard and the product

The idea of the layered representation is based on the assumption, that computerised standards[3] representation is limited with two boundary conditions: the paper version of the standard on one end, and with a product to which provisions in the standard apply, on the other. In normal practice, the intellectual distance between the two is bridged by a knowledgeable engineer. Computer representation of standards should give him considerable help, providing that both the standards and the product are available in computerised form. As the name implies, the layered representation bridges the distance by introducing several layers between a computerised "on the paper representation" and the product data (Fig. 6).

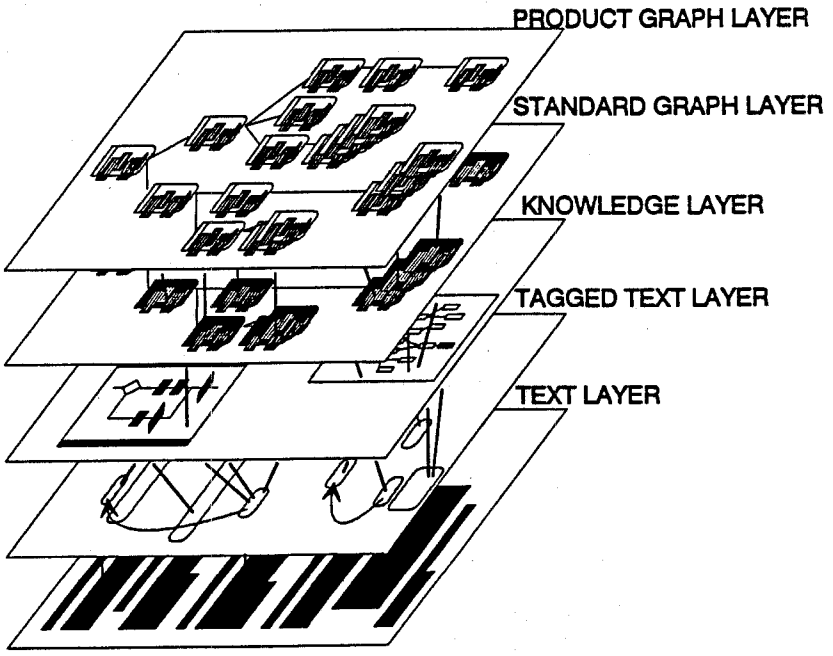


Fig. 6: The layers of the layered representation of standards.

The form and function of the layers are briefly presented in Fig. 7.

[3] The term "standard" is used to denote various kinds of regulations and provisions on different legislative or professional levels.

Fig. 7: The layers of the layered representation. The top to bottom ordering of the layers is the same as in Fig. 5.

NAME	DESCRIPTION	FUNCTION; USAGE	TOOL USED FOR REPRESENTATION
product graph	objects, classes and relations between them as designed (not part of the standard)	provides data to be checked; CAD, conformance checking	OO database
stan-dard graph	objects, classes and relations between them as defined in the standard	uses knowledge and tagged text layer in methods	OO database or KBS shell
know-ledge	declarative and procedural knowledge embedded in the standard	uses layers below to collect data, provides further layers with low level methods; primitive interactive conformance checking	hybrid expert system tool, programming language
tagged text	Tags selected parts of the standard. Tags have name, type and contents. Types are machine usable expressions, quoted text, go-to links, anchors, keywords, titles ...	enables the above layers to access data in the standard; guided electronic browsing	same as text layer, hypertext or hypermedia with typed links
text	machine readable form of the standard as we are used to see it on paper, includes pictures and tables	passive, at best stores constant data - the contents of tags of the ayer above electronic browsing	SGML, TeX, some word-processor text file

3.2 Classes

To make an object model of the layered representation, information in all levels should be modelled as classes and objects. They are shown in Fig. 8. Detailed classes for the representation of products are being defined as a result of the reference model development. The classes for standards representation have been proposed by Garret /GARR90/ and modified in /TURK91a/ to fit into the layered representation scheme. The knowledge layer treats procedural and declarative knowledge in a similar way. The knowledge island technique /TURK91b/ is suggested for the representation. Finally, the tag layer provides an intelligent interface to various information found in the standard text.

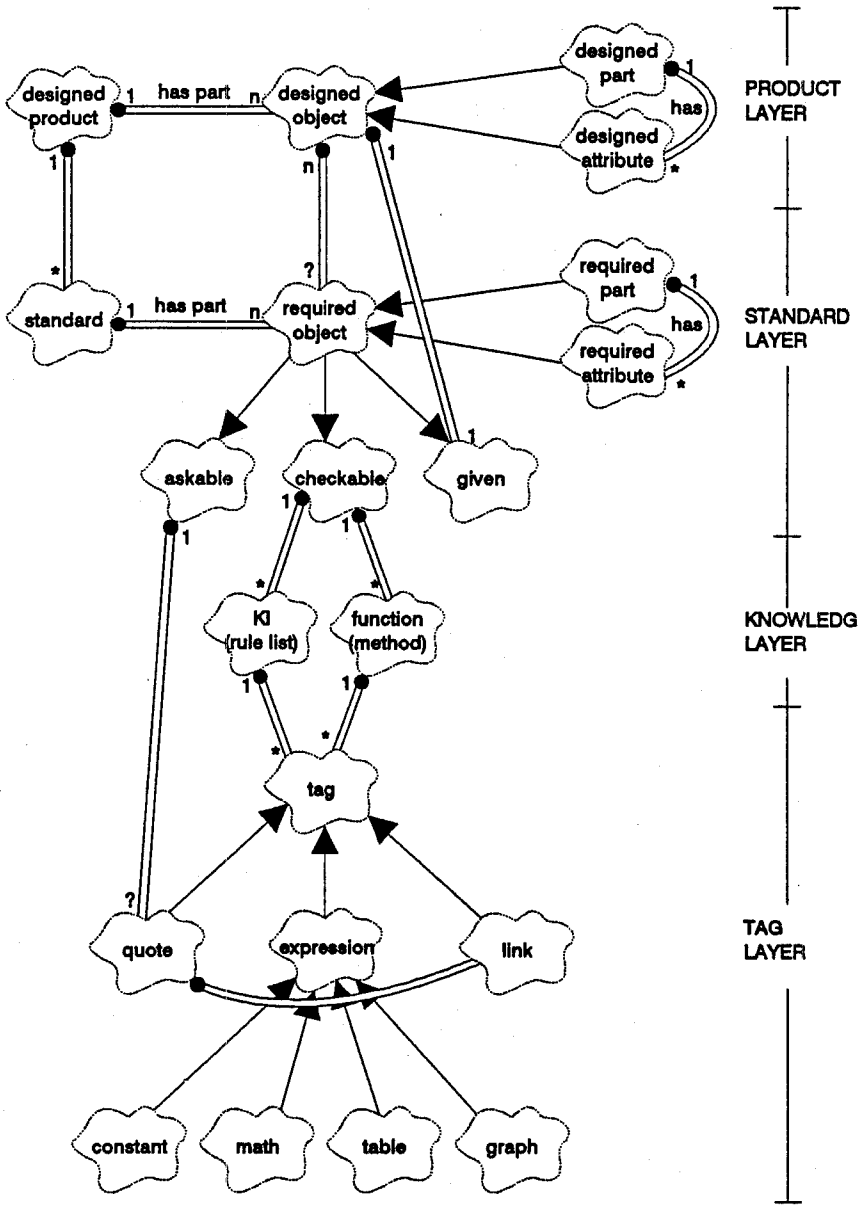


Fig 8: Class relations for the layered representation. There is deliberate symmetry between the classes in the product and the standard layer. Each layer has its own class hierarchy. The exception is the knowledge layer that is being used as a member (method, attribute) of the standard layer.

3.3 Objects and Mechanisms

using relation between objects of different layers for loose coupling

The relations between objects originating in different layers are *using* rather than *containing* because we want or keep the layers only loosely coupled. The objects on the layers above use the objects on layers below. Objects on layers below serve objects on layers above. The relation of objects in standard graph layer to those in the product graph layer is generally that of an agent (they are both used by and use objects in the product layer).

The mechanism for checking the design is shown in Fig. 9. We have decided to choose the product objects as active. Their messages initiate the checking process. Static and dynamic matching between objects in the designed product and those in the standard determine standard applicability. The "check" message asserts the object into the standard. The standard object itself acts as a big blackboard. The actual check is later performed by declarative (aKI) or procedural (aProgram) objects, which may use information from the standard, designed product, the user or standard text.

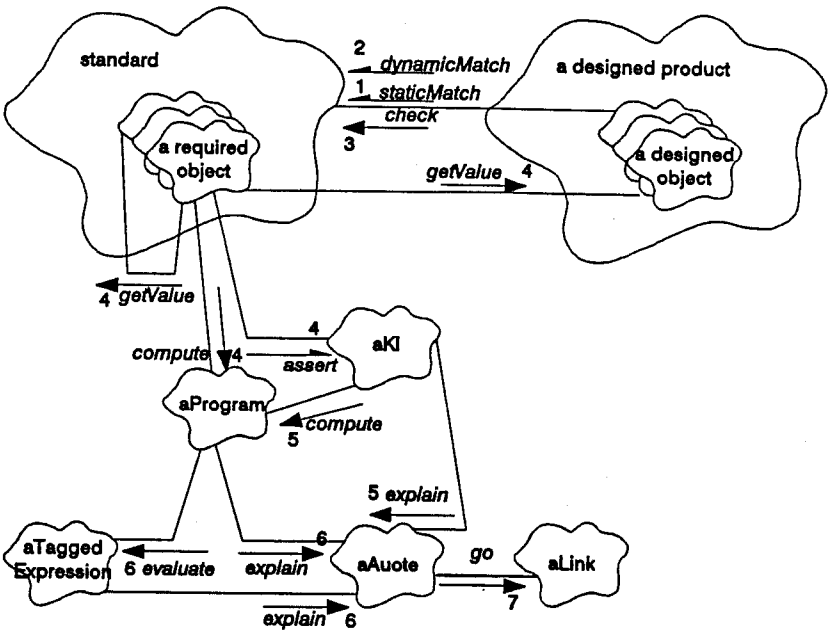


Fig. 9: Object graph that illustrates the messages of design verification. Lines represent message paths between objects and arrows above them the direction of messages from the sender to the receiver. Timing is defined by numbers that accompany messages.

3.4 Timing and State Machine

The state machine of a designed object shows the transition from the unchecked to checked state. Timing during checking is not critical. If the checking process does not alter other states of the object, then the checking processes may run in parallel.

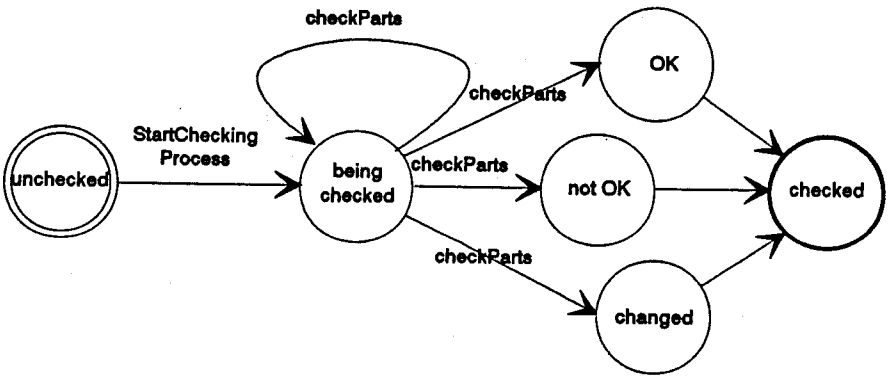


Fig. 10: State machine for the designed object. There are three possible results.

4. Conclusions

OO design forces us to think about active and passive elements of both the standards and the product models in terms of mechanisms that combine the objects. Some of the key classes and key mechanisms of standards representation/usage have been identified. It has been shown that OO design methods *can* be applied to standards modelling. Further research is needed to quantify its advantages and disadvantages over other information modelling methods and the possible implementation of objects with behaviour to the plug compatibility of standards.

exchange of objects

True OO environment should not only support data and information, but object exchange as well. The exchange of objects with well-defined status and behaviour could be the solution for many problems of the integrated CAD domain.

5. References

BJOR91 Bjork, B-C., "A Unified Approach for Modelling Construction Information", accepted paper in Building and Environment, special issue on databases for project integration, 1991.

BOOC86 Booch, G., "Object Oriented Development", IEEE Trans. Soft. Eng., February 1986.

BOOC91 Booch, G., "Object Oriented Design with Applications", The Benjamin/Cummings Publishing Company, Inc., 1991.

CBR91 Kahkonen,K., Bjork, B-C, (editors), "Computers and Building Regulations", VTT - Technical Research Centre of Finland, Espoo, 1991.

FOWL91 Fowler,J., "STEP Modelling Methods", CAD-CAM Data Exchange Technical Centre, University of Leeds, UK, 1991.

KOSH86 Koshafian,S., Copeland, G., "Object Identity", SIGPLAN Notices vol. 21 (11).

SHLA88 Shlaer, S., Mellor, S., "Object-Oriented Systems Analysis: Modelling the World in Data", Yourdon Press, Englewood Cliffs, NJ, 1988.

TURK91a Turk, Z., "Building Model Standard as a Foundation for Computer Integrated Design", in /CBR91/.

TURK91b Turk, Z., Duhovnik, J., "Using Knowledge Islands in an Object Oriented Framework for Integrated Structural Design", in Artificial Intelligence and Structural Engineering, Civil-Comp Press, Edinburgh, 1991.

VANI91a Vanier, D.J., "A Parsimonious Classification System to Extract Project Specific Building Codes", in /CBR91/.

YOUR88 Yourdon, E., "Modern Structured Analysis",Yourdon Press, Englewood Cliffs, NJ, 1988.