Published as: van Leeuwen, J.P. and S. Fridqvist (2003) "Object Version Control for Collaborative Design" In: Tunçer, Özsariyildiz, and Sariyildiz, *E-Activities in Building Design and Construction, Proceedings of the 9th EuropIA International Conference*, Istanbul, TR, October 8 – 10, 2003, EuropIA Productions, pp. 129-139.

Object Version Control for Collaborative Design
Characteristics of the concept-modelling framework

Jos P. VAN LEEUWEN*, Sverker FRIDQVIST*

Abstract: In this paper we describe the main characteristics of the concept-modelling framework that is developed in the DesKs project on Design Knowledge services. Concept modelling gives end-users access to the schema of design models and provides a high level of flexibility for modelling. To support collaborative design, it provides remote data access and allows users to share active resources that, instead of being exchanged or stored centrally, remain at their source in tight relation with business processes. The concept-modelling framework implements object version control and uses timeline management of versions and revisions of objects to increase the integrity between objects that are accessed and edited by multiple users across a network.

Keywords: Collaborative Design; Concept Modelling; Object Version Control; Design Support Systems; Information Modelling

1 Introduction
The *DesKs* research project aims to develop *Design Knowledge services* by implementing the so-called *concept-modelling framework*. Additionally, the framework provides a common theoretical and practical basis for the development of design and decision support systems, which can be applied in the context of collaborative design. This technology provides designers with the tools to formalise design knowledge and to share active design resources.

2 Concept Modelling
Concept modelling[1] is a dynamic form of product modelling. It gives the end-user access to the schema of models used for the representation of designs and products. In concept modelling, the schema used for modelling is expressed in objects called *Concepts*, while a model for a particular design or product is expressed in objects called *Individuals*[2]. The DesKs research project described in this paper has implemented the concept-modelling paradigm into a theoretical framework and an API that form the basis for the development of design and decision support systems.

---

* Eindhoven University of Technology, The Netherlands
(J.P.v.Leeuwen@tue.nl - http://www.ds.arch.tue.nl)

1 In previous publications, we have used the term *feature modelling*. Although in many early publications in the area of feature modelling the term feature has generic meaning, most of the later research on feature modelling is restricted to form features. Therefore, using the term feature in a generic sense appeared to be confusing and we have decided to change our terminology, using *Concept* for Feature Type and *Individual* for Feature Instance.

2 For reason of clarity, capitalization is used to distinguish the terms Concept and Individual as objects in the concept-modelling framework from the terms concept and individual in the English language.

2.1.1 Concepts and Individuals

Any Individual is related to a Concept that defines its initial structure and value. The relationship between an Individual and its Concept is maintained, although it is a loose relationship. It is possible to change what Concept an Individual is based on, which allows a designer to adapt the meaning of the objects representing the developing design as the design process proceeds. For example in spatial design, the outer contours of a building can initially be modelled as Individuals that are created from a generic Concept for space boundaries, while at a later stage these Individuals may refer to a more specific Concept defining some kind of cavity wall.
In the concept-modelling framework, both Concepts and Individuals are internally part of an object model that is available for application development.

2.1.2 Property-oriented modelling

Concept modelling is a property-oriented modelling approach [LHF 2001]. A property can be explained as a characteristic that is attributed to a thing. Examples of properties are: colour, shape, mass, location, and function. Many different kinds of things can have a property in common, but no two kinds of things share all properties [LF 2002a].
A property-oriented modelling system allows the designer to initially create an empty object to represent his design, and then to add properties to that object to reflect the different things he needs to express about the design. Alternatively, the designer can just create the desired properties of the design and add them to the model, while postponing the decision to which objects these properties are attributed. Property-oriented modelling allows the designer more freedom to decide upon the context of properties and how properties work together in achieving the required functionality of the designed object, which is an important capability during the course of design. Thus, a property-oriented system does not impose any order or other restrictions on the design process [LF 2002a].
In the concept-modelling paradigm, property orientation is achieved by allowing the addition of properties to Individuals whose Concepts do not predefine those properties. This enables designers to make Individuals more specific than their original, generic Concepts. At a later stage, an Individual's reference to Concepts can be refined to better reflect the actual meaning of the particular Individual. For example, an individual 'door' in a model, which is initially based on a generic 'door' concept, may receive additional properties during design that make it a sliding door and transparent. At the appropriate moment in the design process, the designer can decide to make this particular door refer to a specific concept provided by a manufacturer of glass sliding doors. Properties in the concept-modelling framework are represented by *components*. Components are used to define the meaning of both Concepts and Individuals.

2.1.3 Multiple instantiation and multiple inheritance

Individuals can relate to multiple Concepts. This reflects the notion, for example, that a particular object in a design exhibits several functions, or can be regarded from various points of view. This is useful for making it possible to attach

knowledge to objects whenever the knowledge comes into view. For example, a Concept definition for costs can be added to a particular object in the design when costs become an issue in the design process. In object-oriented terminology, this capability could be explained as an object being an instance of multiple classes, which are determined by the end-user.
Concept modelling also supports inheritance of components. This allows the definition of sub-concepts that inherit components of other concepts but that have additional components to make it a more specific concept. Multiple inheritance is supported to allow concepts to inherit components from more than one other concept.

2.2 Examples
Some examples of how the concept-modelling framework is used will clarify its capabilities. Figure 1 shows the graphical notation that is used for the representation of Concepts and Individuals. Note that this graphical notation does not reflect the full features of the modelling framework but merely serves illustration and interfacing purposes.

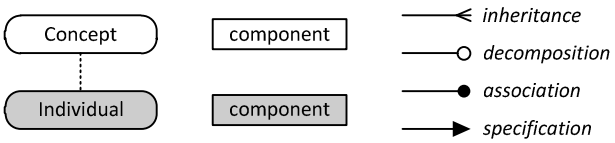

Figure 1 Graphical notation of Concepts, Individuals, components, and relationships.

Figure 2 shows a Concept 'Door' that has two components that define the width and height of doors. Both these components refer to the Concept 'Measure.' Not shown in the picture are the properties of the components, such as their cardinality. In grey, the figure shows an Individual 'MyDoor' that is based on the Concept 'Door'. Through the component 'width' it provides a relationship with an Individual of the Concept 'Measure' that is shown by its value of '1000mm.' The component 'height' is not created at this point. In addition to the components that are defined by the 'Door' Concept, 'MyDoor' has two components that define it as a sliding, transparent door. Note that in this example all components form a so-called specification relationship between Concepts.
At a later stage in the design process, the designer may want to find a more specific Concept for the Individual 'MyDoor' that better matches the description he has given it. Searching manually, or using a technique called Concept Recognition, such Concepts may be found using resources on a network, for example, a manufacturer's resources. Figure 3 shows a Concept for a sliding aluminium door that inherits from the generic 'Door' Concept, adding components for the opening method and for the materials used. Since one of its materials is glass, this door matches the search criteria. Note that the Concept inherits the width and height components from the generic door and that the additional components refer to Individuals, rather than

Concepts. When a component refers to an Individual, this means that its value is fixed at the conceptual level: all Individuals based on this concept will be sliding doors in aluminium and glass. This mechanism allows the definition of static information at the conceptual level.

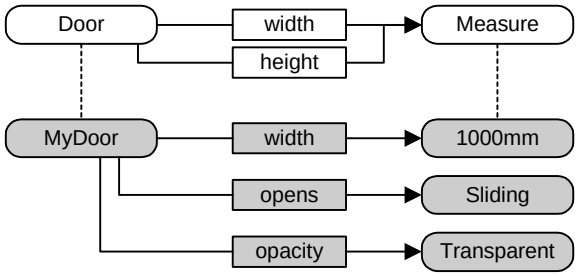

Figure 2 Example of a generic Concept 'Doors', defining 'width' and 'height' components that refer to the Concept 'Measure.' An Individual door that specifies one of these components also has additional components for the opening method and opacity of the door.

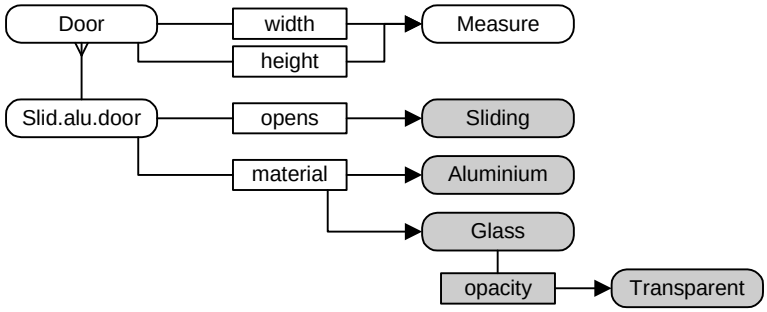

Figure 3 Example of a Concept that inherits from the 'Door' Concept and that has references to Individuals as fixed values for some additional components.

3 Collaborative Design

The concept-modelling framework, succinctly described above, is designed to support collaboration in design. In collaborative design, apart from the social and organisational aspects, a number of important technological issues refer to means of communication and to the shared access to design information by multiple users on distributed locations. The concept-modelling framework enables the sharing of design knowledge by making use of remote data-access technology. In contrast with, for example, document management systems and project web sites, in our approach the point of departure is not to centralise project data, but to keep data at its source. This has the advantage that the data will remain tied to the business processes and that access control and validity of data remains the responsibility of the owner of the data. Access control is managed on the basis of objects: various levels of access to

individual objects can be granted to groups and users. Editing in this environment is thus not restricted to the information that is owned by a particular user or group; one can also get the rights to edit information in other sources. Communication is not based on the exchange or sharing of documents, but on the sharing of objects. Design information, be it generic information or information concerning a particular design project, is stored and accessed through a network of distributed resources.

3.1 Multi-user access control

To facilitate collaborative design, it is necessary to authenticate the access of users to shared information resources. Also, levels of access must be specified for controlling how users are authorised to perform operations on information. In the concept-modelling framework, access-levels are used to govern reading, copying, using, referring to, and editing information. Editing is controlled by a checkout-and-commit mechanism that works on object-basis. Users have to check out an object, marking it as being under revision, before they can make changes to it. Once the changes are made, the object is committed back to the source and stored as a new version or revision (see section 4). The checkout mechanism can be applied to work automatically or manually. This is related to three modes of editing that are distinguished:

- *Instantaneous editing* is required when changes made by one user should instantly be visible to other users. This mode of working is applied, for example, in virtual workspaces when users collaborate synchronously on a design and need to see and communicate about each other's modifications, such as dragging an object, in real time. During such a drag-operation, the changes to the coordinates are instantly made available to all users.
- *Intermittent editing* is sufficient when users do not need to have instant updates of modifications in synchronous collaboration sessions. The changes are made available only when the user has committed them.
- *Off-line editing* is relevant when network facilities are not permanently available. Objects remain checked out for a longer period and changes are committed only the next time a user is online.

The implementation of the concept-modelling framework uses remote data access and ensures that multiple accesses to an object actually address one single object.

4 Object Version Control

In this section concentrate on one aspect of multi-user access to shared information: version control. Since the concept-modelling approach uses objects instead of documents as the basis for structuring and organising information, version control is required on the level of objects. In the remainder of this paper 'object' is used to denote all objects for which version information is maintained: Concepts, Individuals, and Components (for both Concepts and Individuals).

4.1 Why object versions?

Maintaining versions of objects representing a design is interesting for the purpose of documenting alternatives of that design. Additionally, in the context of

collaborative design, version management of objects is important to maintain the consistency of an object model that is accessed by multiple users. Changes to objects will be administered through the creation of versions and revisions, which ensures that the state of objects recorded in previous versions will remain available. References between objects can make use of the version information of objects, so that the data consistency is not compromised when new versions are created. Semantic consistency is, of course, not ensured by the implementation of object version management.
In literature, version control at the object level is described in [CJ 1990], who use so-called 'stamps' to identify object versions in multi-version databases; in [BER 1997], proposing basic operations on versions that are identified through a *succeeds* relationship; in [KNN 1999] who describe referent tracking documents as a means to control version information through hyperlink management.
Administering versions and revisions of objects provides a means to archive the changes to objects. In combination with authenticated access, it is possible to trace the changes of objects to the users who made those changes. Having a record of the history of each object also facilitates the browsing and restoring of previous states of a design model. This has potential for, e.g., the narrative representation of designs and for computer applications used in design education and research.

4.2 Levels of versions

Version information for objects in the Concept Modelling framework is structured in three levels. In the top two levels, an integer number is used to identify versions: one for *major versions* and another for *minor versions*. Numbering starts at 1 and minor version numbering is restarted within each major version. New major versions may be initiated by the user either when he regards the changes significant enough for a new major version, or by the system when the changes are such that consistency problems are likely to arise in other places of the model. For example, a new major version is created by the system when a component is removed, because existing references to the concept may rely on the presence of the component.
The third level of version information is for *revisions*. When an object is checked out for editing, it will remain under revision until it is submitted again as a new version. Also, new objects are initially under revision until they are submitted. Revisions are identified by their creation time. The revision information is also maintained for versions of objects, so the timestamp is available for each object-version as well.
In the concept-modelling framework, an editing activity is concluded by either *committing* a revision or *submitting* a version. How this is done, manually or automatically, depends on the implementation in the user's application that is based on the framework. The implication of this is that, once committed or submitted, revisions and versions are fixed and can no longer be changed. Changes on objects will always lead to the creation of new revisions or versions. On the one hand, this helps ensure consistency in the model. On the other hand, it calls for smart ways of referencing objects, such that up-to-date information is used when referring to an object. This is discussed further in section 4.4.

4.3 Timeline management

Versions and revisions of objects have timestamps that designate their lifetime. Because each version of an object is also a revision, we will refer to 'revision' in this text to indicate both. Each revision of an object always has a 'valid from' timestamp, indicating the moment this revision was created. When a revision becomes outdated, either because the object was deleted or because a newer revision was created, this revision will also get a 'valid to' timestamp. This concludes the lifetime of the particular revision. Subsequent revisions together form the lifetime of an object. Normally, the 'valid from' timestamp of a revision corresponds to the 'valid to' timestamp of its predecessor. It is possible, however, to revive an object that at one point has been deleted. In this case, the timeline of revisions will show a gap. Figure 4 shows the graphical notation that is used for the representation of timelines of objects. Blocks indicate the beginning and ending of a particular revision's lifespan; an arrowhead denotes 'no ending time' meaning that the revision is the current one.

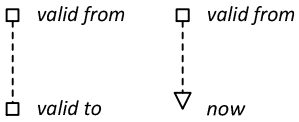

Figure 4 Graphical notation of a revision's timeline

Using the examples in the following figures, we will explain the functioning of the timeline of objects. Figure 5 shows a Concept *C1* that was created at time *t1* and a Concept *C2* that was created at time *t2*. Component *a* that refers to *C2*, was created and added to *C1* at time *t3*. The addition of this component to *C1* signifies a new minor version of *C1*. At point *t5*, a new version of component *a* was created (for example, because its cardinality was enlarged). Note that this does not result in a new version of Concept *C1* that owns it. The deletion of component *b*, however, results in a new major version of *C1* at point *t6*. The timeline management of objects makes it possible for the system to find the correct references at any particular moment in time. A change to Concept *C2*, as shown at point *t8* in the timeline, is thus automatically taken in account when the reference from *C1* through its component *a* is followed at the current moment in time, indicated as *now*. The mechanism that deals with this follows the timelines of related components and Concepts to their most recent revisions that are alive at a given moment in time. When we want to examine the state of version 1.3 of *C1*, this mechanism would look up the 'valid to' timestamp of *C1*'s version 1.3 and subsequently find the component *b* version 1.1 and component *a* version 1.2 whose 'valid from' and 'valid to' times straddle this timestamp. Following component *a*, version 1.1 of Concept *C2* would be found as the version that is relevant for *C1*'s version 1.3.

Looking at the latest revision of Concept *C1* this way (*now*), the reference to Concept *C2* by component *a* will be followed to the latest version 2.1 of *C2*.

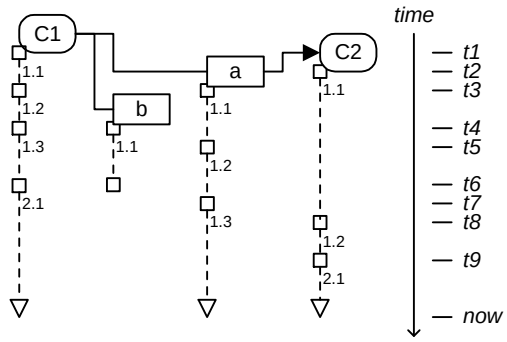

Figure 5 Example of a timeline of a structure of Concepts.

A more complex example is shown in Figure 6 where a new version of component *a* was created at point *t5* by changing its reference from *C2* to *C3*. *C2* was then deleted at point *t6*. Component *a* itself was deleted at point *t10*, leading to a new major version of Concept *C1*.

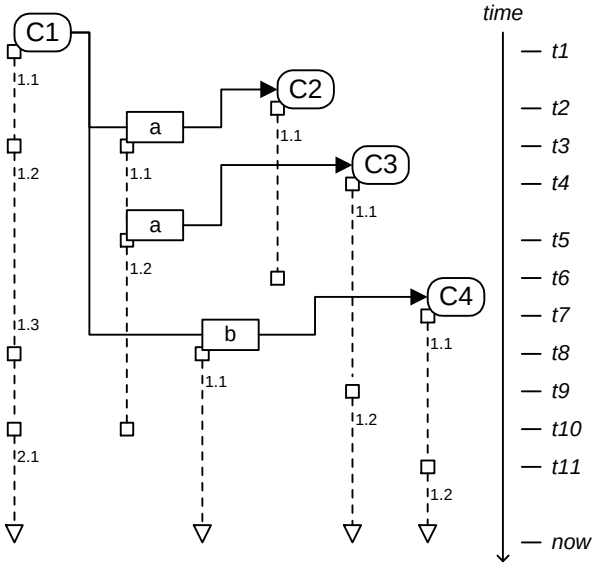

Figure 6 More complex example of a Concept's timeline
Component a first changes its reference and is later removed altogether, leading to a new major version for Concept C1.

4.4 Using the version control mechanism

References in the concept-modelling framework are made with an indication of the level of version information that should be included in the reference. The levels used

in references are *minor*, *major*, and *logical*, as shown in Figure 7. Making a reference to an object without any version information signifies a reference to the logical object (see component *a* in Figure 7). Such a reference will always point to the most recent revision of the referred object at the given moment in time. By including version information, the reference can be restricted to either a particular major version or a particular minor version. When a major version is referenced, the latest minor version within the major version is used. References at the level of revisions are not relevant, since the level of revisions is intended for editing purposes only and cannot be used for making references.

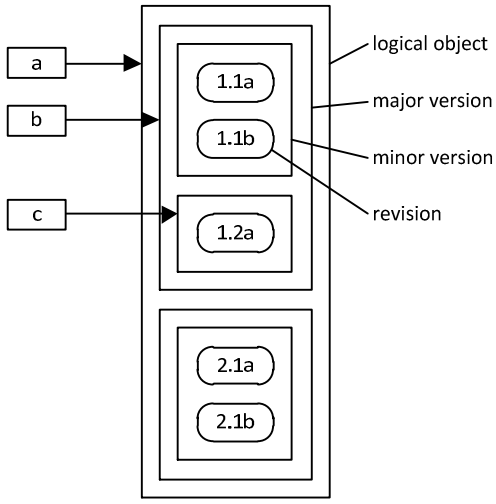

Figure 7 Four version-levels of detail exist:
logical object, major version, minor version, and revision.
References to objects can be made to the first three of these levels.

Looking again at Figure 5, the reference from component *a* to Concept *C2* at the moment *now* can result either in the retrieval of version 1.2, for example in case the reference was made to the major version 1, or in the retrieval of version 2.1 if the reference was made to the logical object *C2*.

Because revisions and versions, once submitted to the system, cannot be removed anymore, the term ‘deletion’ gets a special meaning. When an object is deleted, its latest revision is marked as ended by setting its ‘valid to’ timestamp. References to the existing versions can still be made, but in the de-referencing mechanism their timeline will be taken into account.

One of the advantages of having version control on the level of objects is that ‘undo’ operations can be performed at the object level as well. ‘Undo’ in this context means to re-establish a previous revision. This does not lead to a factual revival of the particular revision, but to the creation of a new revision that has the state of the previous one. Strictly speaking, ‘undo’ is thus not supported, but the re-establishing of any earlier state of an object is, which is in fact a richer mechanism.

4.5 Subscription and notification

In a collaborative design situation, changes to objects made by one user are often of interest to other users. To get informed of such changes, a user can subscribe to notifications issued by an object. If the subscription request was accepted, the notification is handled autonomously by the system and may lead to an automatic update of references or even an automatic upgrade of object versions. The right to subscribe to an object is one of the access rights that the owner of an object can grant to other users, which is a necessary restrictive mechanism built into the system to be able to limit the amount of communication.

5 Conclusions and current work

Using timeline management for object version control has a number of major advantages. Firstly, adding the time information helps to register all design activities, which makes it easy to interpret what has happened in the process. Secondly, time-based relationships can be found between versions of objects. This makes it easy to find the correct version of related objects in a particular context. Finally, with time-based version control it is not necessary to propagate the creation of new versions up into hierarchical trees. This approach potentially solves some of the problems addressed in research on version propagation, as described in [UO 1996], since it makes evident what the lifetime of each version is.

The presented framework is currently implemented in the form of an Application Programming Interface (API) that forms the basis for the development of design support systems. One system that is under development at the time of this writing is an application that enables designers to search through product descriptions, provided in the concept-modelling format, using the so-called concept recognition algorithm. This algorithm compares the structures of Individuals in a particular design model with the definitions of Concepts in order to find a Concept that matches the design implied by the configuration of Individuals. Integration of this application in, e.g., existing CAD systems will make it possible to create a direct link between design stages and the information available from the supply chain.

6 References

[BER 1997] Bernstein, P.A. 1997. “Repositories and Object-Oriented Databases.” In: Dittrich and Geppert (eds.) *Datenbanksysteme in Buro, Technik und Wissenschaft (Proceedings of BTW Conference)*, Springer Verlag, Berlin.

[CJ 1990] Cellary, W. and Jomier, G. 1990 “Consistency of Versions in Object-Oriented Databases.” In: *Proceedings of the 16th VLDB Conference*, Brisbane, Australia, 1990. pp. 432-441.

[FRI 2000] Fridqvist, S. 2000. *Property-Oriented Information Systems for Design*. PhD thesis, Lund University, Sweden.

[KNN 1999] Kimber, W.E., Newcomb, S., and Newcomb, P. 1999 “Version Management as Hypertext Application: Referent Tracking Documents.” In: Usdin, B.T. (ed.) *Proceedings of Markup Technologies ’99*. Philadelphia, Pennsylvania, USA, Dec. 7-9, 1999. pp. 185-198.

[LEE 1999] van Leeuwen, J.P. 1999. *Modelling Architectural Design Information by Features*. PhD thesis, Eindhoven University of Technology, The Netherlands.

[LHF 2001] van Leeuwen, J.P., A. Hendricx, and S. Fridqvist. 2001. "Towards Dynamic Information Modelling in Architectural Design." In: *Proceedings of CIB W78 workshop on Construction Information Technology*. Mpumalanga, South Africa, 30 May - 1 June, 2001.

[LF 2002a] van Leeuwen, J.P. and S. Fridqvist. 2002. Supporting Collaborative Design by Type Recognition and Knowledge Sharing. Electronic Journal of Information Technology in Construction (ItCon). vol 7 (2002). 167-181.

[LF 2002b] van Leeuwen, J.P., and S. Fridqvist. 2002. On the Management of Sharing Design Knowledge. In: Proceedings of the CIBW78 Conference – Distributed Knowledge in Building. June 12 – 14, 2002, Aarhus, Denmark.

[UO 1996] Urtado, C. and Oussalah, C. 1996. “Semantic Rules to Propagate Versions in Object-Oriented Databases.” In: Novikov, B. and Schmidt, J. (eds.) *Advances in Databases and Information Systems*. International workshop, Moscow, Sept. 10-13, 1996.