# A FRAMEWORK FOR EVOLUTIONARY INFORMATION MODEL DEVELOPMENT

P.H. Willems [a], P. Kuiper [a], G.T. Luiten [b], B.F.M. Luijten [a] and F. P. Tolman [a,b]

[a]TNO Building and Construction Research, Department of Computer Integrated Construction, P.O. Box 49, 2600 AA, Delft, The Netherlands.

[b]Delft University of Technology, Civil Engineering, Computer Integrated Construction, P.O. Box 5048, 2600 GA Delft, The Netherlands.

**Abstract**

Large scale information modelling projects, like the development of ISO/STEP, require a modelling approach that a new model not be developed from scratch but to base it on a more generic model which, in its turn, can be based on an even more abstract model, etc. The resulting structure shows a layered framework. On top you will find the most generic concepts and downward the more specific concepts with increased semantics. The benefits of such a model development approach are improvements in: version management, object orientated modelling, concurrent model development, controlled change, standardized interfaces, conformance testing etc.

This paper describes an environment which supports the development of a new model out of one or more generic parent models. The generation process consists of two steps. In the first step entities of the parent models can be instanciated while constraining the inherited behaviour and introducing new behaviour. In fact this process is identical with instanciating run time objects from class templates in the object oriented paradigm. However, in our development environment an important (inherited) property of each entity is *self-reproduction*. In the second step, therefore, each instance is forced to represent its run time state into some kind of information modelling language specification. Appropriate measures are taken to guarantee that the resulting model will conform the behaviour of its parent model(s).

The paper will demonstrate this approach in a multi-layered example currently being implemented and will explore several implementation issues.

## 1. INTRODUCTION

Six years of continuous effort to develop the large scale international Standard for Exchange of Product Model Data (ISO TC184/SC4 STEP) reveal a number of problems. Probably the toughiest issue is: how to integrate all present (and future) models into one unambiguous standard. Until now a more or less bottom-up approach has been applied, where relatively independently developed models are integrated gradually assembling a common core (Generic Product Definition Resource) [2] and a set of integrated topical models. The integration process is a very demanding and difficult task which is essentially sequential in nature. This centralized approach will become a tremendous bottle-neck for a rapidly increasing queue of unintegrated models. In fact the situation is even worse, because the models themselves are not static. New versions replace obsolete versions which again must be integrated, etcetera.

What is needed is an approach that supports decentralized, concurrent model development. A lot of integration nuisances can be prevented by restricting the modelling freedom of the developers somewhat. Instead of always starting from scratch, new models should be based on one or more integrated generic models. Following this strategy, models will be very alike in struc-

Construction Informatics Digital Library http://itc.scix.net/
paper w78-1991-26.content

ture and for that reason already highly integrated. An example of this approach is the Road Model Kernel, an information model for road design [11], which in its turn is based on the General AEC Reference Model (GARM) [5].

In the recently initiated STEP Framework project (WG5/P1) a set of basic concepts have been formulated to establish such a practice [7]. In the next section an abstract of this approach is sketched which here will be referred to as *layered modelling*. The remainder of this paper deals with an implementation technique to support layered modelling.

## 2. LAYERED MODELLING

The modelling space which will ultimately be addressed by STEP is huge. To achieve some order in this modelling heap the concept of *modelling domain* (also called *Framework dimension* [1]) is introduced. A modelling domain, in the context of the STEP Framework, represents a general property of interest. The idea is to define the total STEP modelling space as the intersection of a set of *n* orthogonal modelling domains. This modelling space can be represented as a hypercube with n dimensions (modelling domains). Figure 1 illustrates this principle with $n = 3$.

Modelling domains may be organised in levels or value sets creating a linear subdivision for that dimension. Any subvolume in this modelling space is uniquely identified by stating its various levels for each dimension. Because of this identification principle domains have to be orthogonal otherwise ambiguity will definitely occur.

Examples of *modelling domains* [6], or *Framework dimensions* [1] are:

- Specialization
- Life cycle
- Composition/decomposition
- Concretization (required, proposed or realized)
- Parameterization & Specification (generic, specific, occurrence)
- Definition & Representation

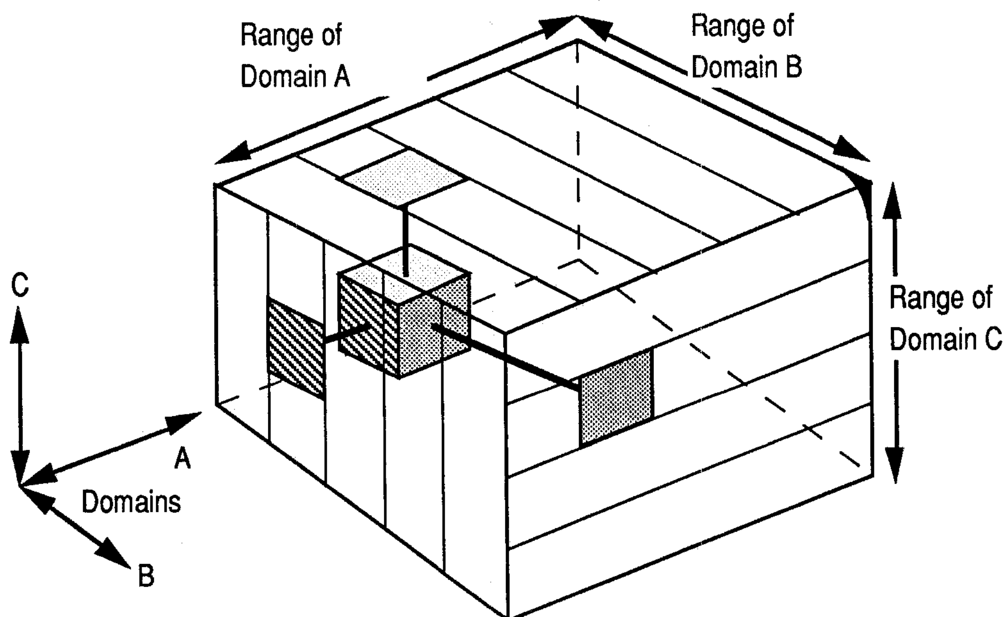


Figure 1.

The orthogonal modelling domains fulfil one important Framework goal: *Provide the means to clearly and formally declare the boundaries of any particularly modelling area.* Another important goal is: *Provide flexibility for the modeller working within specified domain boundaries to change, enhance, and refine internal elements at will, without invalidating other dependent models.*

The rational behind this goal is the fact that the set of models which constitute a particular STEP version have dynamic aspects: they change from version to version. On the other hand, STEP is not just a bunch of unrelated models, on the contrary, they must be highly integrated making use of each others functionalities. To be able to allow models to change their exported functionality should be offered via a public interface. In close analogy with the object oriented paradigm models should not directly use constructs in other models but address its public interface only. Along a one-dimensional domain-related view the models and model-interfaces may be assembled into a multi-layered structure. This leads to the layer paradigm as illustrated in figure 2. In general the client/server relations may occur in both directions, however, certain domains like Specialization allow only one direction[1].

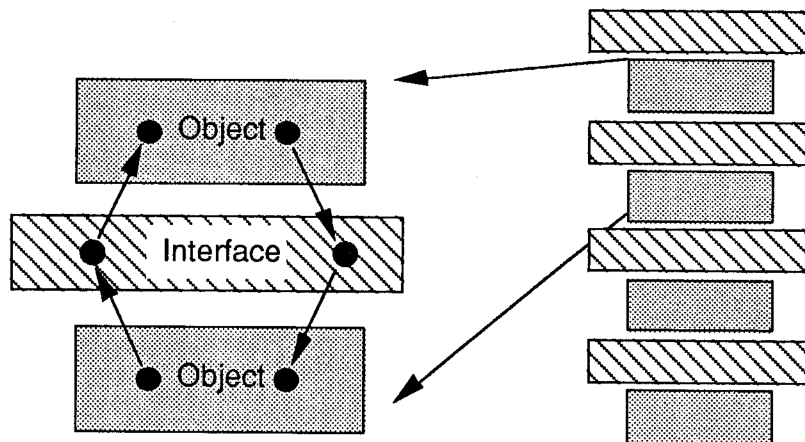


Figure 2.

## 3. IMPLEMENTATION ISSUES

A model development approach as described in the last section is much more appealing if there is some software tool to support it. The application of object oriented techniques and languages seems obvious. But then the next question arises which can be phrased roughly as: *Shall I implement the models as (1) classes, i.e. as source text of an object-oriented programming language, or as (2) data, i.e. as input to operate upon, or as (3) something in between, i.e. some models as class source text and others as data?*

Implementing models in class source texts has the tremendous advantage that compilation results directly into a modeling kernel to instanciate the corresponding models. The disadvantages are the inflexibility of the system for model evolution (every change may lead to lots of recompilations), while the notion of layers is not reflected in the class structure (in fact, it will be one massive collection of classes).

Implementing models as schema data is very flexible but there is no way to take any advantage of the specific functionality offered by the applied object oriented implementation language. To create a system with the same functionality as the first implementation method (models described in class source texts), all object-oriented advancements must be programmed explicitly.

---

[1] The Specialization modelling dimension will probably be implemented as subtype/supertype structure.

Something in between? It looks pretty arbitrary where to draw the line between models in source and models in data. However, to choose the Specialization Dimension for this purpose is less arbitrary because of the mono-directional relations between the layers (figure 3). The more generalized layers are candidates to implement in software while the more specialized layers may be implemented in data. Specialization in the 'software' area is easily implemented using the inheritance feature of the object oriented paradigm [10]. The area of interest is, of course, the zone defined by the layers on both sides of the source/data line. Here, instanciating a class into an object is conceived as a method to represent specialization.

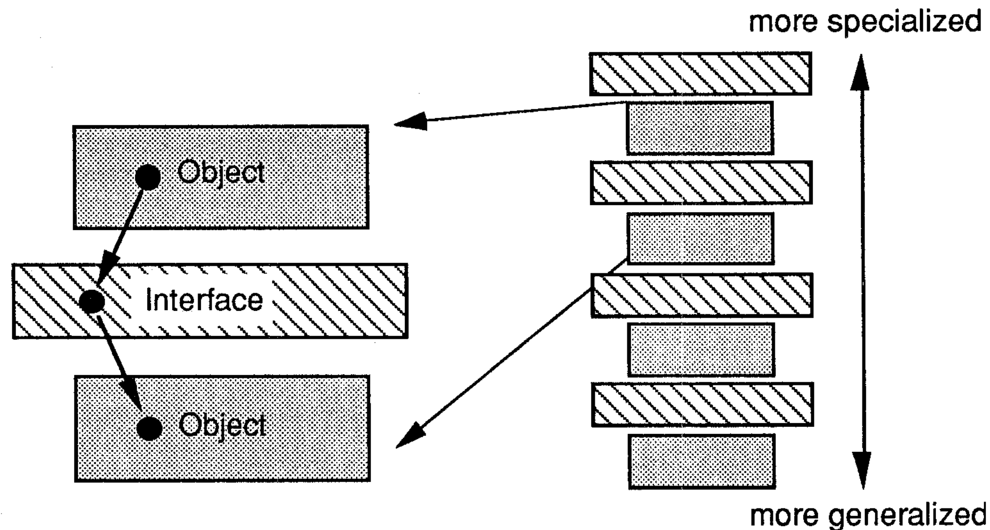


Figure 3.

I.e. take a class BUILDING with an attribute 'id' of type STRING. At run-time this class may be instanciated while setting the value of 'id' to 'office' which can be denoted as BUILDING ('office'). Other instances could be BUILDING('hospital'), BUILDING ('shopping_centre'), BUILDING('house'), BUILDING('car_park'), etc. Each instance can be considered as a specialization of building.

This still leaves the problem where to draw the line. The behaviour of the system will lie somewhere between the afore mentioned extremes of all layers in software or all layers in data. But actually, we do not like this choice at all. However, specialization by instanciating seems to be a very powerful (and elegant) concept. It would be excellent to introduce this concept between *all* layers in the Specialization Dimension. This principle of the *shifting* source/data line will be elaborated in the next section.

## 4. THE EVOLUTIONARY APPROACH

Firstly, to avoid any confusion, the distinction between class and object, in the object oriented context[2], must be clarified. Here the view of Meyer [10] is used: A class is a type; an object is an instance of the type. Furthermore, classes are a static concept: a class is a recognizable element of program text. In contrast, an object is a purely dynamic concept, which belongs (...) to the memory of the computer, where objects occupy some space at run-time (...). The distinction is just the same as that between a program and its possible executions, between a type and the values of the type, or more generally between a pattern and its instances.

To realize the idea of the shifting source/data (class/object) line, as mentioned in the last

---

2 Here is chosen for a clear distinction in contrast with languages like Smalltalk, which treat everything, including classes, as objects.

section, two transformation procedures must be installed:

- From class to object. This is easy because this function belongs to the elementary facilities of an object oriented programming language.
- From object to class. This is less straightforward. This transformation functionality must be added somehow.

What must be done to convert data into source or an object into a class? Of course, without a clearly defined goal this transformation has no rationality. Here, the goal is the implementation of the Specialization Dimension over a finite set of layers. As we have seen, instanciating a class into an object can be interpreted as a way to implement the Specialization Dimension for one step. An implementation for $n$ steps demands for each step to upgrade the object specializations into class specializations. Each object must therefore have a feature to generate a class text out of its current run time state. It is only a matter of efficiency to define this characteristic for self reproduction in a common ancestor inherited from by all classes in the modelling system.

Let us call this common root class: OBJECT. Now the first layer can be defined by instanciating this OBJECT class. A simple, but realistic, example model for this first layer is a data model for graphs. In this simple model we will only distinguish nodes and links between nodes. The function for instanciating is called 'create'. OBJECT has an attribute field called 'id' to identify individual instances, an attribute field called 'generator' to refer to the class where this object is an instance from while it optionally may reference two lists for property definitions and procedure definitions respectively. Properties reflect the state of an object, i.e. each property has a value, while procedures operate on this state, changing one or more property values.

The first step, from class to object, is shown in figure 4. Executing twice OBJECT's create procedure (and initializing the 'id' and 'generator' fields) results into two objects: OBJECT('node') and OBJECT('link').

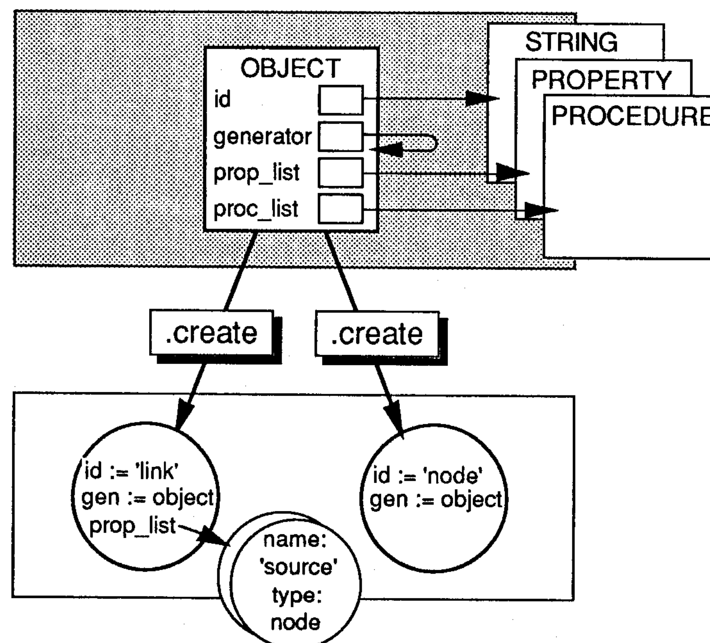


Figure 4.

For the second step we need a generate function to map the run time states of OBJECT('node') and OBJECT('link') onto two class definitions: class NODE and class LINK. This

generate function has several components (figure 5):

- the 'id'-field becomes the name of the class,
- the 'generator'-field is used to compose a proper inheritance clause,
- the lists of newly added property definitions and procedure definitions result in an equivalent number of separately accessible class features.
- explicitly set attribute values are interpreted as redifinition's.

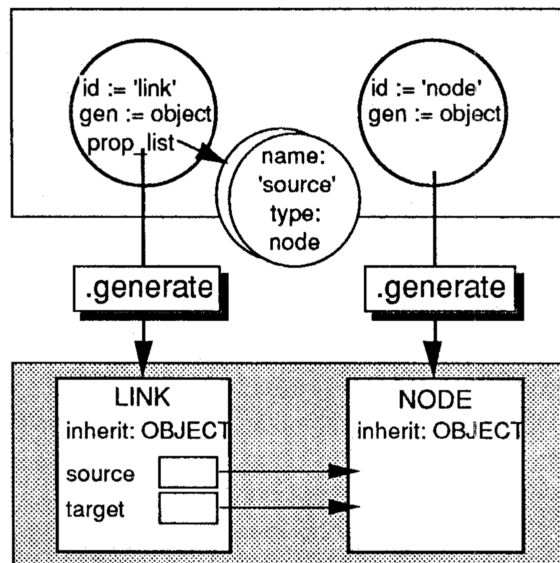


Figure 5.

This last component of the mapping function needs some clarification. An important item of the object oriented paradigm is polymorphism. Polymorphism means the ability to take several forms. Within the O-O context, this refers to the ability of an attribute to refer at run-time to instances of a restricted set of classes. In this graph model example the 'source' attribute of link-objects may refer to node-objects or descendants from class NODE. By redefining the attribute type we are able to constrain this behaviour for descendants of class LINK.

Say this graph model is specialized, in another layer, into NODE('railway_station'), NODE('airport'), LINK('railway_line') and LINK('air_line'). In that case we want to restrict polymorphism to avoid an air line connected with a railway station. This can be achieved by setting the source (and target) value of LINK('air_line') to 'airport'. The mapping function uses this information to redefine the attribute type of source (and target) of class AIR_LINE to (subtype of) class AIRPORT.

Of course, there is much more to say (and to find out) with regard to the mapping function. Here, we have restricted ourselves to the main principles.

## 5. EXAMPLE LAYERED FRAMEWORK

The concept of layered modelling imposes no restrictions to the number of layers. But, at least while we are still in the R&D stages, it is probably wise to introduce some predefined levels in this stack. The next example [8] is intended as a proposal for those predefined levels.

### A. Object layer

On top (or bottom if you like) a layer to implement all required general object features (e.g. for data storage, data interchange and user interface communication) and to define new layers as collections of objects and to convert them into collections of classes afterwards.

**B. Graph layer**
A set graph classes (graph, node, link) to organize and access ever repeating structures as subsystems (decomposition), aspectsystems (views) and all kinds of networks (topology).

**C. Reference model layer**
A layer that contains rather abstract models (e.g. meta models) to define generic product design and production properties. Examples are: Product Structure and Configuration Model (PSCM) and General AEC Reference Model (GARM) both developed within STEP, the IMPPACT Global Reference model and the NEUTRABAS model for ship structures, both developed within ESPRIT projects.

**D. Product type layer**
Consists of clusters of models each dealing with a product family and developed as instances of the afore mentioned general reference models. Product type models contain general knowledge about the product class. It seems only logical that this layer itself will eventually contain a hierarchy of sub-layers. Examples are: the steel structure model [3] and the road design model [WIL90], both based on GARM.

**E. Product model layer**
Not a real layer because it is only implemented in data. On the other hand it seems reasonable to expect that part of all product type models will evolve from this layer in a bottom up model development process.

## 6. CONCLUSIONS

The paper discusses a framework for evolutionary development of large, integrated information models (i.e.. like ISO/STEP). The framework is based on the principles of layered modelling. The main advantages of layered modelling are: improvement of the model integration process and simplification of the model development process. The framework described supports:

- Concurrent and decentralized model development,
- Evolutionary model development by specializing more generic and already integrated models,
- Implementation of those systems, using the O-O paradigm and a shifting class/object boundary interpreted as specialization.

As an example, a five layered framework, currently being implemented in Eiffel, is presented. This implementation (called PMshell, from Product Modelling shell) allows users to develop and implement their ideas on different layers. Product models are derived as instanciations of product type models, to which shape and material information are added. Product type models (or project type models [9]) can be described in the product type layer as specializations of a reference model specified at the Reference model layer. Reference models, tailored to the requirements of the product type model, can be described in the Reference model layer as specializations of the Graph layer and the Object layer.
Specially interesting is the implementation strategy followed. Instances of one or more classes of objects in a layer form a new class of complex objects in the layer below.

## 7. REFERENCES


1 M.H. Böhms, 'Reference Models for Industrial Automation', PhD thesis, TU-Delft, June 1991.

2 W. Danner, 'Generic Product Definition Resource', ISO document, April 1991.

3 J. de Gelder, 'A Steel Structural Application of GARM', TNO Report BI-89-133, July 1989.

4 J. de Gelder, 'A GARM based logical product model for steel structures', TNO Report BI-89-214, December 1989.

5 W. F. Gielingh, 'General AEC Reference Model', document ISO TC184/SC4/WG1 N329, October 1988.

6 W.F.Gielingh, W.J.de Bruijn, H.M.Böhms, A.Suhm, R.Cremer, J.Bassan, 'An Architecture for Open Systems Information Integration of CIM Modules', Proceedings CAPE'91 Bordeaux France, September 1991.

7 J. R. Kirkley, B.K. Seitz, 'STEP Framework, Concepts and Principles', March 1991, ISO TC184/SC4/WG5/P1 Draft 2.

8 G.T. Luiten et al., 'Development and Implementation of Multi-Layered Project Models', second international workshop on Computer Building Representation, Aix-les-Bains, France, June 91.

9 G.T. Luiten and F.P. Tolman. 'Project Information Integration for the Building and Construction Industries', proceedings CAPE '91 Bordeaux.

10 B. Meyer, 'Object Oriented Software Construction', Prentice Hall, 1988.

11 P.H. Willems, 'Road Model Kernel', TNO Report B-89-831, January 1990.