Setting objects to work: adding functionality to an architectural object model

Ann Hendricx and Herman Neuckermans

K.U.Leuven
Faculty of Applied Sciences
Department of Architecture
Heverlee
BELGIUM

ABSTRACT

Several research initiatives in the field of product modelling have produced static descriptions of the architectural and geometrical objects capable of describing architectural design projects. Less attention is paid to the development phase in which these static models are transformed into workable architectural design environments. In the context of the IDEA+ research project (Integrated Design Environment for Architectural Design), we use the object-oriented analysis method MERODE to develop and describe both an enterprise (or product) model and a functionality model. On the one hand, the enterprise model defines the architectural and geometrical objects, their methods and their relation with other objects. On the other hand, the functionality model organizes the functionality objects – ranging from single-event objects to complex-workflow objects – in a layered and easily expandable system. The functionality model is created on top of the enterprise model and closes the gap between the static enterprise model and the dynamic design environment as a whole. After a short introduction of the envisaged design environment and its underlying enterprise model, the paper will concentrate on the presentation of the higher-level functionality model. Elaborated examples of functionality objects on the different levels will clarify its concepts and proof its feasibility.

1 INTRODUCTION

A uniform building representation can be of great help during the entire course of the design process. Ideally this building representation is located in an integrated design environment (Neuckermans 1992). For the architectural designer (or team of designers) on the one hand, he or she can call the assistance of modelling and evaluating tools situated in the environment. And by using the computer from the early stages in the design process, the unproductive translation of a design made by hand into a digital version would be avoided.

For all partners involved in a building's life cycle on the other hand, they can use and contribute to one single building representation (see Figure 1), instead of producing their own. Next to avoiding the waste of time and manpower, the communication between the building partners will improve and inconsistencies between models will be avoided. To make this possible, the digital building representation should be located at a central and accessible place (using web technology).

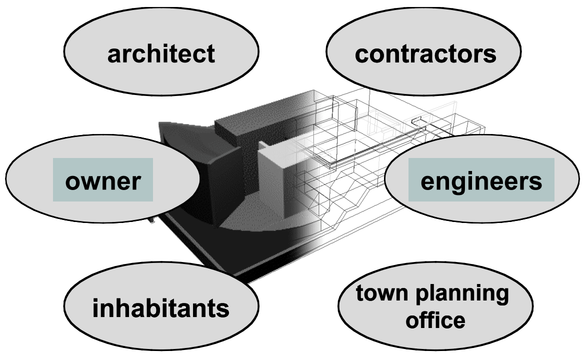

Figure 1: **Central position of the central building description** (Hendricx et al. 1998)

2 USING AN OBJECT-ORIENTED ANALYSIS METHOD

Several research initiatives in the field of product modelling have produced static descriptions of the architectural and geometrical objects capable of describing architectural design projects. Less attention is paid to the development phase in which these static models are transformed into workable architectural design environments. In the IDEA+ research project (Integrated Design Environment for Architectural Design, Hendricx et al. 1998, Hendricx 2000) emphasis was laid on the systematic development of both phases. This particular goal was achieved by leaning on the object-oriented analysis method MERODE (Model-driven Existence-dependency Relationship Object-oriented Development, Snoeck et al. 1999).

The authors of MERODE detect two major problems in the object-oriented analysis approach; problems for which their method tries to offer a solution.

The first problem is the lack of higher-level modularity of object-oriented systems. When the size of a project increases, one can easily end up with one complex heap of communicating objects that, despite the basic object-oriented concepts that enhance maintainability, is not easily maintained. Some structure is needed. To solve this problem, MERODE defends an additional *model-driven approach*. The complexity of information systems is driven by the complexity of the underlying real world. By giving special attention to modelling the real world prior to the specification of the desired functionality, the complexity of developed systems is more manageable. Analysis objects are therefore systematically partitioned in objects of the problem domain and objects that embody functionality. The result is an analysis model that consists of two submodels: an enterprise model and a functionality model (see Figure 2). The enterprise model acts as a solid kernel on top of which functionality is modelled, leading to a set of independent function object classes. For both modelling phases specification techniques are clearly defined.

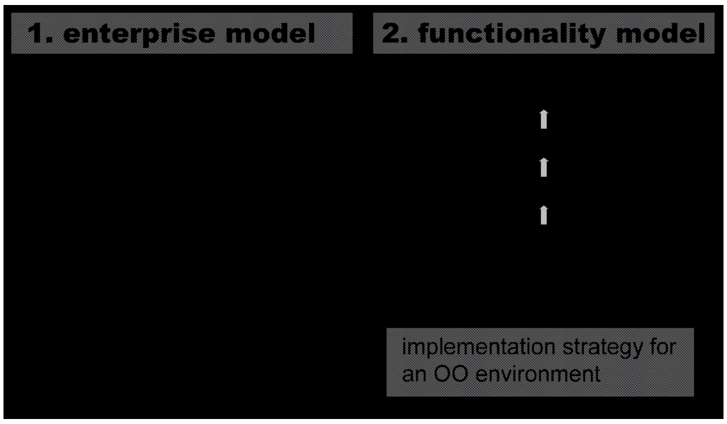

Figure 2: **Enterprise model and functionality model**

The second problem is the lack of formality: current object-oriented analysis methods are often based on vaguely defined concepts. Hence, they provide little help for checking internal consistency (for instance between separate schemes used to manage complexity) and quality of specifications. Next to a clear formal definition of modelling techniques, the concept of *specification by contract* is an answer to this quality control problem (Snoeck et al 1999).

3 THE ENTERPRISE MODEL

The first column in Figure 2 shows the different steps to be taken during enterprise modelling. As in most product modelling research projects, pointing out the relevant **enterprise object types** and the static relationships between them leads to a model apt to give a static description of a design project. Such a core object model for architectural design is presented in (Hendricx 2000). Prototypes of this model have been validated by means of actual design cases.

In addition, **event object types** describe the fundamental behaviour of the enterprise object types. The associations between enterprise object types and event object types are represented in an object-event table (OET). Figure 3 shows a part of the OET of the above-mentioned architectural core model. For every event object type (rows in the OET), the participating object types (columns in the OET) are indicated. Strict rules (e.g. the propagation rule for existence dependent object types, meaning that a dependent object type should not be involved in an event object type without its master object type having knowledge of it) further determine when and how an object type participates. For every participation, an object method carrying the same name as the event object type becomes part of the participating object type's class definition.

O/C
O/M
O/M
O/E
A/M
A/M
A/M
CREATE_CAAD_ENTITY
A/M
A/M
O/C
A/M
A/M
A/M
A/M
A/M
A/M
A/M

Figure 3: **Linking enterprise objects types and event object types in the object-event table**

```
deferred Class      CAAD-ENTITY

Inherit
-- list of ancestors
Attributes
-- references to mother object types
            ref_element: ELEMENT
-- references to marsupial object types
            ref_local_coordinate_system: LOCAL-COORDINATE-SYSTE
            ref_representation_link : SET ( REPRESENTATION-LINK)
            ref_dependence_link: SET (DEPENDENCE-LINK)
-- State indicators
            state: INTEGER
-- additional attributes
            id: INTEGER
            name: STRING
Invariant
            unique {id}
Methods
-- methods for atomic event types

            create_caad_entity (Input_id: INTEGER; Input_name: STRIN
            Input_element: ELEMENT)is
                 require
                      Input_element  /= Void
                 do
                      state := 1
                      id := Input_id
                      set_caad_entity_name (Input_name)
                      ref_element := Input_element
                 end

            set_caad_entity_name (Input_name: STRING)is
                 require
                      state = 1
...              do
```

relation with other object classes

attributes and constraints

object methods

Figure 4: **Object class definitions as the final result of the enterprise-modelling phase**

These class definitions (of both enterprise object types and event object types) gather all information acquired in the previous modelling phases. In the last enterprise-modelling step, these class definitions are further refined by adding attributes and constraints, and by elaborating the object methods. An example of such a class definition – described in MERODE's Eiffel-inspired pseudo code – is given in Figure 4. The class definitions are the final result of the enterprise-modelling phase.

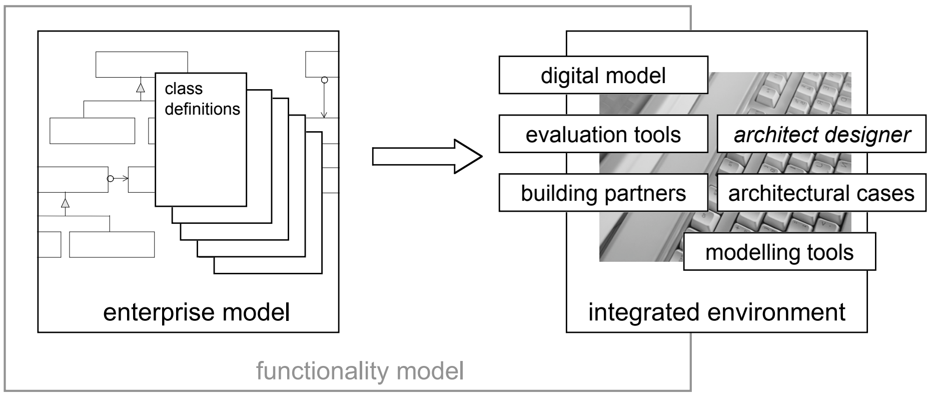

Figure 5: **The functionality model closes the gap between the enterprise model and the design environment**

4 THE FUNCTIONALITY MODEL

4.1 **Towards an implementation strategy**

Due to its model-driven approach, MERODE separates objects of the problem domain (the real world objects defined in the enterprise-modelling phase) from objects that embody functionality. The functionality model organizes the functionality objects – ranging from single-event objects to complex-workflow objects – in a layered and easily expandable system. It is created on top of the enterprise model and closes the gap between a static enterprise model and the dynamic design environment as a whole (see Figure 5). For instance, a daylight tool situated in the environment can communicate with the core object model through functionality objects elaborated in this phase. From now on, the model's description is no longer completely independent from its later implementation. The MERODE method presents implementation schemes towards both object-oriented technology and more legacy technologies. In the remainder of this paper, the presented functionality model follows an object-oriented implementation strategy, since this is the route we are taking for our integrated design environment.

4.2 **MERODE's layered approach**

Figure 6 presents MERODE's layered system architecture. Higher-level events and functionality are composed of events of a lower level, leading to a system of cascading event definitions. The basic behaviour of the enterprise objects is located at the two lowest levels and is described in the enterprise model. The object methods - defined in the class definitions of the enterprise object types - are located in the Methods Layer. In the Atomic Event Layer, an atomic event (a row in the OET table) triggers all methods with the same name by broadcasting its message across all the enterprise objects.

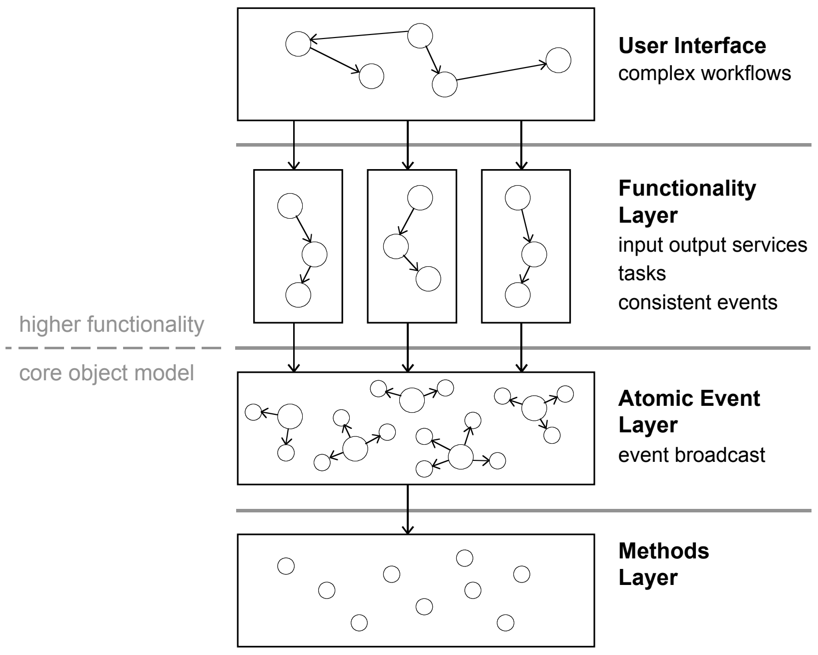

Figure 6: **Functionality in MERODE's layered system architecture** (Hendricx 2000)

In the Functionality Layer we find:
- consistent events: a mandatory series of atomic events to keep the object database in a consistent state
- tasks: a voluntary series of atomic events to improve efficiency
- input and output services: composed of input and output functions taking care of the communication between the core model and the outside world

At the highest level MERODE situates the complex workflows, which make up the interface between the user and the software system.

4.3 **Description of a functionality object type**

The techniques for describing a functionality object are illustrated by an example: the input and output service TRANSFORM-ENTITY (Hendricx 2000). This function can be used by an application program built on top of the core object model, to change an entity's geometry or topology, for instance by a rotate, move, scale or stretch operation. Such an operation may start a series of cascading events on all architectural objects involved.

The final goal of the functionality-modelling phase is a collection of class definitions describing the functionality objects. As for the description of a function object, it should be as complete as possible to improve later implementation.

Class TRANSFORM-ENTITY

Attributes
-- references to participating object types: …
-- additional attributes: …

Methods
• initiate_transform_entity (Input_graphical_entity: GRAPHICAL-ENTITY) **is** ...
• check_validity **is** ...
• run **is** ...

-- Different options:
if solitary graphical-entity: perform transform operation on graphical-entity
else: define caad-entity and all connected caad-entities
(via dependence-link, mandatory boundary-link)
if physical-element: perform operation on physical-element-type
else: perform operation on caad-entity
The transform operation is conducted on the appropriate graphical-primitive, after which the geometry_topology attribute is set in accordance with the new situation. This setting triggers the VISUALIZE service.

Figure 7: **Textual description of the TRANSFORM-ENTITY function**

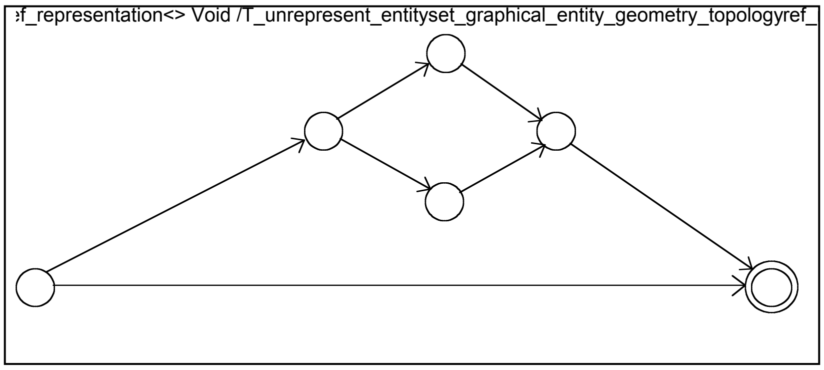
ef_representation<> Void /T_unrepresent_entityset_graphical_entity_geometry_topologyref_

Figure 8: **Final State Machine of the TRANSFORM-ENTITY function**

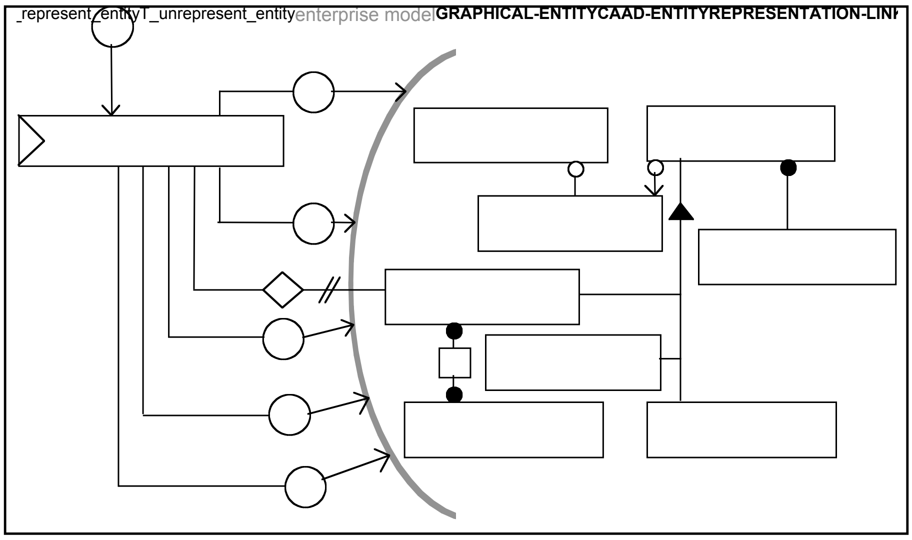
_represent_entityT_unrepresent_entity
enterprise model
GRAPHICAL-ENTITYCAAD-ENTITYREPRESENTATION-LINK

Figure 9: **Service Specification Diagram of the TRANSFORM-ENTITY function**

Several notation techniques may therefore complement this description:
- a textual description (Figure 7)
- a Final State Machine (Figure 8), pointing out the alternative sequences of events
- a Service Specification Diagram (Figure 9), focusing on the events and the involved enterprise objects

4.4 **Functionality in an architectural design environment**

By following the MERODE methodology, we end up with a layered system of event types, tasks and services. Higher–level event types may be composed of lower level ones. By working with encapsulated objects, a function's contents can be changed with minimal changes in other functionality object types. Figure 10 gives an impression of possible functionality building blocks in the context of an architectural design environment. In this picture, the Functionality Layer of figure 6 is subdivided in three parts to situate the three different kinds of functionality objects types of this layer (see 4.2).

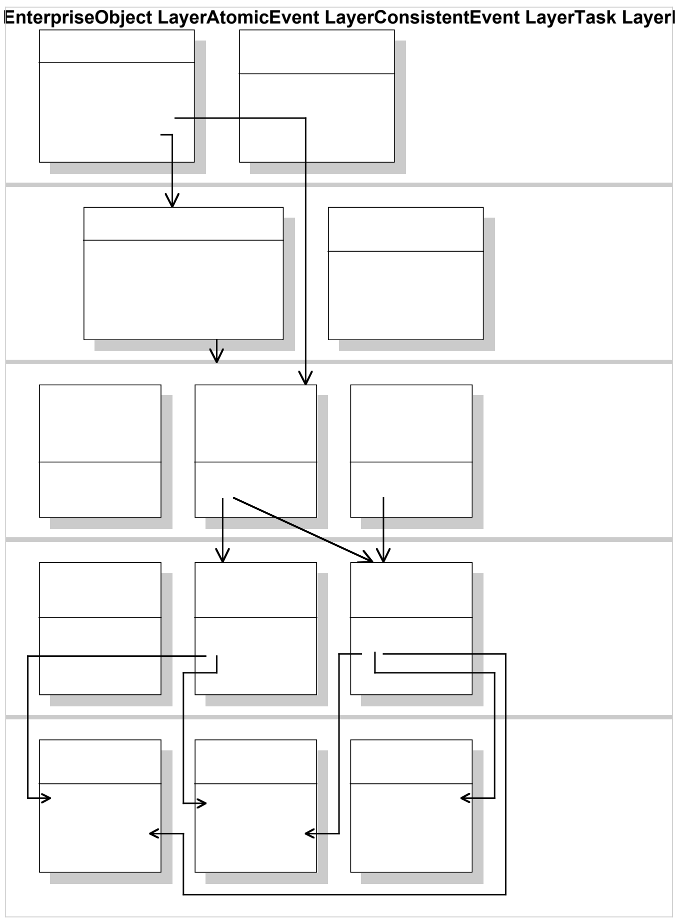
EnterpriseObject LayerAtomicEvent LayerConsistentEvent LayerTask LayerI

Figure 10: **An impression of the functionality built on top of the core object model** (Hendricx 2000)

An event type such as CREATE-TYPE-LINK in the Atomic Event Layer is broadcast across all the enterprise objects in the lower layer and triggers all methods sharing the same name. The same event CREATE-TYPE-LINK, in its turn, is used as a building block to compose the consistent event C-CREATE-CAAD-ENTITY. For efficiency, frequent sequences of consistent and atomic event types can be defined as tasks, such as T-REPRESENT-ENTITY and T-UNREPRESENT-ENTITY.

In the Input Output Layer, a service such as DESIGN-WALL combines lower level tasks (e.g. T-REPRESENT-ENTITY) and consistent events (e.g. C-CREATE-CAAD-ENTITY).

Since one can rely on already existing building blocks, adding new functionality becomes easy. This stepwise-elaborated functionality is a very attractive MERODE feature, especially because adding functionality is often a point paid less attention to in product modelling initiatives.

5 CONCLUSION AND FUTURE DEVELOPMENTS

A systematically developed functionality model is the stepping-stone between a static product model for architectural design and a workable integrated design environment. The use of the object oriented analysis method MERODE leads to the development of a layered and easily expandable system of function objects. As it is now, prototypes of such function objects in the context of architectural design have been elaborated. Following work will include the testing of the above structure by elaborating the functionality for a new natural lighting design tool, IDEA-l (Geebelen 2000).

6 REFERENCES

Hendricx, A., Geebelen, B., Geeraerts, B. et al. (1998) A methodological approach to object modelling in the architectural design process, *Proceedings of the 4th International Conference on Design and Decision Support Systems in Architecture and Urban Planning*, Maastricht, The Netherlands, July 26-29, 1998, CD Rom publication.

Hendricx, A. (2000) A core object model for architectural design, Ph.D. Thesis, Faculteit Toegepaste Wetenschappen, K.U.Leuven, Leuven.

Geebelen, B. (2000) IDEA-l, An Early-Stage Architectural Deisng Tool for Natural Lighting, to be published in the *Proceedings of IBPC2000*, Eindhoven, The Netherlands, September 18-21, 2000.

Neuckermans, H. (1992) A conceptual model for CAAD, *Automation in Construction*, vol 1, nr 1, pp. 1-6.

Snoeck, M., Dedene, G., Verhelst, M. et al. (1999) Object-oriented Enterprise Modelling with MERODE, K.U.Leuven University Press, Leuven.