Paper from proceedings of the CAADfutures'99 Conference Computers in Building (Godfried Augenbroe and Charles Eastman eds) Georgia Institute of Technology, Atlanta, USA 1999.

The BAS•CAAD information system for design – principles, implementation, and a design scenario

Anders Ekholm and Sverker Fridqvist
Computer Aided Architectural Design, Lund University, Lund, Sweden

Keywords: CAD, design, dynamic schema evolution, information systems, object oriented modelling, product modelling, design scenario

Abstract: The objectives of the BAS•CAAD-project are to investigate into theories and methods for computer aided architectural design, with emphasis on requirements of early stages of the design process. Information systems can be characterised as static or dynamic concerning the definition of classes in the model schema, and concerning classification of model objects. The paper presents the BAS•CAAD system, a prototype software that implements the conceptually most important features of a dynamic information system for design. The BAS•CAAD information system is built on a generic ontological framework. The system allows a free combination of attributes, supporting the incremental way that knowledge is built up during design. It provides a generic library structure that allows definition of objects classes in different levels of generalisation that may originate from international standards or the individual designer. For example, in the construction context, it allows modelling of buildings and their parts, as well as user organisations and user activities. The function of the system is illustrated in two scenarios.

1 INTRODUCTION

In a computer integrated construction process, information is generated, used and communicated through computers. In order to enable computer based analyses of the products and processes developed, this information must be model based. This concerns not only building information, but also information about other objects, e.g. the user organisation, the site, the construction process, and the facility management process.

The questions of the structure of building product models and the communication of building product data between different actors and computer systems have been given much attention within construction information research. Analyses of characteristic features of such models in the construction context have been done by Björk (1995) and Galle (1995). Lately, principles for structuring computer based information about the user organisation have been discussed, see e.g. Eastman and Siabiris (1995), Ekholm and Fridqvist (1996 and 1998), Maher, Simoff and Mitchell (1997).

Many approaches to product modelling focus only at modelling, and seem to overlook the process of creating the models. The most outstanding feature of this process is that the information changes and evolves over time, not only in quantity but semantically as well. This would make it hard to use a product modelling system based on a fixed classification schema in the earliest, most dynamic phases of design, since the fixed schema would be at odds with the evolving semantics of design.

This paper presents the implementation of a prototype information system, which has been constructed as part of the BAS•CAAD research project. The project aims to find solutions to both the problem of modelling different products and processes, and the need to reflect and support the evolving nature of the design process.

2 INFORMATION SYSTEMS FOR DESIGN

2.1 Design problems

Design is a problem solving process, it is similar to problem solving both in everyday life and science. Problem solving is a process of exploration, where solutions are alternately hypothesized and tested. In the process, the properties of the hypothesised object are determined in an incremental manner, the designer adds and removes attributes from the conceptual representation of the object.

A problem statement is a description of an object whose state, according to certain presuppositions or objectives, is unknown, or unsatisfactory. The state of an object is its attributes at a certain stage of time. A problem solution, or hypothesis, is information that describes the state of the object, or action that leads to a satisfactory state. Design problems are procedure problems, they deal with questions of human action (Bunge 1998:208). The test of a design hypothesis may be theoretical, relating the solution to existing knowledge, or empirical, involving the construction of an artefact.

A design problem may be characterised as open or closed concerning the determining factors of the designed artefact, e.g. in building design such factors are environmental impact, user requirements and available technology. To a closed problem, the determining factors and their combinations are well known, while to an open problem neither the determining factors nor their combinations are known, but must be explored or invented. Open problems are also called ”wicked” (Rittel 1984).

Design can also be categorised as routine or innovative. Routine design is a closed problem solving process, it consists in selecting a prototype solution and determining the values of its attributes. Innovative design is necessary when no such prototype solution can be applied, and new kinds of things or new uses for known things have to be created. Building design is an example of both routine and innovative design, the latter especially during early stages. The approach to building product modelling today, e.g. in object-oriented CAD programs, does not support innovative design and is best suited for the later stages of the design process.

2.2 Dynamic and static information systems

An information system is a computer based system which makes it possible to store and retrieve information of relevance to the information needs of a user. It consists of a conceptual schema, an information base and an information processor (ISO 1985:15). In a traditional implementation, the classes of the conceptual schema refer directly to the objects in the domain of discourse. A specific model of an object is achieved through selecting an appropriate class in the schema and determining the values of the attributes that describe the object in the information base.

In the BAS•CAAD project, we have found that information systems can be characterised as dynamic or static concerning the possibility for the user to 1) define new class concepts in the conceptual schema, and 2) classify model objects. These two characteristics are mutually independent, see Figure 1.

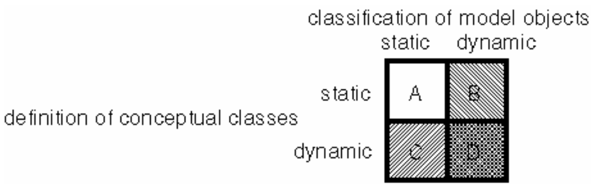
classification of model objects
static
dynamic
definition of conceptual classes
static
dynamic
A
B
C
D

Figure 1: Dynamic and static information systems

The four kinds in Figure 1 are:

a) Static systems: the user is restricted to a predefined set of modelling classes, model objects have to retain their classification once instantiated into the model.

b) Dynamic classification: the user is restricted to a predefined set of modelling classes, but can reclassify model objects between these classes during modelling.

c) Dynamic schemas: the user can create new classes, but cannot reclassify model objects during modelling.

d) Fully dynamic systems: the user is free to create new classes and to reclassify model objects between all classes, predefined and new, during modelling.

The literature on information systems, e.g. (ISO 1985), describe systems that belong to category A, but the terminology and theory can be used for all kinds of system in Figure 1. A static, or closed, classification is often suitable for a routine design process, which presupposes a high degree of detailed knowledge about the domain of discourse; however, it is not suitable for a more search-like innovative design process.

2.3 The BAS•CAAD approach to dynamic modelling

Through developing a design schema (see section 0) that defines and relates classes that only indirectly and in a generic way refer to the domain of discourse, it is possible to create a dynamic modelling system. Instances of these "meta"-classes are used for the development of model schemas that describe and directly refer to the members of the domain of discourse.

The domain of implemented meta-classes is orthogonal to (independent of) the domain of runtime data or instances. The user of an information system has access only to the latter; the former is available to the system developer only. In the static approach, the model schema resides in the domain of implemented classes, and thus is not open for user manipulation. In the BAS•CAAD system, we have made possible both dynamic classification of design objects and dynamic definition of classes by 'sliding down' the model schema from the domain of implemented classes to entirely reside in the domain of instances, see Figure 2.

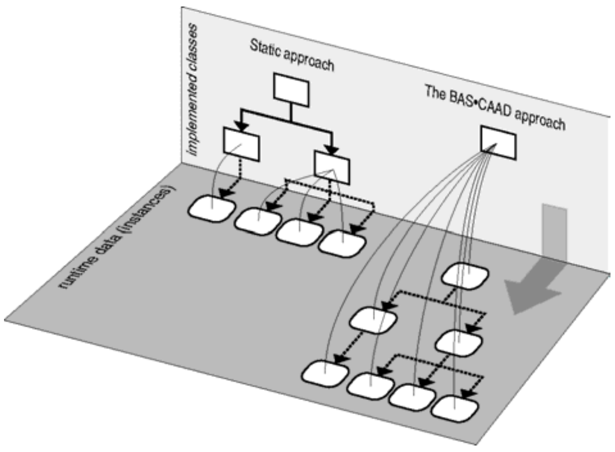

Figure 2: The model schema is 'slided down' into the domain of instances. Arrows symbolise superclass-subclass relationships; arcs symbolise instantiation

The traditional static or hardcoded approach is not in every respect inferior to the dynamic approach; one of its advantages is that it is easier to ensure consistency in a fixed class structure. Dynamic systems, such as the BAS•CAAD system, have to implement mechanisms for dynamic consistency checking. Also, since in the static approach a definite set of classes is implemented, only operations for managing these classes are needed . In contrast, a dynamic system must provide operations on a generic level, which is a much more complex task. The static approach is especially fit in situations where the modelling context is very specific and beforehand well known, such as creating information systems for routine design.

A more thorough discussion of information systems for design can be found in (Ekholm and Fridqvist 1998). There, we defined that information systems for design must: 1) support representing objects in the domain of interest, 2) be able to communicate with other software, 3) support defining the design goal, and 4) have a dynamic object structure.

Most proposals for product modelling software focus on the first two requirements. To be useful not only for describing the results of the design process, but as a tool in this process, an information system for design must have all properties mentioned above. This is also discussed by e.g., Eastman and Fereshetian (1994), Eastman, Assal and Jeng (1995), Galle (1995), Junge, Steinmann and Beetz (1997), Leeuwen and Wagter (1998), and Ekholm and Fridqvist (1998).

3 FOUNDATIONS FOR THE BAS•CAAD INFORMATION SYSTEM

3.1 The ontological framework

The BAS•CAAD information system is intended to cover all levels of generalisation of modelling in the construction context, from international classification standards to specific buildings. It is built on a generic ontological framework including systems and property theory, with the object classes *thing class*, *relation* and *unary attribute* see Figure 3. It allows models to be multi-contextual; that is, several contextual views can be co-ordinated in one model. For a detailed account of this framework, see Ekholm and Fridqvist (1998). The BAS•CAAD information system supports generic design operations, like generalising and specialising, aggregating and decomposing, and adding and removing attributes; for an account of design operations, see (Fridqvist and Ekholm 1996).

In Figure 3, the ontological framework is depicted as an EXPRESS-G diagram. The figure differs from the one in (Ekholm and Fridqvist 1998) in that the entity Attribute is now named UnaryAttribute. In addition, the superclass BAS_CAAD object is removed.

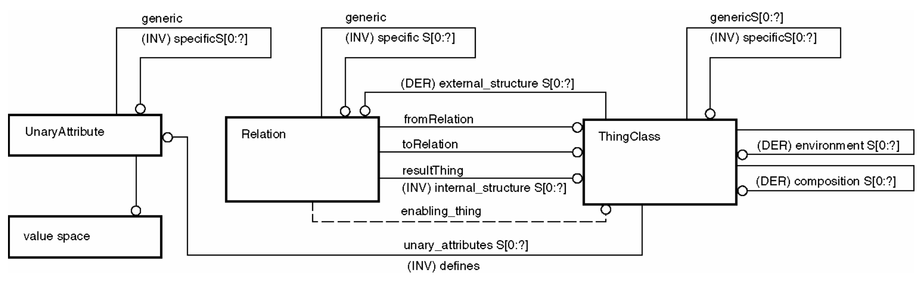

Figure 3: Ontological framework of the BAS•CAAD information system

3.2 The design framework

Artefact design can be characterised as making statements about the designed thing and a desired 'satisfactory situation' that the artefact is to support. Statements, or predicates, consist of attributes of different kind; in case of design, these represent properties of the designed artefact. The classes in conceptual schemas are such statements or predicates, made up of one or many attributes. For a more detailed account of this, see (Ekholm and Fridqvist 1998). An information system for design should accordingly at least be able to record such statements. The statements, of course, need not be verbal. Also 2D drawings or 3D models or parts thereof can be treated as statements. The BAS•CAAD information system is designed to collect such statements and to support analysis and communication of them.

A model is defined through a consistent collection of correct statements involving the three basic classes of the ontological framework in Figure 3. The schema in Figure 3 is not suitable for a direct implementation as a design tool, since it implies that all connections between objects must be established. The trouble with this is that some connections might imply statements that the designer would not intend. For example, some things are used in numerous different environments; it would not be desirable to have to include all these environments in the definitions of thing classes like 'bolt' or 'nut'. Instead, we have found that all information necessary for the creation of such a more suitable design schema is contained in the set of design statements made possible through the schema in Figure 3.

The attributes of the entity ThingClass in Figure 3 reflects that it can be defined as a 6-tuple of sets of attributes, T = (T_G, T_C, R_I, T_E, R_E, A_U). Given the BAS•CAAD ontological framework, only seven different types of statements are possible. The first is the statement "there is a kind of things, called T"; this corresponds to creating ThingClasses. The other six statements implies to add attributes to the six sets mentioned above:

- T_G: A T-thing is a kind of Y-thing (= kind T is a subkind or specialisation of Y).
- T_C,: A T-thing is composed by a P-thing part.
- R_I,: A T-thing is internally structured, so that any part of kind P_1 is related to any part of kind P_2 by an R_I -relation.

- T_{E},: A T-thing has an environment which includes an E-thing.
- R_{E}: A T-thing is related by an R_{E}-relation to any E-thing in its environment.
- A_{U}: A T-thing has the unary property Q.

The BAS•CAAD design schema in Figure 4 has been developed for the data structure to be implemented. Design data are stored in instances of *ThingClassDefinition*, which collects instances from the six subclasses of *Attribute* that correspond to the six different kinds of attributes. The classes *RelationDefinition* and *UnaryAttributeDefinition* define relations and unary attributes, and ensures that attributes referred to in several places are identically defined.

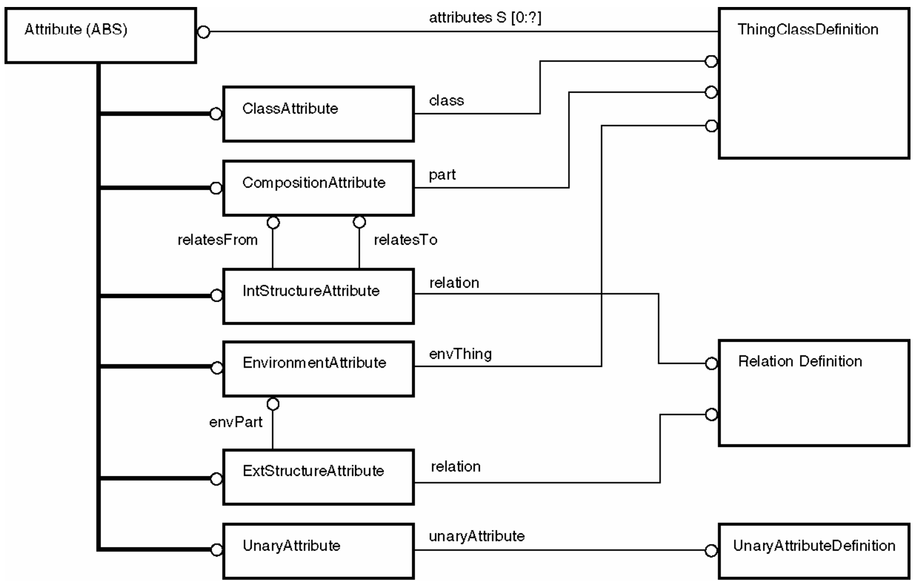

Figure 4: Object structure of the BAS•CAAD design schema

3.3 Other features of the object structure

Besides generating and maintaining the object structure, the prototype currently supports a simple schema for object identification, inheritance and specialisation of attributes, and class libraries. Additionally, class subsumption is planned to be implemented.

3.3.1 Object identification

The BAS•CAAD information system for design allows thing classes to be defined by references to predefined libraries (see section 4.3). Such a mechanism requires that libraries and classes can be unambiguously identified, in order to ensure that the correct libraries are used. The current version can identify objects within a library, but there is no secure identification of libraries. The development of such a mechanism is a task for database specialists and international standardisation organisations, and outside the scope of this phase of the BAS•CAAD research project.

3.3.2 Attribute inheritance

Class attributes indicate a more generic kind, also called a superclass. All statements defined for the superclass are also valid for the subclass; the attributes of the superclass are inherited attributes of the subclass. An example: the class House has the attributes 'man-made' and 'provides dwelling'. If Palace is defined by the

	House	Palace
Defined attributes	man-made provides dwelling	kind of House
Inherited attributes		man-made provides dwelling

Figure 5: Defined and inherited attributes

attribute 'kind of Building', Palace inherits the attributes 'man-made' and 'provides dwelling' (Figure 5) A mechanism for attribute inheritance has been implemented.

3.3.3 Attribute specialisation

However, with the above set of attributes, the Palace class doesn't reflect that a palace is a specific kind of house. This type of difference can be defined in to ways; either by adding a new attribute to the specific class, or by specialising an inherited attribute. Which way to choose is a question of clarity in modelling and usefulness of the model; se Figure 6 for a illustration of the differences of the two methods in terms of sets.

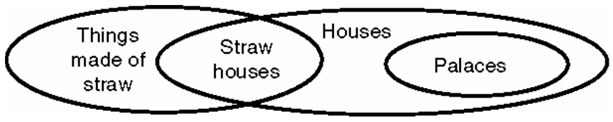

Figure 6: Classes and subclasses

In this case, we have chosen to specialise the attribute 'provides dwelling' into the more specific 'provides luxurious dwelling', and to define the class Palace with the specialised attribute. A mechanism for this has been implemented. The new attribute is defined to substitute the inherited attribute 'provides dwelling' in the Palace class, Figure 7.

	House	Palace
Defined attributes	man-made provides dwelling	kind of House *provides luxurious dwelling*
Inherited attributes		man-made ~~provides dwelling~~

Figure 7: Attribute specialisation

3.3.4 Class subsumption

Description logics (see next paragraph) provides a mechanism called subsumption. This could be described as the inverse of inheritance, since it answers what superclasses a given class has according to its set of attributes. For instance, if a thing Bungalow has the attributes ‘man-made’ and ‘provides dwelling’, then it is subsumed to be a kind of House, i.e. the class Bungalow is a subclass of the class House. Subsumption is not currently implemented in the BAS•CAAD prototype system, since it would make implementing inheritance more complicated. However, since subsumption could arguably provide a powerful tool for case retrieval, it will probably be implemented in a future version of the BAS•CAAD system.

Description logics (DL) is a field within artificial intelligence research, and is based on first order predicate logic. It aims to develop mechanisms for describing concepts, and to automatically classify concepts. . DL system implementations are usually similar to programming languages (Lambrix 1996). Although DL is similar to the BAS•CAAD approach in many ways, it seems that it cannot serve as the base for a design information system.

4 THE BAS•CAAD PROTOTYPE INFORMATION SYSTEM

An objective of the BAS•CAAD project is to support expression of objects in a computerised database during design. The prototype is intended to study the feasibility of organising an information system based on the BAS•CAAD schema.

The prototype displays some features we consider important in an information system for design, but it is not intended for productive design work. The current version is centred on building symbolic schemas for concepts referring to things, and lacks most of the abilities to specify measurable values necessary for a production tool.

4.1 Smalltalk

The BAS•CAAD prototype information system is currently implemented in Smalltalk under the Macintosh operating system. The reason for choosing the

Smalltalk computer language was that it is object oriented, and that it supports explorative program development.

In product modelling, products are structured as objects that are assembled of objects in several levels. This makes it natural to choose an object oriented programming language for implementation. In explorative program development, the software needs only to be partly defined before it is executed and tested. Thus, experimental solutions can be tried and kept if successful; otherwise, they are discarded. This way the final software solution is obtained through exploration.

In Smalltalk, code segments can be run and tried directly, without any time-consuming compilation or linking procedures. Actually, Smalltalk allows the programmer to change the code while the software is running, thereby relieving the programmer from repeatedly punching in lots of test data. This is not possible in C++, a popular programming language that also supports object oriented programming. Smalltalk is profoundly object oriented; every concept is treated as an object class. Thus, in Smalltalk it requires an effort to not be object oriented, as opposed to C++. A drawback with Smalltalk is that the resulting programs do not run as fast as those created with e.g. C++.

4.2 Implementing the system

The foundation for the BAS•CAAD prototype is the three concepts thing class, relation and unary attribute that are implemented through the corresponding object classes *ThingClassDefinition*, *RelationDefinition* and *UnaryAttributeDefinition*.

To create and manipulate these objects, there are, in principle, two choices: a command line user interface or a graphical user interface (a GUI). In a command line user interface, the user types commands, and then usually hits a key to get the command executed. A command line interface requires developing a command line interpreter, a piece of software that can translate the command lines into software actions and input data. A problem in software design is to create software that correctly interprets the user's intentions. A GUI features a set of standardised tools, each having a specific function. The user controls the software via the buttons, menus, text input fields etc. Thus, the software developer can restrict the user to input only such instructions or data that are valid.

Regarding these concerns, the choice fell upon a GUI as the method to interact with the software. The Smalltalk dialect used for the BAS•CAAD prototype (Smalltalk Agents, STA), provides generic classes of GUI components, such as windows, menus, buttons etc. These can be subclassed and modified to suite the particular needs of the software being developed.

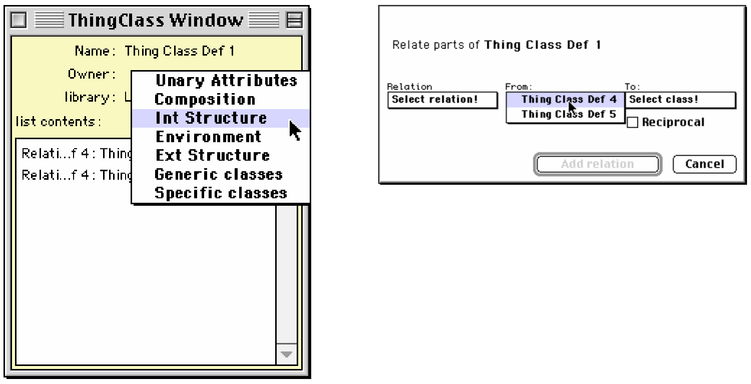

Figure 8: Examples of user interface windows

Windows communicate with the corresponding model objects through messages. Each user interaction with a window causes at least one of the window's methods to be executed. This results in messages being sent to the model object, which responds with the appropriate actions. Finally, some messages are sent from the model object back to the window, to ensure that the window reflects the new state of the model object.

In the current BAS•CAAD implementation, there are windows for the three main classes, and in addition to this, there are windows for specific tasks, such as adding different kinds of attributes to thing class definitions, see Figure 8. Most of the work with the development of this prototype system has been, and continues to be, to decide how to obtain the desired functionality. To perceive a user action in the terms of objects exchanging messages is a complex task. One particularly difficult question is to decide what object is the agent, and what objects are acted upon. The reason for this is that many objects may be involved in the execution of one user action, with many messages exchanged. If the resulting web of such message interchanges is too entangled, the programmer is likely to get lost. Additionally, there is a need to make the software code as intuitive as possible to any future programmer (which, incidentally, often is the same person, but a couple of days later).

4.3 Libraries

The purpose of the BAS•CAAD system is to record information about concrete things during the design process. Acknowledgeably, defining a coherent conceptual understanding of the concrete world from scratch is a very demanding task. To allow the BAS•CAAD system to be used in normal design situations, pre-defined well-considered schemas of Thing Classes have to be provided in libraries, from where the designer can fetch definitions to build up his design database.

In order for the BAS•CAAD system to function in the construction context, it is necessary that its libraries contain attributes and classes belonging to established building classification systems. Today, these systems are not structured for use in the early design stage. The classes are mostly combinatory, i.e. built up from combinations of several attributes. One example of such a classification system is the BSAB system, a de facto Swedish construction industry standard. In order to implement a classification system into a system for design, including the earliest stages, it is necessary to make the constituent simple attributes availlable. The BSAB system would be the natural choise for analysis for an application to the Swedish context.

Principles for structuring product libraries are developed as part of the STEP standardisation activity, the so called Parts Lib work (ISO 1997). An example of the development of an operational class library is POSC/Caesar with its Reference Data Library for documentation of facilities for the oil and gas industries (POSC/Caesar 1999).

The current BAS•CAAD prototype provides an ontology for and supports class libraries. Objects in libraries can refer to objects in other libraries, so that hierarchies of libraries can be created. This feature is implemented to explore a mechanism that would allow international and national standardisation organisations to create generic libraries. Clients, building material providers, and others can then create intermediate level libraries that can be used as references in specific building projects.

The concept of class library has been generalised, so that library is the sole format for databases and files. Thus, the work of any level of specificity can be the foundation for further specialisation. For instance, it would be possible to use the project file from a window designer as a library file in a building design project without any reformatting or reclassification.

5 TWO DESIGN SCENARIOS

5.1 Coordinated design of user-organisation and building

This section describes how the BAS•CAAD system can support actual design through two scenarios. The first scenario describes the task of designing a building for Mr and Mrs Smith's small business. The object of design is on one hand the building, and on the other hand the activity to be carried out inside the building. To document this knowledge, we create two Thing Classes, *Business-building* and *Business-activity*. We also add attributes that define them to be each other's environments.

THINGCLASS *Business-building*
ENVIRONMENTATTRIBUTE *Business-activity*
EXTERNALSTRUCTUREATTRIBUTE *provides-environment-to Business-activity*

THINGCLASS *Business-activity*
  ENVIRONMENTATTRIBUTE *Business-building*

Our task is now to develop the design, and to document the knowledge we may collect during the process. In the BAS•CAAD system this is accomplished by creating additional Thing Classes and by adding attributes to them.

When consulting the Smiths, we learn that the main activity of their business is decomposed into two parts, workshop and office work. We document this with two Thing Classes, *Workshop-activity* and *Office-activity*. The new Thing Classes are attributed as parts of *Business-activity* through composition attributes. The Smiths also tell us about the shape and size of the two activities. This information is added to *Workshop-activity* and *Office-activity,* respectively, through Unary Attributes.

THINGCLASS *Workshop-activity*
  UNARYATTRIBUTE *shape* VALUESPACE *(rectangle 8 x 15)*

THINGCLASS *Office-activity*
  ENVIRONMENTATTRIBUTE *Workshop-activity*
  EXTERNALSTRUCTUREATTRIBUTE *administrates Workshop-activity*
  EXTERNALSTRUCTUREATTRIBUTE *is-adjacent-to Workshop-activity*
  UNARYATTRIBUTE *shape* VALUESPACE *(rectangle 5 x 8)*

THINGCLASS *Business-activity*
  ENVIRONMENTATTRIBUTE *Business-building*
  COMPOSITIONATTRIBUTE *Office-activity*
  COMPOSITIONATTRIBUTE *Workshop-activity*
  INTERNALSTRUCTUREATTRIBUTE *Office-activity administrates Workshop-activity*

Since we were told that the two activities don't co-exist very well, we conclude that each activity needs a space of its own. Therefore the *Business-building* is divided into two spaces, *Workshop-space* and *Office-space*, which become parts of the building. The primary function of the spaces is to provide environments to the activities. This is reflected by the attributes in the definitions, which relate the activities to the spaces.

THINGCLASS *Workshop-space*
  ENVIRONMENTATTRIBUTE *Workshop-activity*
  EXTERNALSTRUCTUREATTRIBUTE *provides-environment-to Workshop-activity*

THINGCLASS *Office-space*
  ENVIRONMENTATTRIBUTE *Office-activity*
  EXTERNALSTRUCTUREATTRIBUTE *provides-environment-to Office-activity*

THINGCLASS *Business-building*
  ENVIRONMENTATTRIBUTE *Business-activity*

EXTERNALSTRUCTUREATTRIBUTE *provides-environment-to Business-activity*
COMPOSITIONATTRIBUTE *Office-space*
COMPOSITIONATTRIBUTE *Workshop-space*

In the example above, we have seen how a design can be represented by thing classes from the most generic notion, and through consecutive steps of refinement to a more developed model. Using the BAS•CAAD system, the design process would progress in a similar manner until all things in the building were described to a sufficient detail and uniquely identified.

5.2 Detailed building design

The second scenario describes how the BAS•CAAD system supports detailed building design including aspects like spatial, functional, and compositional views. Our task is to design a building, and we will use pre-defined library Thing Classes, representing common things and systems in building.

To begin with, we create the Thing Class *building* to represent our building, see Figure 9 a. Since we are interested in the spatial aspect of buildings, i.e. the ability to define a space, we decide our new Thing Class to be a subclass of the pre-defined Thing Class *space*. Here, the Thing Class *space* represents both the spatial and enclosing aspects of things. Being composed of *enclosures* is an attribute of the *space* Thing Class. The *enclosure* attribute is inherited by the *space* subclasses, e.g. our new *building* Thing Class. An *enclosure*, in this example, represents both the shape and enclosing aspects of things, e.g. prohibiting pass-through.

In the next step of the scenario we want to subdivide our building into subspaces. This is done by creating new subclasses of *space*, and by adding composition attributes to the class definition of *building*, that point to these new subspace Thing Classes, see Figure 9 b. In this schema only one of the building's subspaces is shown.

When the spaces are defined, we need to specialise the composition attributes of the spaces and the building, so that they are defined to be composed of things that specificly enclose them. The specialised *enclosures* are subclasses of the generic *enclosure* thing class, see Figure 9 c.

The enclosures of the building represent the enclosing aspect of the building's exterior walls, e.g. shape and climate enclosing. Since the *space 1* faces the exterior of the building, some of its enclosures, too, are aspects of exterior walls. A Thing Class that represents an exterior wall would need to include both aspects. This is done through multiple inheritance, where the wall *Thing Class* inherits all attributes from both enclosures, see Figure 9 d.

The design process will continue by consecutively adding attributes to the model objects until a satisfactory design has been achieved.

In order to have the building specified for cost calculation, tendering and construction, we need to define *building elements* and *work results*. Building elements are pre-defined classes of functional parts of a building, and work results are pre-defined classes of compositional parts of buildings, i.e. things that results from constuction work. In the Swedish classification system BSAB 96, we can find such classes. If these classes are expressed as thing classes in the BAS•CAAD system, we can transfer the attributes that define the BSAB classes directly through mutiple inheritance, thus including these aspectual views into the building model, see Figure 9 d.

The tests in these relatively simple scenarios indicate that the BAS•CAAD system provides a suitable generic structure for product documentation, also during the design stage.

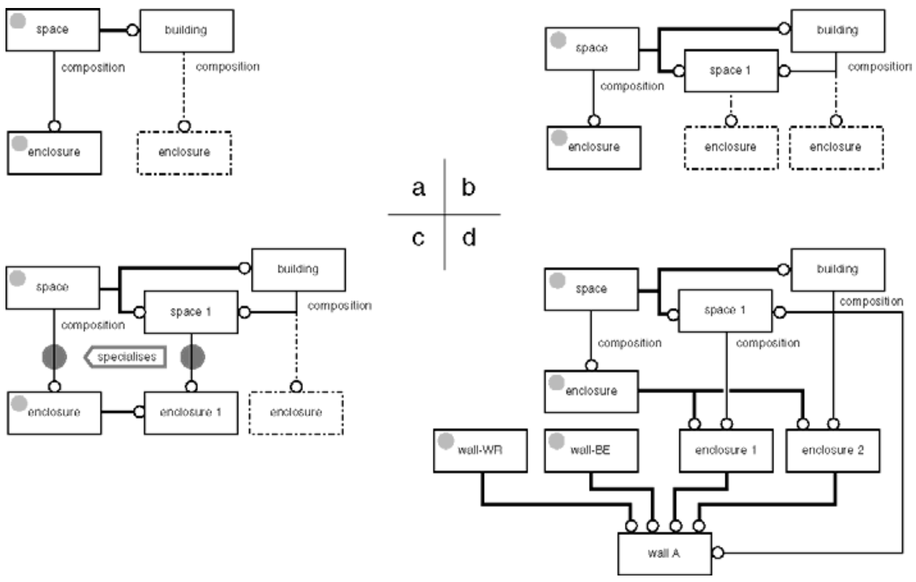

Figure 9: Four stages of the design of a building (in modified EXPRESS-G). Grey dots indicate library classes. Thick connecting lines indicate subclass relationships, thin connecting lines indicate attributes, dotted enitities indicate inherited attributes.

6 REFERENCES

Björk B C, 1995, Requirements and information structures for building product data models VTT Publications 245 (Technical research centre of Finland, Espoo)

Bunge M, 1998, *Philosophy of Science Vol 1 From Problem to Theory* (Transaction Publishers, New Brunswick, New Jersey)

Eastman C M, Fereshetian N, 1994, “Information models for use in product design: a comparison”, *Computer-Aided Design* 7 (26) 551-572

Eastman C M, Assal H, and Jeng T, 1995, “Structure of a database supporting model evolution” *Modelling of buildings through their life-cycle.* CIB Proceedings Publication 180, (Eds. Fisher M, Law K, and Luiten B) (Stanford University, Stanford, Ca, USA)

Eastman C M, Siabiris A, 1995, “A generic building product model incorporating building type information” *Automation in Construction*, 4 (3) 283-304

Ekholm A, Fridqvist S, 1996, “Modelling of user organisations, buildings and spaces for the design process” *Construction on the Information Highway* CIB Proceedings Publication 198, (Ed. Ziga Turk) (University of Ljubljana, Slovenia)

Ekholm A, Fridqvist S, 1998, “A Dynamic Information System for Design Applied to the Construction Context” *The Life-cycle of Construction IT Innovations* (Eds. Björk, B-C. and Jägbeck, A.) Proceedings from the CIB W78 workshop, 3-5 June 1998, Stockholm, Sweden

Fridqvist S, and Ekholm A, 1996 “Basic ObjectStructure for Computer Aided Modelling in Building Design” *Construction on the Information Highway* CIB Proceedings Publication 198, (Ed. Ziga Turk) (University of Ljubljana, Slovenia)

Galle P, 1995, “Towards integrated, ”intelligent”, and compliant computer modeling of buildings” *Automation in Construction* 3(4) 189-211

ISO, 1985, *Concepts and terminology for the conceptual schema and the information base* ISO/DTR 9007 (TC97), also SIS teknisk rapport 311 (SIS Stockholm)

Junge R, Steinmann R, Beetz K, 1997, “A dynamic product model” *CAAD futures 1997* (Ed.Junge R) (Dordrecht: Kluwer Academic Publishers)

Lambrix P, 1996, *Part-Whole Reasoning in Description Logics* (Diss. Linköping University, Linköping, Sweden)

Leeuwen J P, Wagter H, 1998, “A Features Framework for Architectural Information”, *roceedings of the Artificial Intelligence in Design Conference 1998* (Ed. Gero, J. And Sudweeks, F.) (Dordrecht: Kluwer Academic Publishers)

Maher M L, Simoff S J, Mitchell J, 1997, “Formalising building requirements using an Activity/Space Model” *Automation in Construction* 1(6) 77-95.

POSC/Caesar, 1999, URL: http://www.posccaesar.com/snapshote/snaptoc.htm, POSC/Caesar Reference Data Library Accessed January 1999.

Rittel H, 1984, In Cross N, *Developments in Design Methodology* (London: John Wiley and Sons)