Building Simulation 2003

Eighth International IBPSA Conference
Eindhoven, Netherlands
August 11-14, 2003

MODELING THE BUILDING AS A SYSTEM

Gerhard Zimmermann
University of Kaiserslautern
67653 Kaiserslautern

ABSTRACT

We define *building systems* as systems that not only consist of the load bearing building structure and its environment, the service systems (installations), and the control systems, but also include building usage and users. Our ultimate goal is a *virtual building laboratory* for the concurrent simulation of all components of such building systems. This will be an extension of performance simulators. In this paper we focus on an efficient modeling process for deriving object-oriented structural models for virtual building laboratories, with the specialization towards the development, test, and maintenance of integrated control systems. We also focus on control systems that support individual user environments and that can easily be adapted to changes in the use of building systems. For the purpose of efficient modeling, we divide the building system domain into five domains, called building, service, control, functional unit, and user role domain. We present corresponding domain models that predefine domain dependent modeling primitives for project specific building system models. All models are based on the notion of space and matter. In each domain, spaces have special semantics and build aggregation hierarchies, which we call "*backbones*". The backbones of the domain models are linked by requirement and spatial relations, thus composing one integrated building system model.

INTRODUCTION

Nowadays, building simulation is mainly used during the planning process for improving energy efficiency. For this purpose, specific models have been developed for computing heat flow, gains, and losses. This kind of simulation is called performance simulation. More advanced models allow the simulation of long wavelength radiation, light, sound, air flow and service systems for heating and air conditioning. Simulators based on these advanced models can also be used for planning and improving the internal environment for work places or living spaces and for testing building control systems.

Performance simulators do not include the inhabitants of buildings, more precisely the organization of the users and the user activities. New modeling approaches have included these entities, but only for the purpose of requirements descriptions and not for simulation. Also, only very simple building control algorithms have been included in performance simulators.

Control systems in contemporary buildings are more complex than what is available in performance simulators. There is a demand for even more complex integrated building automation systems. We have shown how such control systems can be efficiently developed (Queins et al., 2002). One of the difficulties that has to be solved by control systems is the increasing and changing demand of the users to take part in the control of their individual personal environment. Another difficulty is the fact that the use of commercial buildings frequently changes and control systems have to be flexible enough or easy to adjust to allow for such changes.

During the development and maintenance of building control systems that support individually controllable environments and change of building use, we need an environment for validation ("Do we build the right system?") and verification ("Do we build the system right?"). This requires more than performance simulation. It requires a view of a *buildings as a system*, including the *building* itself (the load bearing structure), the *service systems* (installations), the *functional units* (work places, activity spaces), and the inhabitants or *users* themselves (organization). To complete the building as a system, in short *building system*, we include the *control system*. Because of the differences of these five components of a building system, we define five corresponding domains.

The result of our current work, which we present in this paper, is a modeling approach that supports the process of deriving models in each of the five domains and of connecting the five models by relations. Models are derived on a project by project basis and thus the modeling process has to be very efficient.

Since the goal is an environment for validating and verifying control systems by prototyping, all models have to be transformed into computer programs. Therefore, we use model notations that can automatically be transformed into executable programs. With additional features that support experimentation, the envisioned simulation environment will become a *virtual building laboratory* (Mahdavi et al., 2002). Besides the testing of control systems, such a virtual laboratory can also be used for virtual construction tasks, as defined in (Clark, 2001). This is a great challenge because during the design of buildings, changes in all domains can occur and consistency has to be maintained. At the moment we only envision a fixed building design with changing usage. We can also see a useful application of our approach in the support of the architectural programming process.

Our object-oriented modeling approach is mainly based on separation of concerns, hierarchical structures, and the notion of space and matter. Separation of concerns is achieved by dividing building system models into five domain specific models. Domain experts can derive these models in parallel and the propagation of changes can be minimized. In each domain a hierarchical *backbone* of spaces is defined. The semantics of the spaces is domain dependent, but all are geometric entities that are related to matter entities. The spaces in the different domains are related by requirements and spatial relations.

The process of deriving models is based on several levels of model abstraction, ranging from very general models to project specific models. The more abstract models define domain specific model primitives and are presented in this paper. In a project, these primitives are used to create specific models with a high degree of reuse capabilities. Examples are shown in Figure 5 and Figure 10.

Our approach has been tested in several case studies in the control domain (Queins et al., 2002) and in one case study in the building simulation domain (Zimmermann, 2002). Case studies in other domains are in progress.

RELATED WORK

Many individual aspects of our approach have already been published and used in the computer science and in the building simulation communities. Here, we will only refer to a selection.

Object-oriented modeling has a long history in computer science, especially for data bases and in software engineering, and has been widely adopted with the introduction of the UML (Booch et al., 1999). In our work, we use UML class diagrams to specify the system structure and SDL state transition diagrams for modeling behavior (Olsen et al., 1997). Telelogic's Tau SDL Suite (Telelogic) is used to generate executable C-code from SDL models, including a real-time runtime environment. It has been successfully used in the mentioned case studies.

Building simulators can be partitioned into three parts and classified regarding object-orientation: The input of building data can be object-oriented (o) or not (n) and will be called the *interface model*. For example, drawing tools can either enter and manipulate building objects (o) or geometric primitives (n). The internal data representation is called the *data model*, also either (o) or (n). The simulation itself, called the *execution model*, is typically based on the numerical solution of large systems of equations (n), but can also be based on assigning physical properties to objects and calculations hidden in these objects (o). With this classification, we can describe the degree of object-orientation of simulation environments. (ooo) would describe a fully object-oriented environment.

The prototype of an (nnn) environment is DOE-2 (DOE-2). But there are several examples of building simulation environments with object-oriented interface models; e.g., ESP-r (ESP-r). COMBINE (Combine), IDEA+ (Hendricx, 1997), and SEMPER (Mahdavi, 1996) use object oriented data models. The importance and a clear definition of space in data models is given in (Eastman et al, 1995; Ekholm et al., 2000). The latter also gives a good overview over other data models. The model that we present in this paper is fully object-oriented and is closely related to the aforementioned models.

Object-oriented execution models have been the goal of the Energy Kernel System (Clarke et al., 1993). We understand this as a mixture between distributed problem solving and central numerical solvers. A fully distributed (object-oriented) execution model has been shown by Riegel (Riegel et al., 1997), using design patterns to create simulators from scratch very efficiently. Another fully object-oriented execution model has been demonstrated by the author (Zimmermann, 2001), using automatic code generation. Both approaches can be called (ooo).

Clarke (Clarke, 2001) has pointed out the importance of integrating different effects in simulation environments. Current practice is still based on reentering or reformatting the input data for different simulation environments. The Building Design Advisor (Papamichael et al., 1997), the ESP-r, and the SEMPER environment provide all or a subset of thermal, light, air flow, sound, and HVAC-system simulation. We go beyond these approaches by also integrating the simulation of functional units and of users.

Functional units, also called activity spaces, have been formally modeled by (Eastman et al., 1995) and (Ekholm et al., 2000) to represent requirements of the organization that will occupy a building. Activities and organization, representing persons, are combined in one model. Functional units are also modeled in SEED (Flemming et al., 1995) for the generation of space programs. In all cases, no simulation was intended. In our approach we construct models that can be transformed into executable software for the purpose of dynamic simulation. We also assign functional units and users to separate domains to be more flexible in reaction to changes of building use.

Executable models of inhabitants of buildings are equally rare. Human figure models that can move autonomously (Armstrong et al., 1987) are good for virtual reality purposes, but not in our context of testing control systems. More to the point are abstractions of users, as for example route analyses (Koutamanis et al., 2001). Post-occupancy analysis is a useful method to define user behavior.

Several projects are concerned with the design process itself and its modeling. The COMBINE project uses Combi Nets to model the control of the process. The M.E.R.O.DE development method (Hendricx, 1997) is a way for producing a central computer model for all data involved in the architectural design process.

MODEL ABSTRACTIONS

Computer science has a long history in abstract modeling of complex systems. Physics is strong in modeling real world entities and relations. On the basis of both sciences we express the relation of the physical entities *space, time, matter,* and *energy* with computer science models.

Building system models deal with different types of information and we have to communicate with experts from different domains. Therefore, we use *textual, geometric, topological, object type, process graph, state transition,* and *mathematical models,* whatever is most suitable. The first three are used to describe interface models, the last four describe the data and execution models. In this paper, we concentrate on object type models.

Object types are abstractions of sets of instances (objects). As a suitable notation, we chose UML class diagrams (Booch et al., 1999). Consequently, in the following object type diagrams, boxes denote types, lines with a Δ denote generalization relations, lines with a ◊ denote aggregation relations, and normal lines denote other relations. Names can be assigned to relations and directed by filled arrowheads.

Throughout this paper we will abbreviate key words in graphical representations and tables and use the full words in running text. Both representations are linked by using bold and normal fonts, as for example **BldSpaceT** and **B**ui**ld**ing **Space T**ype.

Figure 1 shows the model hierarchy. The three top levels, the *system level*, the *domain level,* and the *application domain level,* define types of object types (grey boxes). All entities in models that are types of types have names with the postfix **Type** or **T**. In principle, these models define a notation for object type models at the *project level*. For all entities at the three top levels semantics are defined. These semantics can have the form of physical equations or of behavioral models, for example. Thus, they form a repository of reusable development products.

At the system level, the *Building System Model* (see Figure 2) defines entities and basic relations that are specialized in the next two levels. At the domain level, five different *domain models* (e.g., Figure 4) are defined. At the application domain level, further specializations for application domains are defined, the

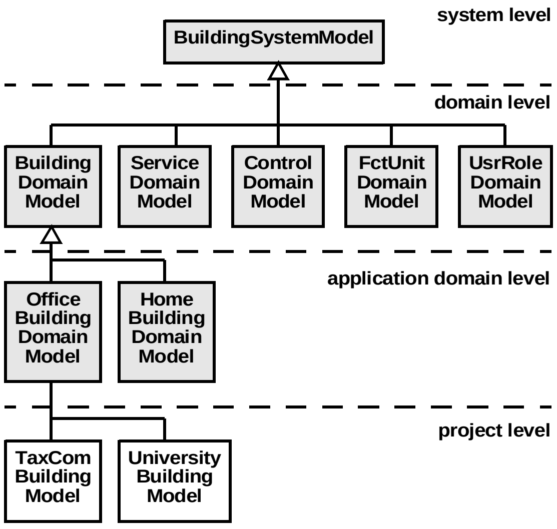

Figure 1 Model hierarchy. The grey boxes denote domain specific models, the white boxes project models. At the application domain and the project level only examples for the building domain are shown.

application domain models (e.g., Table 1). The project level shows *project models*. One example is a **Tax Com**pany project. Figure 5 shows an example. The project models are composed of object types specific for the project building system. They are instantiations of types of object types of the domain and the application domain models.

To simplify the graphical representation of the models, the generalization relations of entities at the domain and the application domain levels to entities at the system level are denoted by «**Type**». In Tables we call this relation *System Type*.

THE BUILDING SYSTEM MODEL

The model in Figure 2 shows the **requires** and **fulfilledBy** relations between the two major **Artifact** types **SystemObjectType** and **Requirement**. It has to be pointed out that **Requirement** is not a type of types and thus is inherited by all models below the system model. **Requirement** mainly occurs between entities of different project model domains, thus linking the domains to each other. The meaning is that one object type has a requirement that is fulfilled by another object type. In further stages of the design of the virtual laboratory these relations will lead to spatial or communication relationships.

As the model shows, all object types of **SystemObjectType** are either a specialization of **SpaceType** or **MatterType**. Energy and time are features of the behavior that is not considered at this point. In regard to the five domain models, the **SpaceType** object types have different semantics and are **realizedBy** different **MatterType** object types. These semantics are explained in the corresponding five domain chapters. Common to all domains is that spaces define abstract

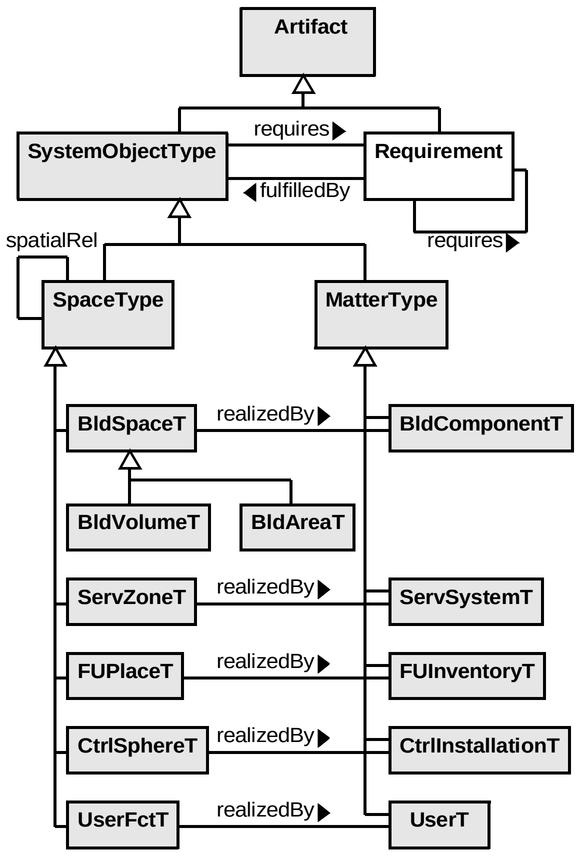

Figure 2 Building system model

volumes. In the case of the building domain this includes areas (**Build**ing**Area**Type). Following the definition in (Ekholm et al., 2000), we distinguish *factual space,* which is precisely defined by enclosing matter, and *experiential space,* which is defined by observation and interpretation.

As we will see in the domain models, spaces build aggregation hierarchies, the “backbones” of the models. The **spatialRel**ation of **SpaceType** connects the backbones of different domain models.

Before we go into the details of the domain models, we look at the requirements engineering process that creates project models.

THE PROCESS

In this paper we regard requirements engineering as the process of creating a problem description with a customer and of translating the problem description into a requirements specification for a building system. The first phase of the translation process is the definition of the structure and the assignment of requirements to object types of this structure. This phase is closely related to the architectural programming process and our modeling approach can be used to support and formalize this process.The functional program would relate to the **FunctionalUnitModel**, the space program to the **BuildingModel**. The second phase is the modeling of the behavior of object types to realize the requirements. The third phase is the generation of an executable prototype of the system. In this paper, we are concerned with the first phase.

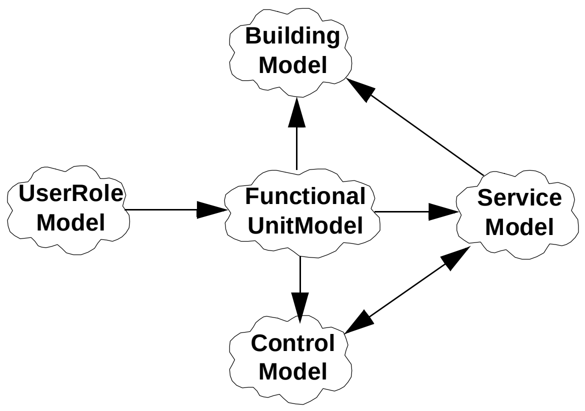

*Figure 3 Process overview. The arrows indicate the **requires** relation.*

Figure 3 shows the principal dependencies in the process. The arrows are a symbol for **Requirement** entities as well as the **requires** and the **fulfilledBy** relation, pointing in the direction of both relations. Starting point in our philosophy is the **UserRoleModel**. It describes the organization and its members with roles and functions of the users of a building or building complex. These functions require functional units in the **FunctionalUnitModel**. These may already exist or have to be provided. This requirements relation can change with changes of usage of the building system.

Functional units require physical space that has to be provided by spaces in the **BuildingModel**. If such spaces exist, this is an assignment process leading to spatial relations between functional unit places and building spaces. Otherwise, the building spaces have to be created. The relation between functional unit places and building spaces can change with changing usage. Additional requirements are environmental properties, like for example air quality, light level, accessibility and security. Our idea is not to assign these requirements to persons or user functions, but to the corresponding functional units. Most of these properties require control. In such cases the requirement is fulfilled primarily by control spheres that secondarily may require services from service zones. Therefore, we have a requirements chain from the **UserRoleModel** via the **FunctionalUnitModel** to the **ControlModel** and the **ServiceModel**. Some properties can be directly fulfilled by service zones and service systems may require control spheres without being related to other models. Service zones and service systems also require building spaces. These relations are reflected in the arrows in Figure 3.

It should be noted, that Figure 3 describes possible flows of reasoning, but no order in time. In real projects large parts in each domain may already exist

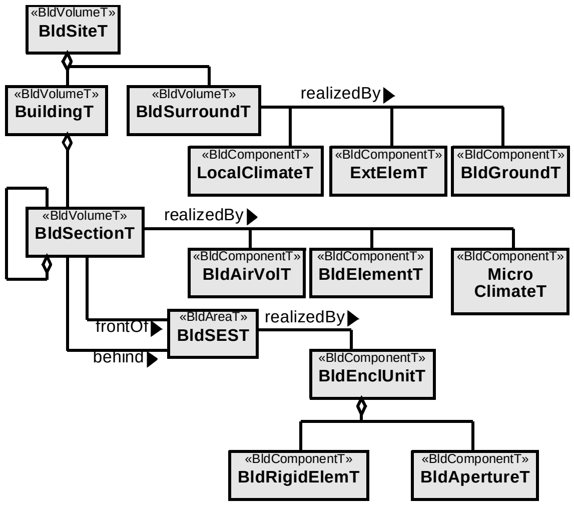
«BldVolumeT» BldSiteT
«BldVolumeT» BuildingT
«BldVolumeT» BldSurroundT
realizedBy
«BldComponentT» LocalClimateT
«BldComponentT» ExtElemT
«BldComponentT» BldGroundT
«BldVolumeT» BldSectionT
realizedBy
«BldComponentT» BldAirVolT
«BldComponentT» BldElementT
«BldComponentT» Micro ClimateT
frontOf
behind
«BldAreaT» BldSEST
realizedBy
«BldComponentT» BldEnclUnitT
«BldComponentT» BldRigidElemT
«BldComponentT» BldApertureT

Figure 4 Building Domain Model

or are already specified. In such cases, modeling can start in all domains simultaneously and only additional object types and relations are added in the above sequence of reasoning.

We will now describe the five domains in more detail by starting with the building domain, which is generally known best.

THE BUILDING DOMAIN

All entities in the building domain model in Figure 4 are derived from the building system model in Figure 2. In principal they are specializations of building system model entities. For example, **B**ui**ld**ing**SiteT**ype is a specialization of **B**ui**ld**ing**VolumeT**ype.

Most entities are self-explaining and follow the usual nomenclature of buildings. The BuildingSectionEnclosureSegmentType (**BldSEST**) needs some explanation. Objects of this type separate exactly two **B**ui**ld**ing**Section**s (**frontOf** and **behind**) and are **realizedBy B**ui**ld**ing**Encl**osure**Unit**s. These units can be composed of rigid elements as for example walls and windows and of apertures. Figure 5 shows an example of using these relations..

Table 1 Office Building Domain Model (excerpt)

Object Type	Parent Type	System Type
RoomT	BldSectionT	BldVolumeT
CellT	BldSectionT	BldVolumeT
OutdSectT	BldSectionT	BldVolumeT
StoreyT	BldSectionT	BldVolumeT
WallElemT	BldRigidElemT	BldComponentT

An application domain model for office buildings is the *Office Building Domain Model.* We do not show a graphical representation here, but an excerpt of a table representation (Table 1). It shows specializations of **B**ui**ld**ing**SectionT**ypes and of **B**ui**ld**ing**RigidEle**-

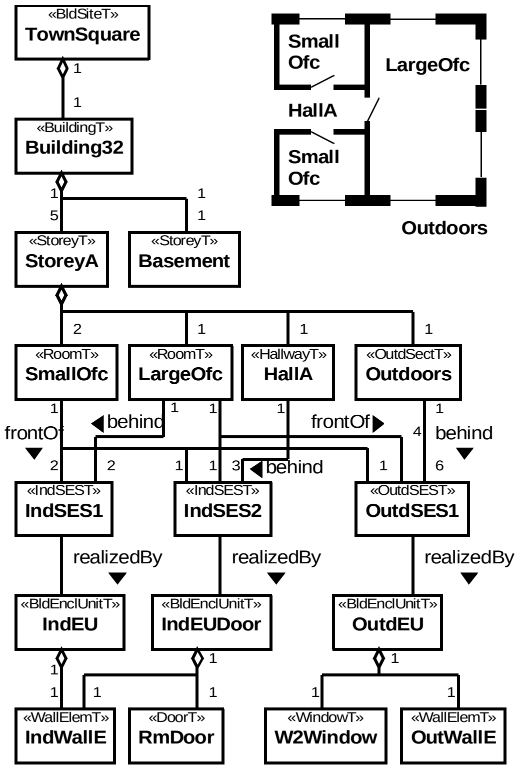

Figure 5 Project Building Model with corresponding floorplan of type ***StoreyA****.*

ment**T**ypes. **RoomT**ypes are typically defined by walls and represent factual spaces. **CellT**ypes are used to partition rooms or sections, often based on a grid. **CellT**ypes represent experiential spaces. All other new entities should be self-explaining

All projects with office buildings use the types of object types from the Office Building Domain Model. If, during the modeling process, special types are missing, they can be added to this application domain model.

Figure 5 shows an example project model. The cardinalities show the number of instances of each type. For example, each **SmallOfc** space is defined by the front of 2 **IndSES1**, 1 **IndSES2** with a door, and 1 **OutdSES1** with a window. During the modeling process instance names will be given to these instances.

THE SERVICE DOMAIN

ServiceZoneTypes are defined by the space that is serviced by a **ServiceSystemT**ype. This space is not always enclosed by matter, but rather an experiential space. Within a zone, a physical value like illuminance in a light zone may vary geometrically and over time.

Figure 6 shows the specialization of ServiceZoneTypes into five subdomains that are typically **realizedBy** different system types. Within each of the subdomains, which are not explained further, finer specializations in application domain models exist.

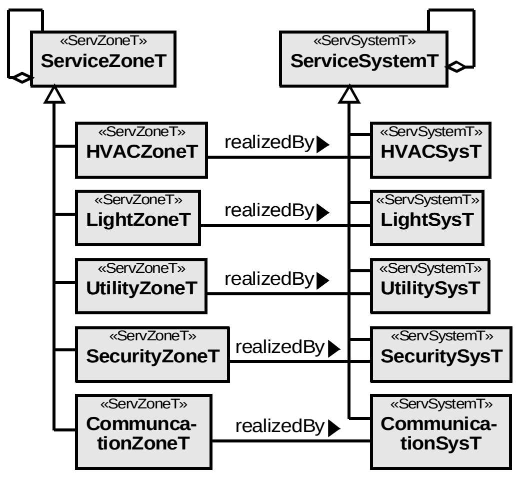

Figure 6 Service Domain Model

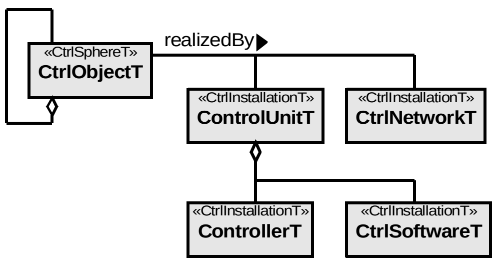

Figure 7 Control Domain Model

THE CONTROL DOMAIN

A **C**on**tr**o**lSphereT**ype is defined in our model as the space for which a **C**on**tr**o**lObjectT**ype assumes the responsibility for distinct control tasks. Other alternatives, like the definition by actuator or sensor influence spheres, are not proposed, although possible. The advantage of our definition is the direct correspondence to service zones or to functional unit places.

During the requirements engineering phase, **C**on**tr**o**lObjectT**ypes are **realizedBy** **C**on**tr**o**lSoftwareT**ypes. During the design phase of the control system, **ControllerT**ype hardware and **C**on**tr**o**lNetworkT**ypes will be added. Figure 7 shows the relations between these object types.

For offices and similar buildings, a specialization of the **C**on**tr**o**lObjectT**ype is not necessary. The relation to the other domains, especially the service and the functional unit domains, will be made through requirement relations.

The requirements engineering process for control systems has been studied and tested in several case studies. Several publications exist, e.g., (Queins et al., 1999). Therefore, we do not elaborate this domain in this paper, although test environment for control systems is the ultimate goal of the presented research.

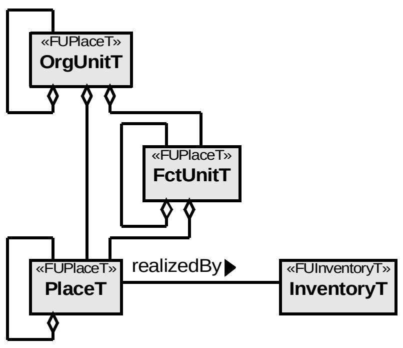

Figure 8 Functional Unit Domain Model

THE FUNCTIONAL UNIT DOMAIN

The space that defines a functional unit is called **F**unctional**U**nit**PlaceT**ype. The space can be factual or experiential. As Figure 8 shows, we have three major levels and additional sublevels in an aggregation hierarchy of places. The top level **Org**anizational**UnitT**ype is defined by structures that correspond with structures in the user role model. There may be requirements such as security that are related to organizational entities. Similarly, **Fu**n**ct**ional**UnitT**ypes are **FUPlace**s for groups of functions that may have requirements in common. Both are composed of **PlaceT**ypes that support one or more functions and that are **realizedBy** the assigned **InventoryT**ypes, for example furniture. But there may be **PlaceT**ypes, as for example for circulation, that have no inventory.

Table 2 Office Functional Unit Domain Model (excerpt)

Object Type	Parent Type	System Type
WorkPlaceT	PlaceT	FUPlaceT
SocialPlaceT	PlaceT	FUPlaceT
ServicePlaceT	PlaceT	FUPlaceT
StoragePlaceT	PlaceT	FUPlaceT
FurnitureT	InventoryT	FUInventoryT
DeskT	FurnitureT	FUInventoryT
ChairT	FurnitureT	FUInventoryT
CabinetT	FurnitureT	FUInventoryT
EquipmentT	InventoryT	FUInventoryT
Computer	EquipmentT	FUInventoryT

Table 2 shows part of the application domain model for functional units in offices. In projects an office work place will typically be realized by a desk, chair, cabinet, and computer. The work place will have requirements for ambient and work light, for temperature control, and for air quality. All of them may depend on occupancy.

THE USER ROLE DOMAIN

User functions and roles are strongly bound to spaces, especially to functional units as far as this model is

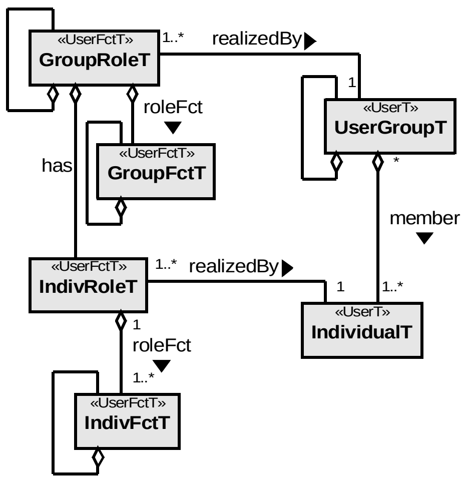

Figure 9 User Role Domain Model

concerned. Therefore, we have defined **UserFctT** as a specialization of **SpaceType**, although it is not really a space. Like in the functional unit model, we also use an aggregation hierarchy to introduce roles that are composed of several functions and to distinguish between group and individual roles and functions. Figure 9 shows the relations.

Roles of groups and of individuals are **realizedBy UserGroupT**ypes or **IndividualT**ypes respectively. The member relation is redundant because we can derive it from the other relations. During the modeling process it may be temporarily necessary, if we model the user hierarchy before we model the roles and functions.

As mentioned in section "THE PROCESS", user roles and functions are the source of most requirements. In our approach we try to avoid defining requirements of users directly. This gives us the freedom to exchange assignments of users to roles without influencing the relations to functional units. But this does not exclude the definition of requirements of users, for example a specific control requirement for a handicapped person.

For the application domain model we show an excerpt in Table 3.

Table 3 Office User Role Domain Model (excerpt)

Object Type	Parent Type	System Type
StaffRoleT	IndivRoleT	UserFctT
VisitorRoleT	IndivRoleT	UserFctT
AccessFctT	IndivFctT	UserFctT
WorkT	IndivFctT	UserFctT
SocialFctT	IndivFctT	UserFctT
BodilyFctT	IndivFctT	UserFctT
HandlingFctT	IndivFctT	UserFctT
CompanyUserT	UserGroupT	UserT
HumanUserT	IndivUserT	UserT

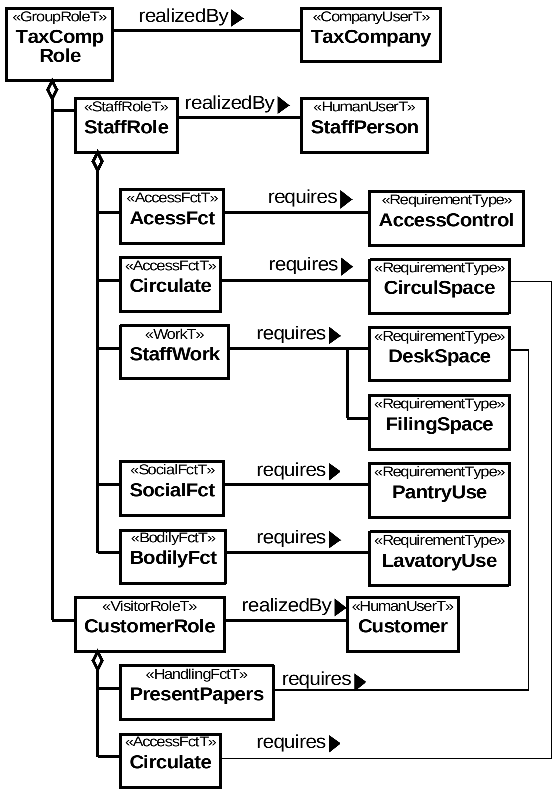

Figure 10 Tax Company User Role Model (simplified) with requirements

The selection has been made in order to explain a simplified Tax Company User Role Model.

This model in Figure 10 shows different functions that are part of a **StaffRole**, for example **StaffWork** or **SocialFunct**ion. The **StaffRole** also includes access to the company work place and moving (circulate) within the space in order to accomplish the different functions. This fine partitioning has been made in order to find requirements that can be fulfilled by similar partitions in the functional unit model. The **CustomerRole** is composed of other, but also the same functions as the **StaffRole**. The **Circulate** function has been copied in the diagram to simplify the layout.

The requirements are self-explaining. In this case all requirements can be fulfilled by **Place**s in the corresponding functional unit model. This relation is not shown in Figure 10.

CONCLUSION

In this paper we have shown a set of five domain models and corresponding specializations for the office application domain. All models are based on the physical base-types space and matter. The models cover the building structure in its environment, the service systems that supply HVAC, light, and other services, the integrated control systems, the functional units, and the user roles. All models are, on the one hand, united by kind of parent model that defines the top of a generalization hierarchy, on the other hand, the five models are connected by requirements relations. To our knowledge it has never been attempted to integrate these five domains in one model before.

The five domain models, especially the application domain models, define a modeling language in terms of the application that provides a means of communication between modeling and domain experts. They are used in project models as predefined elements and thus provide reusable artifacts.

The next goal in our research is the introduction of behavior. For this purpose we will transform the object type structures of the project models into SDL block type structures, using patterns that have been developed for the development of control systems (Metzger et al., 2002). This transformation can be automated as it has been shown by Metzger (Metzger et al., 2003). For each object type a process is generated that defines its behavior by means of an extended finite state machine. Processes communicate by exchanging signals. Dynamic behavior can be introduced by timers. These models can be automatically transformed into C-code and then be executed in a run-time environment.

The final goal is to extend this approach to create virtual building laboratories on a project by project basis and test it in a number of realistic case studies.

ACKNOWLEDGEMENTS

This modeling work could not have been done without the support of specialists in the field of architecture and of modeling in computer science. We learned much from Ardeshir Mahdavi about building models for simulation and the five submodels went through many versions and improvements in detailed discussions with Stephanie Bartos. Modeling principles, especially UML conformity, have been introduced with the help of Stefan Queins and Andreas Metzger. Both are also the main contributors to the underlying requirements engineering process for reactive systems with a strong prototyping component. We also have to thank the Deutsche Forschungsgemeinschaft for funding support as part of the Sonderforschungsbereich 501 “Entwicklung großer Systeme mit generischen Methoden”.

REFERENCES

Armstrong, W. W. et al. 1987. ”Near-Real-Time Control of Human Figure Models”, IEEE Computer Graphics and Applications, vol. 7, pp. 52-60.

Booch, G., Rumbaugh, J., and Jacobson, I. 1999. ”The Unified Modeling Language User Guide”, Addison-Wesley.

Clarke, J.A. and Mac Randal, D.F. 1993. ”The Energy Kernel System: Form and Content”, Proc. Building Simulation ’93, pp. 307-315

Clark, J.A. 2001. ”Energy Simulation in Building Design”, Butterworth-Heinemann, 2nd Edition, Oxford.

COMBINE, http://erg.ucd.ie/combine.html

DOE-2, http://gundog.lbl.gov/dirsoft/d2whatis.html

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

Ekholm, A. and Fridquist, S. 2000. ”A concept of space for building classification, product-modeling, and design”, Automation in Construction 9, pp. 315-328.

ESP-r, http://www.esru.strath.ac.uk

Flemming, U. and Woodbury, R. 1995. "Software Environment to Support Early Phases in Building Design (SEED): Overview" Journal of Architectural Engineering 1, pp. 147-152

Hendricx, A. 1997. ”Shape, Space and Building Element: Development of a Conceptual Object Model for the Design Process”, 15th eCAADe Conf. Proceedings, Vienna.

Koutamanis, A, et al. 2001. ”Route Analysis in Complex Buildings”, Proceedings of the 9th Int. Conf. on Computer Aided Architectural Design Futures 2001, pp. 711-724.

Mahdavi, A. 1996. “SEMPER: A New Computational Environment for Simulation-based Building Design Assistance”, Proceedings of the 1996 International Symposium of CIB W67 (Energy and Mass Flows in the Life Cycle of Buildings). Vienna, Austria.

Mahdavi, A., Metzger, A., and Zimmermann, G. 2002. ” Towards a Virtual Laboratory for Building Performance and Control”, in R. Trappl (Ed.) Cybernetics and Systems 2002, Vol. 1, pp. 281-286

Metzger, A., and Queins, S. 2002. ”Specifying Building Automation Systems with PROBAnD, a Method Based on Prototyping, Reuse, and Object-Orientation”, OMER - Object-Oriented Modeling of Embedded Real-Time Systems, Lecture Notes in Informatics, P-5, Köllen Verlag Bonn, pp. 135-140.

Metzger, A.,and Queins, S. 2003. "Model-Based Generation of SDL Specifications for the Early Prototyping of Reactive Systems" in M.E. Sherrat (Ed.) Telecommunications and beyond: The Broader Applicability of SDL and MSC. Springer Lecture Notes in Computer Science, LNCS 2599. Heidelberg: Springer-Verlag. pp. 158-169.

Olsen, A. et al. 1997. ”System Engineering Using SDL-92”, 4th Edition, North Holland, Amsterdam.

Papamichael, K., LaPorta, J., and Chauvet, H. 1997. ”Building Design Advisor: Automated Integration of Multiple Simulation Tools”, Automation in Construction 6(4), pp. 341-352.

Queins, S. and Zimmermann, G. 1999. ”A First Iteration of a Reuse-Driven Domain-Specific System Requirements Analysis Process”, SFB501 Report No. 13/99, University of Kaiserslautern.

Riegel, J. P., Schütze, M., and Zimmermann, G. 1997. ”Pattern-Based Generation of Customized, Flexible Building Simulators”, CAAD Futures 97, Kluwer Academic Publishers, Dordtrecht, pp. 285-296.

Telelogic Tau SDL Suite, http://www.telelogic.com

Zimmermann, G. 2002. ”Efficient Creation of Building Performance Simulators Using Automatic Code Generation” Energy and Buildings, 34, Elsevier Science B.V., pp. 973-983.

Zimmermann, G. 2001. ”A New Approach to Building Simulation based on Communicating Objects”, 7th International IBPSA Conference Proceedings, Rio de Janeiro pp. 707-714.