A Hierarchical Model for Life Cycle Costs of Buildings

John R. Bedell[1] and Niklaus Kohler[2]

Introduction

The construction of a building as well as its use, maintenance, modification, and eventual demolition comprises a gargantuan collection of complex and expensive processes, which recent increased attention to the environment and to resource consumption have only rendered more formidable. The ability to forecast the costs of these industrial procedures, and so to make informed choices among alternative solutions, lies at the forefront of those potential applications made feasible by progress in the development of representative models for buildings.

The presence of those other possibilities, though, raises issues of compatibility, not just because of their common building domain but because of their often interdependent nature. Thus while directing our efforts toward a cost model, we must also follow a unified approach such as that of [Bjork 92] accommodating the specialized viewpoints of architects, electricians, plumbers, and others working on a building over the course of its construction and lifetime. Is it possible to include all of these interests within a single comprehensive model? Failing this, can we find a common conceptual foundation on which a variety of data models can eventually develop and intercommunicate?

Because of the difficulty inherent in trying to anticipate every possible feature required of one all-embracing model, the second approach is probably more realistic. At the same time we must keep in mind the properties of the larger structures. The specialization that increases on descent from categories of activity to individual applications, and thence to finer levels of detail, combined with the composite nature of many of the applications themselves, all agree with the common view of design as a hierarchical process and its results as hierarchies. The medium of representation, then, should express building products as pyramids of encapsulated modules, ideally having a flexibility of formation permitting the user to pursue top-down or bottom-up approaches to design.

Hierarchical Cost Applications

Estimation of Life Cycle Costs

We begin from the perspective of our original application context, in which we seek to estimate the economic and environmental costs of construction, maintenance, and other operations on a building during its lifetime [Kohler 91]. The description of a building is centered around a process entity that can model any procedure required during these phases, whether the production or assembly of a component or ingredient, the extraction of a resource, or the provision of a service. As part of the search for a balance of mass and energy flows between the building activity and an external domain such as nature, a cost evaluation can be made by successive examination of the inputs and outputs of each process.

---

[1]Laboratory of Solar Energy and Building Physics (LESO-PB)

[2]Swiss Federal Institute of Technology, Lausanne (EPFL).

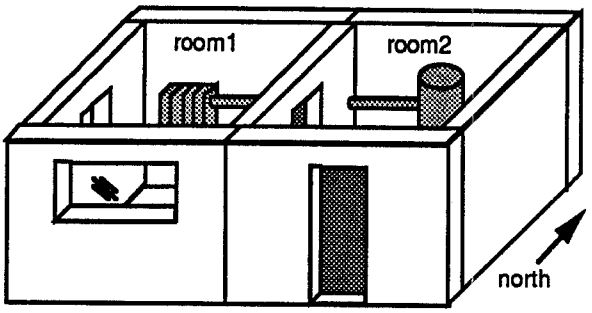

Figure 1 Two-room house example

The parts of a building are broken down into the standard categories and subcategories defined in [CRB 91], the further sub-elements of [IPBAU 91], then beyond this into the components, ingredients, and resources used in the production of those parts. To model the simplified example of the water-heated, two-room concrete house seen in Figure 1, we require only a small subset of the available categories. In the resulting decomposition of Figure 2, each item in the upper half of a node is the product of the aggregation or process in the lower half acting on a set of inputs below it, which in turn are products of earlier process formulae in lower nodes, and so on down to the elementary materials at the leaf nodes. Here we require four of the CRB element groups *A-Z*: *D*, substructure; *E*, superstructure; *I*, mechanical and electrical systems; *M*, finishing work. Group *I*, for example, then contains *I2*, the heating element, which is implemented as a hot water system and further categorized into the IP-BAU sub-elements *I2.2*, boiler; *I2.3*, pipe; *I2.4*, radiator. The boiler is manufactured from steel, which is produced from iron, which is smelted from naturally occurring ore. Subtrees are shared where appropriate, as in the use of the same concrete slabs for floor, walls, and roof of the sub- and superstructure. The near-universal needs of energy and transport are for clarity attached here only to the shaded example of cement production. They, along with labor, actually contribute to nearly all stages of construction. Other simplifications include the omission of components for transport and dynamite, as well as the use of water only in the mixing of concrete and not, say, in metal and wood production. In accomplishing its task, each node's operation entails certain costs: economic costs in paying for required inputs and environmental costs in the consumption of natural resources and the generation of waste, pollution, and (sometimes valuable) byproducts.

To obtain cost figures embodied in the hierarchy, each node can be seen in functional terms. For a given quantity of its product, using that product's chosen implementing process, it calculates the resulting subcosts and required quantities of all components. Thus the output of the node becomes the input of the function and vice-versa. The components, of course, are themselves the products of earlier processes; by recursively adding these subcosts and propagating downward from the latest (uppermost) node, the total costs of the building can be determined. Penetration of the tree down to or down from a fixed depth, or over a range of depths, yields figures for particular stages of construction. Alternatively, by traversing only subtrees of the whole, as in the shaded example of cement, one can concentrate on limited categories of components.

This tree structure allows modelling of a building not only in the usual architectural sense, that is "as designed" or "as built", but also "as maintained", "as refurbished", "as used", and eventually "as demolished". To evaluate these often-neglected later phases of a building's life cycle, nodes can possess alternative cost subtrees as necessary. Thus the operation of window replacement might combine a process for the cost of removing the old

windows with that for manufacturing the new ones. Such a time-based component might be included intermittently within the building structure or estimated as a continuous expense, as with energy consumption, and evaluated over the relevant time period.

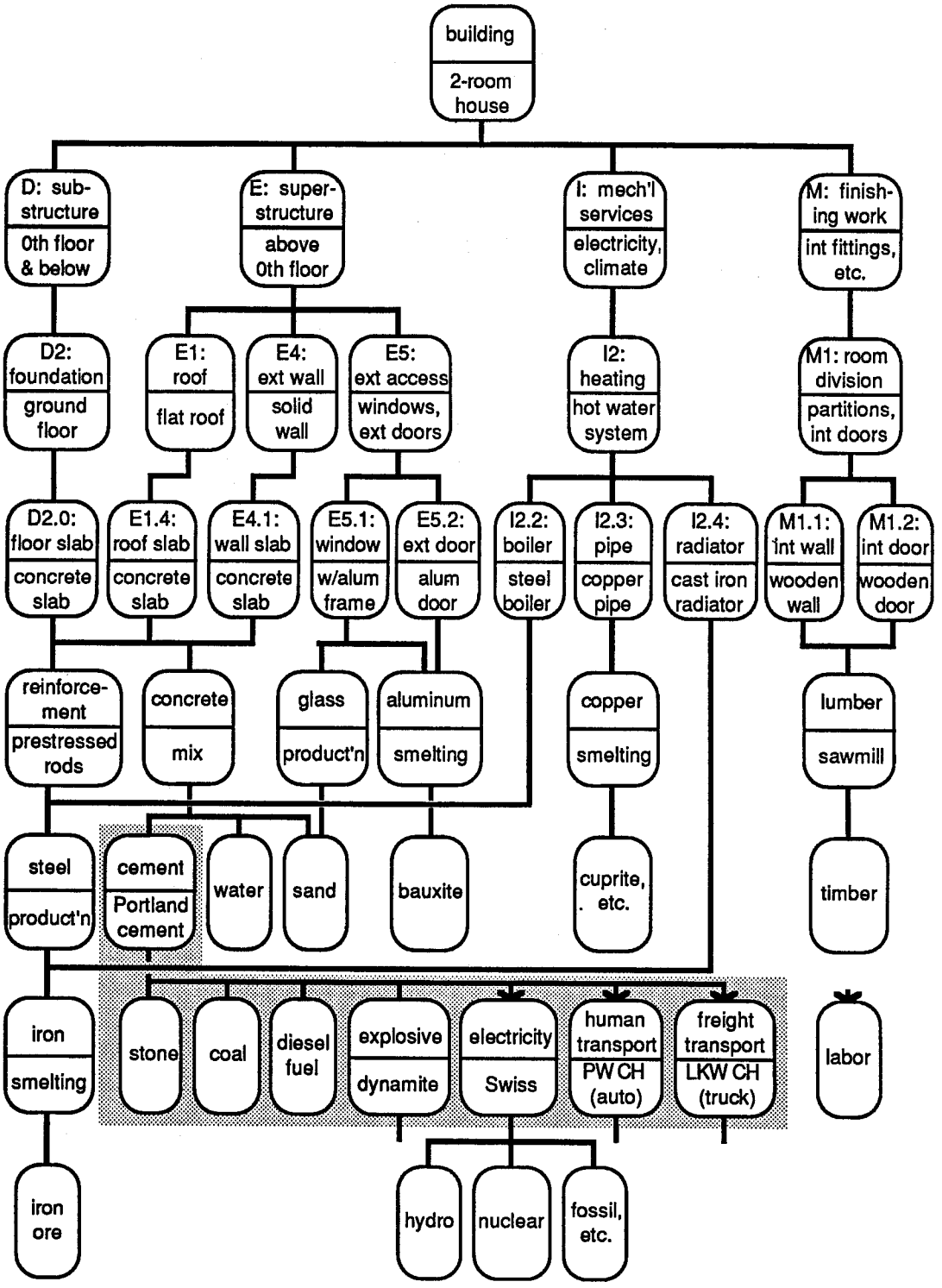

Figure 2 Modeling building costs using process entities

Access Model for Refurbishment

Since the component trees of these various phases are of arbitrary complexity, they may be elaborated as necessary to interrelate with, or contain within themselves, what amount to whole new subapplications. The refurbishment of a residence, for example, involves many tasks whose costs are significantly influenced by the order in which they are executed and by the disturbance they cause to the occupants. The tool described in [Glardon 91] seeks to optimize the former and minimize the latter by generating a schedule tailored to the residence's particular requirements. To accomplish this, its spaces, structural components, and access portals are expressed as nodes in an adjacency graph. The previously mentioned house of Figure 1 has two rooms connected by a door through the partition wall. Room 1, on the west, has southern and western lights and a radiator on the north wall; room 2 has an exterior door to the south and a boiler on the north wall connected by a pipe to the radiator.

Adjacency graphs for this house can display different levels of detail. From the outside, in Figure 3a, the structure appears as four walls (*WN*, *WE*, *WS*, *WW*) with door (*dS*) and windows (*wS*, *wW*) along the perimeter (gray square) of an unknown interior (*I*). The arcs between these nodes define adjacency between this latter space and the exterior walls, to which door and window nodes are connected to indicate their permeability to the exterior spaces (*N*, *E*, *S*, *W*). In Figure 3b we expand the graph to reflect an inner layer of detail. While the perimeter remains the same, the north and south walls can be seen as divisible into boundaries (*W1N*, *W2N*, *W1S*, *W2S*) of the now discernable individual rooms (*R1*, *R2*), between which appear the interior wall (*W12*) and door (*d12*). The two halves of the south wall have inherited respective connections with the south window and door. This and other data within the building model provide a database for generating the appropriate refurbishment strategy. A finer level of detail might represent zones within large open spaces such as semi-partitioned offices and factory floors, while at higher levels units might be grouped into, say, floors of apartments accessible by stairs or elevators, buildings with foyers and service entrances, streets of attached row houses joined with others at intersections, and whole city blocks with main access routes and back alleyways. Nonhuman access, such as that of water flowing through the pipe between the boiler and the radiator, might be represented in a similar way.

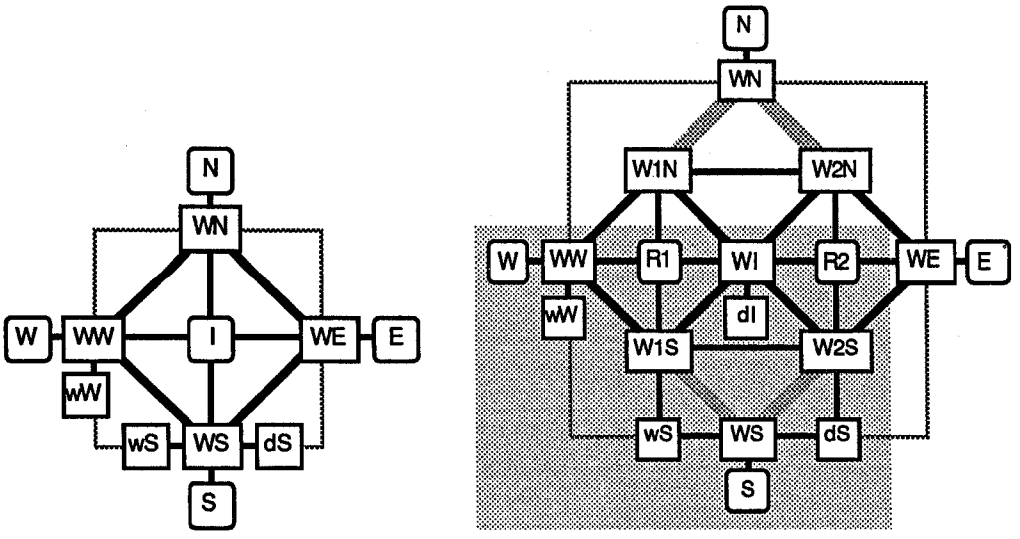

Figure 3: Adjacency graphs for a) exterior view, b) interior view of house

While this access/adjacency structure relates directly to issues of refurbishment and maintenance costs, it also contains elements (e.g. rooms) and aspects (e.g. topology) that fall outside the range covered by the original cost evaluator that connects to or even contains it. Only some of the components in the latter (e.g. walls, floors) will interest the refurbishment planner, which is concerned largely with divisions of space not defined in the other model. An element common to both applications might use a different set of attributes for each, and would be configured into two or more networks. Our fundamental units, then, must be versatile enough for such specializations while providing also for any necessary overall structure.

Approach

STEP/GARM Constructors

Previous work on supporting different viewpoints on a common, or at least connected, database is not unknown [van Nederveen 91, Willems 91] but it is difficult to find something general enough to adapt to our purposes while specific enough to be useful. [Amor 91], embodying objects in the form of frames, allows slots to maintain multiple values to describe alternative versions or *worlds* within a single system. Such a concept is adaptable to our multiple applications, but must we implement our different viewpoints (*aspects*) at such a low level of granularity? We probably do not need alternative versions of the individual attributes, but instead a way to permit different sets of attributes while also assembling the applications' separate data structures according to some common format or constraints. Perhaps these attribute sets, or aspects, could be arranged in some way that would mirror the relationships between the different applications to which they belong.

Since both our desired overall model and its applications involve multilayered and multiconnected entities, we can begin by turning to a familiar concept [e.g. Batori 85] that uses encapsulated subcomponent assemblies as *implementations* of well-defined *interfaces*, thus allowing both top-down and bottom-up hierarchical design, problem subdivision, and alternative versions of a solution. This device is described as an entity of the GARM standard in [Willems 88]; it embodies a single product or subproduct from both functional and technical views. Thus in Figure 4 analogous procedural and physical structures are modeled as an assembly of elementary or atomic *Functional Units* (*aFU1*, *aFU2*, *aFU3*), each with a set of connections called *Ends*. Components *aFU2* and *aFU3* together form a subassembly or *Technical Solution* (*TS*) which, while indifferent to their internal connection, derives their unconnected Ends as *Ports*. This TS joins to a complex or non-atomic FU (*cFU*) with corresponding Ends which then forms a connection (or *interface* in GARM terminology) with an End of *aFU1* at the higher level. Local revisions or alternatives are introduced by implementing *cFU* with different TSs. There is no fundamental difference in nature between atomic and complex FUs, simply one of state. The design of *aFU1*, say, has not yet advanced to a stage requiring the use of a TS, and may or may not in the future. More specialized FU subtypes, however, may prefer to maintain an inherent distinction in this regard.

Since the Ports of a TS always derive those Ends of its subFUs that remain unconnected at the lower level, a set of components can be encapsulated so that only those external (higher level) connections need be known to the outside world. Splitting the FU and TS into separate entities not only encourages this encapsulation but allows the aforementioned substitution of alternative solutions without affecting the rest of the structure. Conversely, a given TS subtree can be reused any number of times within the same or different structures by attaching it to several different FUs. Such a connection is uncomplicated

because the only other vertical links to account for are those between the Ends and Ports of the same FU/TS pair, and the only external lateral links are between FUs' Ends at the same level (subsequent diagrams visually suggest lateral correspondences below this level, but these are not direct connections). Any actual interaction involving, say, the End to the left of *aFU2* in the diagram must ascend through the deriving Port, up to the left End of *cFU*, across to the right End of *aFU1*, and, were *aFU1* complex, down from there. In this case no further descent occurs. Because FUs must maintain distinct connections through their Ends they cannot be reused as TSs can; similar entities are treated instead as different instances of a single FU subclass.

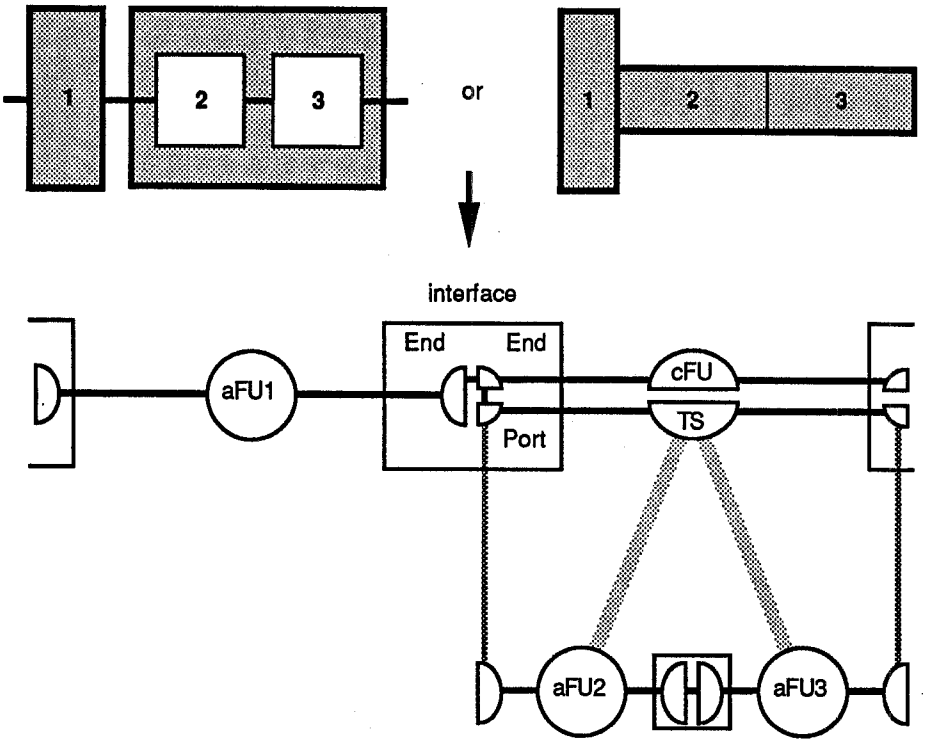

Figure 4 Subcomponent assemblies modeled according to GARM

Topological Issues

[Willems 88] sets out in detail a "meta-topology" of bounded domains that provides a foundation for GARM but no explicit vocabulary of basic configurations. The FU/TS structure exemplified above accounts only for structures whose subcomponents at any given level are connected in series; each juncture involves ultimately just two elementary units. Topologically this corresponds, as shown, to a case of two domains or objects (*1* and *2-3*) whose potential internal decomposition, having no effect on the joint connecting them, is of concern only within, not between, those objects.

One situation that occurs frequently in practice is that of subcomponents joined in parallel, that is with each maintaining a direct connection to the same external entity (as with subobjects *2* and *3* to object *1* in the upper part of Figure 5). Such a multiple juncture of Ends can be represented without difficulty, since in GARM each Port can derive more than one End in much the same manner that a TS can decompose into more than one subcomponent. Thus the Ends of the parallel *aFU2* and *aFU3* find a common external

connection in their derivation by TS's lefthand Port. We add a vertical line joining these two Ends to convey this status visually.

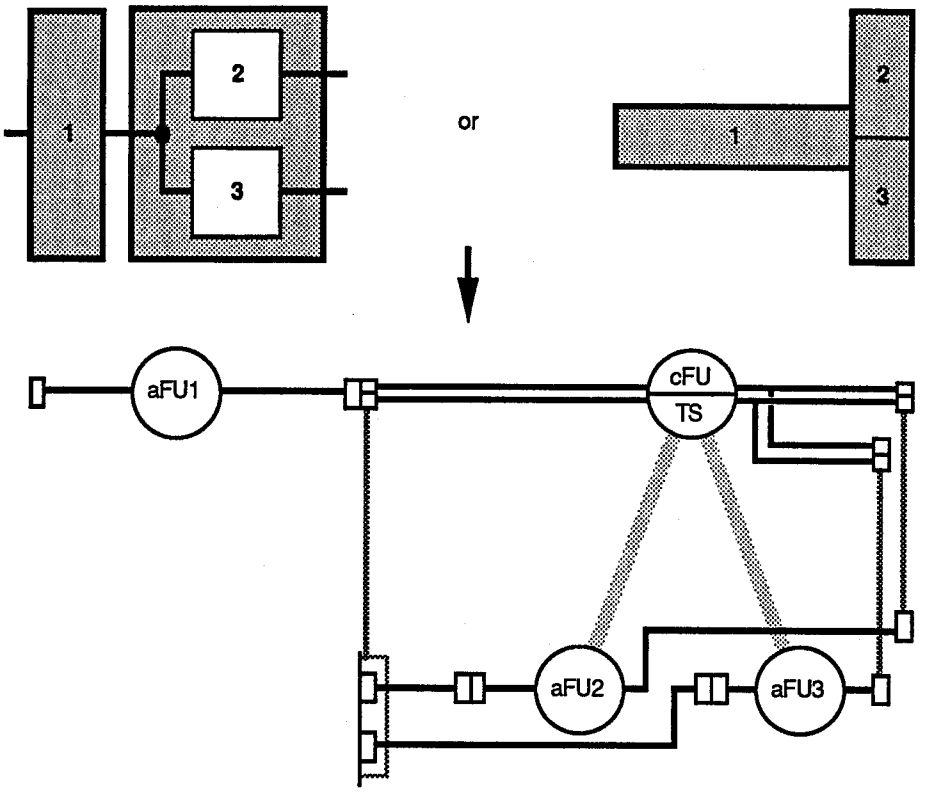

Figure 5 Modeling an assembly with parallel subcomponents

There also exist structures that cannot be gracefully configured as combinations of serial and parallel subassemblies; these would require the addition of extra joints and decomposition levels leaving little conceptual resemblance to the original subjects. In Figure 6, for example, the high-level connection between subassemblies *1-2* and *3-4* conceals, among their components, a set of underlying links between subFUs *1* and *3*, *2* and *3*, *2* and *4*. Such an interface between FUs is not equivalent to a common joint at which all subunits meet. To represent this situation in the GARM structure we can again use multiply-deriving Ports but must preserve the separate connections between pairs of Ends at the lower level. We introduce this alternative form of derivation, embodied in a new type *MultiPort*, for all such cases which, unlike parallelism, are describable only in terms of subconnections between and thus external to separate TS assemblies.

A Formal Protocol for Constraining Object Relationships

FU and TS relationships are described in [Willems 88] using the IDEF-1x model, which easily converts to the popular NIAM notation [Nijssen 89] to clarify formally the relationships among the representation's object types and the constraints that control their assembly. According to Figure 7, each TS can implement several FUs while comprising several subFUs; similarly a Port can implement several Ends while deriving several subEnds. A TS may have several Ports, while each FU may have several Ends. Finally, each End may be mated with at most one other End. Note that this will be an internal connection and prevent either End from being derived by a Port of the TS containing it,

since as described in the section on STEP/GARM Constructors only unmated Ends ascend to the higher level.

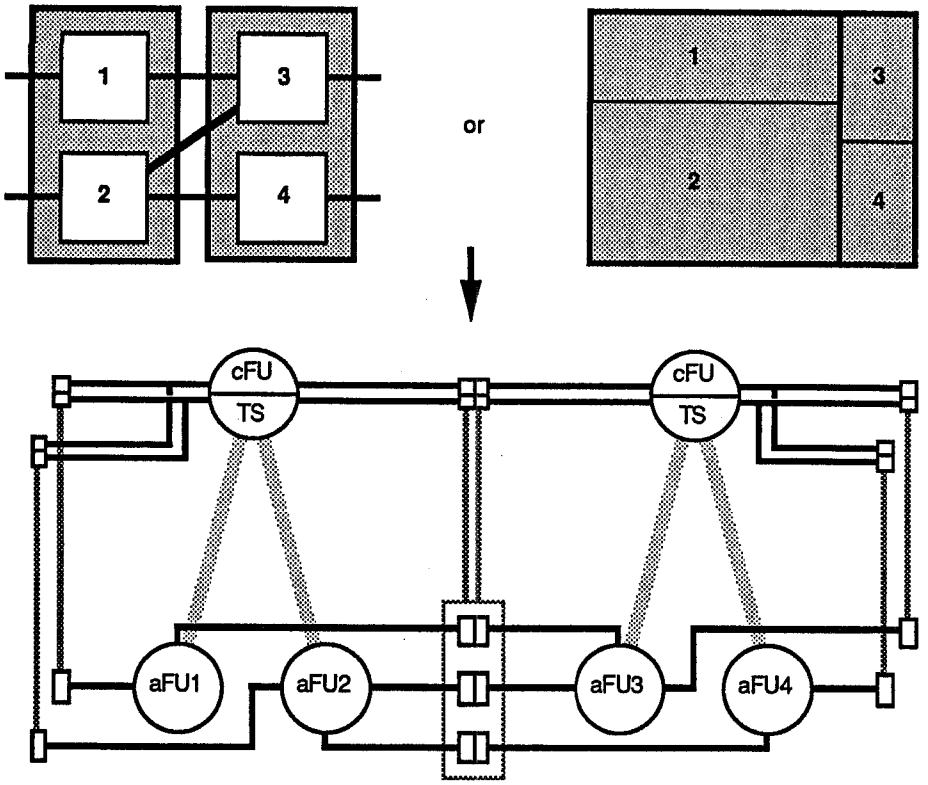

Figure 6 Modeling components with complex connections

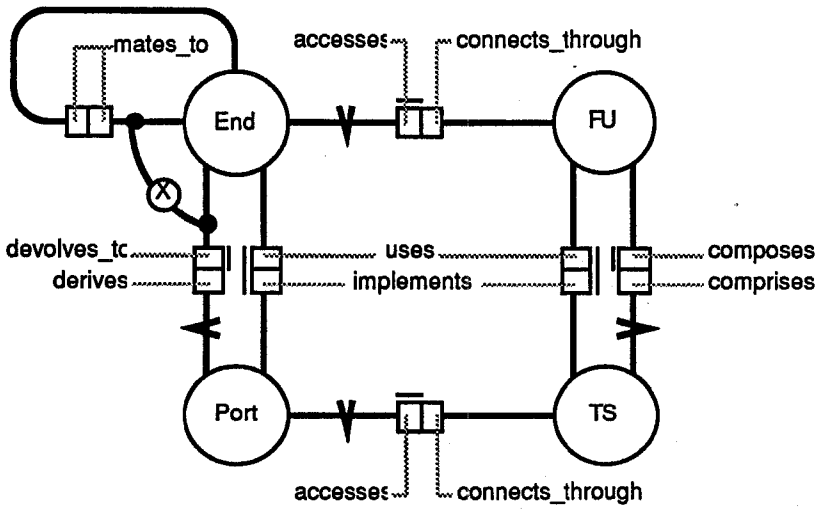

Figure 7 FU/TS relationships represented in NIAM

Representing Hierarchical Applications in GARM

Cost Model

Adapting the GARM format to the previously described applications is straightforward. To represent building costs, simple items such as sand and labor, shown in Figure 2 as basic resources, become atomic FUs, while each category such as superstructure or product such as concrete becomes an FU/TS pair. Within the latter, a complex FU represents the specification in the upper half of each divided box and a TS its categorization or implementation process in the lower half. Figure 5 shows the structure for Figure 2's cement production subtree. The TS for cement has seven FU subcomponents, four of which, an explosive, electricity, and human and freight transport, are considered manufactured products heading their own subtrees. On each side of the seven ingredients, their Ends are derived by a common Port of the cement TS, thus linking them together as parallel siblings and controlling cost evaluation.

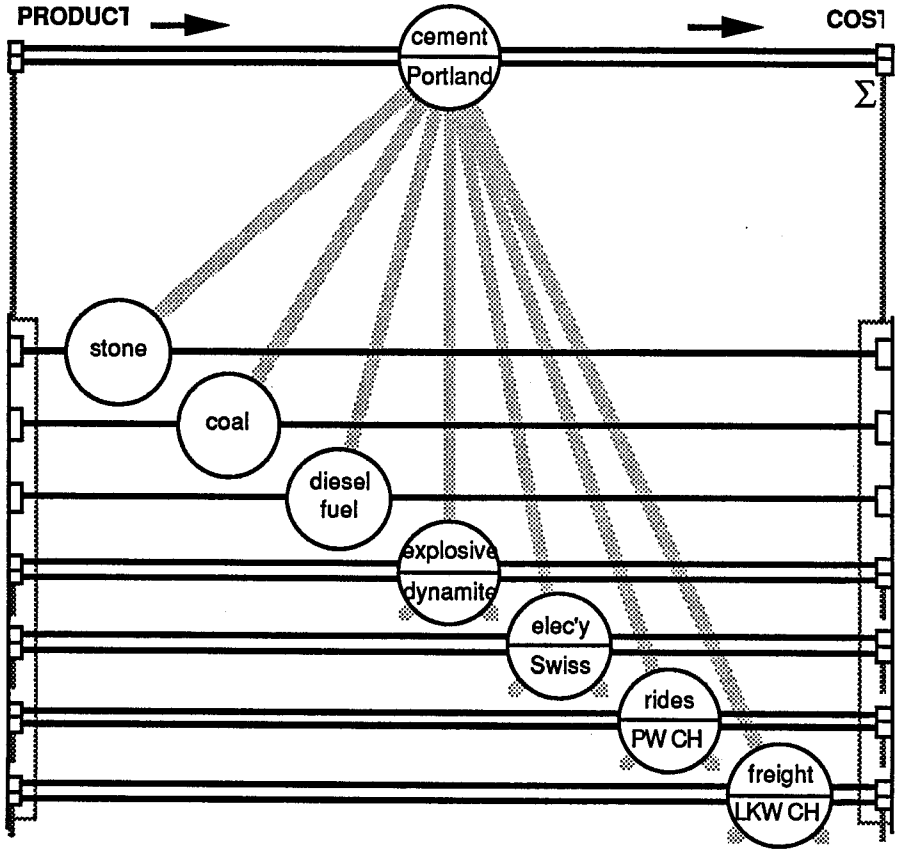

Figure 8 GARM representation for cement production

The construction costs of each component in money, waste, emissions, and byproducts are requested through its lefthand End and returned through the right. When a TS subtree (Portland cement, or the building itself in the complete structure) receives a request for evaluation from one of the FUs it implements (cement), it propagates the request through its lefthand Port down to its subFUs. Each of these receives the request from its lefthand End, calculates its cost on its own (if atomic) or by evaluating its own TS (if complex), and

sends the result out through its righthand End. These subtotals are added by the righthand sigma-Port of the first TS, which returns the final result to the requesting FU. This evaluation process descends recursively as deeply as required to traverse the subtree. Any TS reused by different FUs will be queried and summed more than once; for these subsequent occasions, however, it simply returns the preserved result of the first evaluation without retraversing its subtree.

Representation of costs for phases in a building's life cycle other than construction differs little from the above approach. Where appropriate, the FU at a particular node possesses alternative TSs, each at the top of a subtree representing costs for the corresponding phase. This branching occurs chiefly at the fourth (IP-BAU component) level of Figure 2's building tree, since the nodes above this define categories independent of phase, and those below ingredients and resources that would usually be new whether used for construction or maintenance. A cracked concrete wall, for example, would be patched with new concrete. A subtree for demolition costs contains "ingredients" such as labor and transport required to dismantle and/or remove a component, along with such destructive materials as paint remover or dynamite. Certain phases of a node may use more than one of these alternative TS subtrees; thus replacing windows during refurbishment requires the demolition (removal) of the old windows followed by the installation of new ones. A component node contains evaluation methods for each phase; these know which TS or combination of TSs to invoke.

Access Model

The previously described access structure for building refurbishment also adapts readily to the GARM format. The shaded part of Figure 3b's house network diagram produces the two-layer FU/TS structure of Figure 9. When the exterior view is decomposed into the rooms and room-oriented walls of the interior view, subconnections arise to complicate the topology. The high-level interface between the south wall *WS* and the interior *I* is seen to contain, at the lower level, adjacency links between room *R1* and southwest wall *W1S*, room *R2* and southeast wall *W2S*, partition wall *W12* and *W1S*, *W12* and *W2S*. This is in addition to the internal connections within the composite walls and interior, and the respective allotment of the south window *wS* and door *dS* to the south wall's individual components *W12* and *W2S*.

By incorporating new branches for surrounding elements, a larger tree could show the components of the external spaces—yards, sidewalks, and streets, perhaps, or corridors and other units in the case of apartments—and their relationships with those of the house. Access to the latter from a given external subspace would be through those adjacent exterior walls containing doors and windows. Higher levels in the structure could represent the further agglomeration of housing units described in the section on Access Model for Refurbishment.

Class Extension

The GARM structures discussed here are built from entities defined as object classes and their instances. These classes suffice for the formation of these structures, but to embody their functionality within applications they must be extensible to specialized subclasses. Figure 10 shows the GARM class hierarchy from its root down to the cost model. Beginning on top with an *Objectt* base class, we first define classes for construction of an application-independent GARM framework: *Nodes* to form components and *Connectors* to join them. At the next level these become the familiar FUs and TSs, Ends and Ports. These GARM classes contain the attributes and functionality to represent component hierarchies and the binary relationships of Figure 7's NIAM graph; they also enforce

restrictions ensuring necessary arities and class compatibilities and preventing improper constructions. Thus Ports and Ends are created only through existing TSs and FUs, never as independent entities, and two Ends are connected only through the TS of which they are both immediate components.

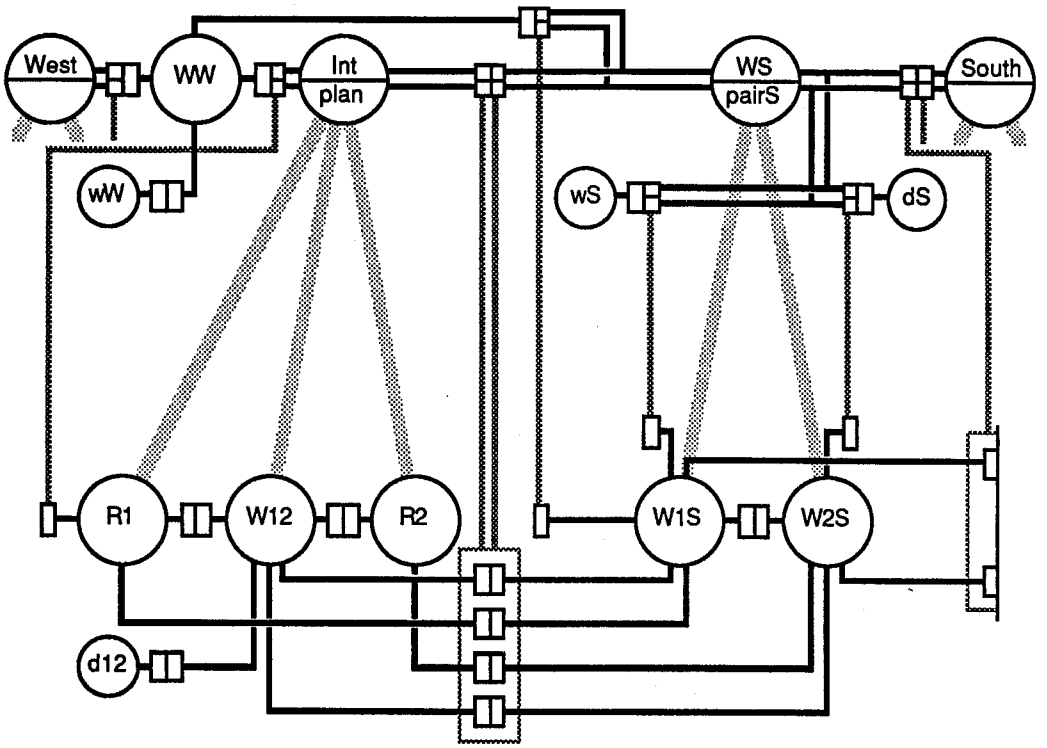

Figure 9 GARM representation of room access structure

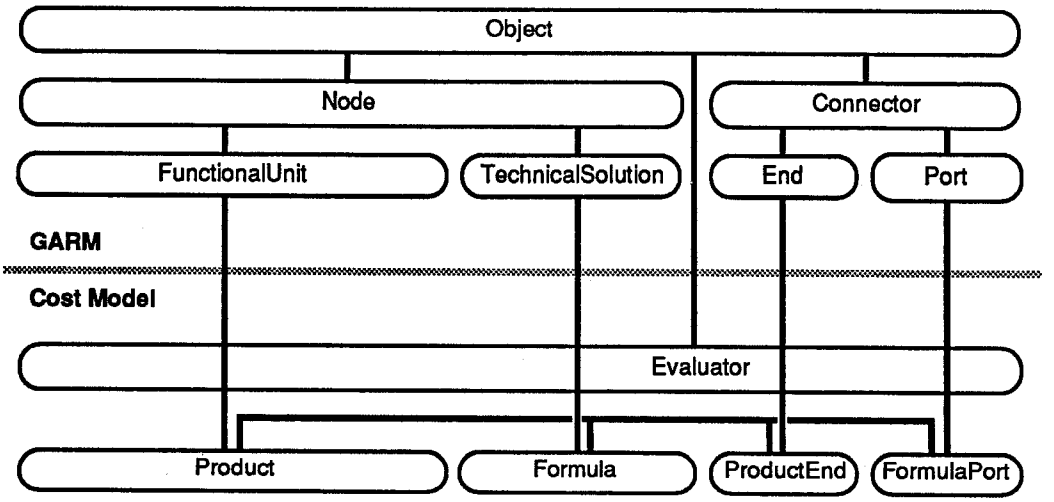

Figure 10 Hierarchy of general and specialized classes for building cost model

Descending to the level of classes peculiar to the cost application, every purchasable item, substance, or service, whether basic or manufactured, is in the *Product* subclass of FU which provides attributes relevant to cost but independent of a particular solution. These

include a name and description, a unit of measurement, and the quantity, particular to each instance, used as an ingredient by the parent TS to produce one unit of the latter's product. If atomic the Product will also contain an aggregate unit cost, which for a composite Product is calculated as already described from its TSs. This latter summation process, which totals the costs of subcomponents and adds a (usually environmental) cost inherent to the solution, is defined in the subclasses *Formula.*, *ProductEnd*, and *FormulaPort*. Finally, the *Evaluator* provides features general to the application. This class hierarchy is directly transportable to an object-oriented language such as C++, in which a first version of the cost evaluator has been implemented.

Unifying Applications

A Collective Data Model

We have now devised different structures using GARM's FU/TS units. Can we connect these applications into a single coherent model? Each viewpoint requires different information, and hence different attributes, from its subcomponents, but we would like to avoid both the disorder of piling information for all applications into one heaping object and the redundancy that might result if complete and separate aspects were simply lined up side by side at a single level. If we organize applications hierarchically according to their interests, as in Figure 11, we reduce this possibility. Thus separate subapplications for construction and maintenance estimates might share data on building product costs, while subapplications for traffic analysis and subsequent refurbishment scheduling would both refer to accessibility data. In the resulting GARM structure (Figure 12), linked Ends might express scheduling relationships such as precedence (as here) or concurrence, or any connections between applications at a single level. Thus construction must precede maintenance, traffic analysis precedes job scheduling, and, at the more general level, cost estimation comes before access analysis.

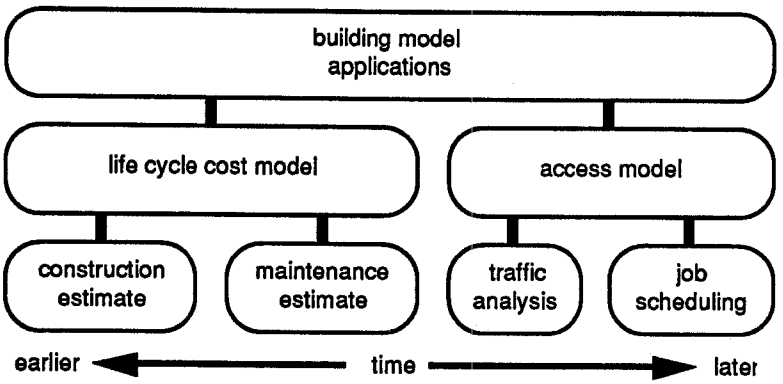

Figure 11 Organizing aspects and applications within a building model

The resulting tree structure is applicable in two ways: as a framework for aspects and as a blueprint for organizing applications at different levels of specialization. Each application could operate directly on the relevant collection of attributes, while that aspect takes the place of the object itself within the application's hierarchy. This helps untangle the situation of, say, a wall entity embedded simultaneously in the cost, access, and other structures, which as a conceptual arrangement is feasible but rather dense. With this approach the participation of the object is hidden until the aspect for the application is retrieved. For example, two subapplications of a building topology structure might operate on the same room, one interested in its traffic flow, the other in the access it provides to the

electrical system. To implement this, the base room class could define as a member a structure of aspects, each a group of attributes and methods relevant to a particular application, the whole grouped into a specialization tree. A new application might then define not a new instance of a room but a new aspect of an existing room. This unique aspect would actually occupy the appropriate position in the application's data structure in place of the commonly used object itself. Some contexts such as cost estimation, interested in quantities rather than individual components, might never need more than one instance of each class. An additional advantage of grouping attributes into aspect objects is that a collection of applications can be built in increments. This approach might begin by defining a single aspect in each object class, or a collection of members to be tucked into a separate aspect as later domains are developed.

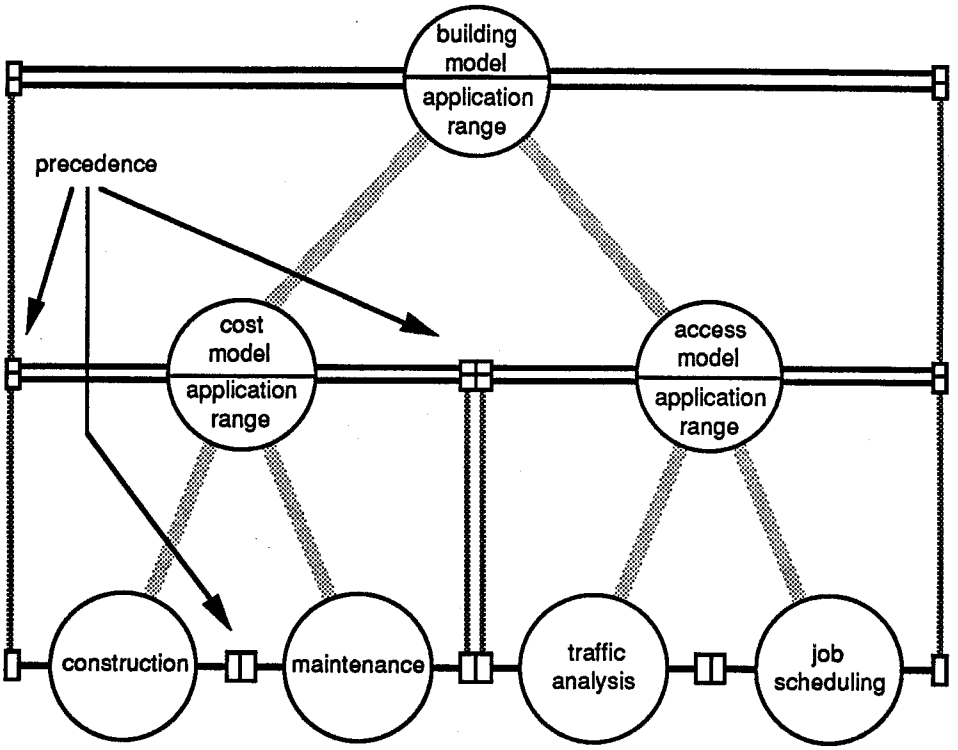

Figure 12 GARM representation for different aspects and applications

Maintaining Integrity

Once different applications establish common interests and begin to share data they will in most cases require a mechanism to preserve the integrity of their unified structure. When in a CAD tool, for example, the user changes the layout of a room, this will alter the sizes of its floor and walls, which will in turn alter the amount of materials required for that part of the building. This, of course, affects any subsequent evaluation of construction and maintenance costs. Interrelated data, then, must be connected by a set of constraints as discussed briefly by Björk as a factor in unification and at more length by such as [MacKellar 91]. In the latter the entire design process is treated as a continuing attempt to satisfy sets of constraints. Each stage of the design is associated with such a set and cannot be completed without satisfying its conditions. One stage may include as a prerequisite the successful completion of a previous stage, which may itself depend on a still earlier stage, thus enforcing a linear order in the design tasks. Alternatively, a stage may depend on several otherwise independent stages which can be developed simultaneously or in any

order desired. It is easy to see how this structure of constraints can be adapted to the GARM format, with each design stage as an FU implemented by one or more TSs, each decomposing into a set of previous stages connected in series or parallel. Method calls and other independently confirmable predicates can appear as atomic FUs in the same hierarchy.

Conclusions

The framework described here promises to assist in orderly and consistent development of a number of related applications in building design. This development has proceeded as far as a working model of building cost evaluation, produced by extending the GARM classes that themselves remain generic enough to use in additional domains as well. The next step towards implementation of the ideas discussed here will be the construction of a database of process objects to represent the costs of transforming basic materials and resources into ingredients for the manufacture of components. This will contain data, gathered from industry, reflecting the building cost hierarchy of Figure 2 from its lowest levels up to but not including the IP-BAU level. However, while several hierarchical configurations and situations presented here have been worked out in some detail, other areas remain vague and must soon be elaborated upon. Aspect and constraint organization must be dealt with in more depth to effect firmer and more useful relationships. Object definitions must become more precise. For now, the GARM model as applied here appears to be a comprehensive and flexible foundation on which to base future development.

References

[Amor 91] Amor, R. W., *ICAtect: Integrating Design Tools for Preliminary Architectural Design*, Computer Science Department, Victoria University of Wellington, New Zealand, Jan. 1991.

[Batori 85] Batori, D. S., and W. Kim, "Modeling Concepts for VLSI CAD Objects", *ACM Transactions on Database Systems*, 10(3), pp. 322-346, Sept. 1985.

[Bjork 92] Björk, B.-C., "A Unified Approach for Modelling Construction Information", *Building and Environment*, special issue on databases for project integration, 1992 (to be published).

[CRB 91] *CCE Cost Classification by Elements*, CRB Swiss Research Centre for Rationalization in Building and Civil Engineering, Zurich 1991.

[Glardon 91] Glardon, C., L. Kling, and N. Kohler, "Building Refurbishment as a Multi-Actor Process", *Proceedings of the Second International Workshop on Computer Building Representation for Integration*, June 1991.

[IPBAU 91] *Elementgliederung für Erneurung und Unterhalt*, Impulsprogramm: Bau-Erhaltung und Erneuerung, Lausanne 1991.

[Kohler 91] Kohler, N., "Modelisation of a Building During Its Life Cycle", *Proceedings of the CIB W78 Seminar: The Computer Integrated Future*, Sept. 1991.

[MacKellar 91] MacKellar, B. K., "A Constraint-Based Model of Design Object Versions", *Proceedings of the Third International Conference on Data and Knowledge Systems for Manufacturing and Engineering*, pp. 285-304, Sept. 1991.

[Nijssen 89] Nijssen, G. M., *Conceptual Schema and Relational Database Design: A Fact Oriented Approach*, Prentice Hall, 1989.

[van Nederveen 91] van Nederveen, G. A., and F. P. Tolman, "Modelling Multiple Views of Buildings", *Proceedings of the CIB W78 Seminar: The Computer Integrated Future*, Sept. 1991.

[Willems 88] Willems, P.H. "A Meta-Topology for Product Modeling", *Proceedings of the CIB W74 + W78 Seminar: Conceptual Modelling of Buildings*, pp. 213-221, Oct. 1988.

[Willems 91] Willems, P.H., P. Kuiper, G. T. Luiten, B. F. M. Luijten, and F.P. Tolman, "A Framework for Evolutionary Information Model Development", *Proceedings of the CIB W78 Seminar: The Computer Integrated Future*, Sept. 1991.