CIFE CENTER FOR INTEGRATED FACILITY ENGINEERING

SPACE CAKE

System for Project mAnagement in Civil Engineering using Computer-Aided Knowledge Engineering

by

Gijsbertus T. (Bart) Luiten
and
Martin A. Fischer

CIFE Working Paper
Number 40

May, 1995

Stanford University

If you would like to contact the authors please write to:

c/o CIFE, Civil Engineering,
Stanford University,
Terman Engineering Center
Mail Code: 4020
Stanford, CA 94305-4020

Preface

This working paper combines four documents related to research into automated and integrated management of scope, budget, and schedule in a building project. The research is ongoing at the Construction Engineering and Management Program at the Department of Civil Engineering of Stanford University in close combination with the Center for Integrated Facility Engineering (CIFE) of Stanford University.

The first document gives an overview of SPACE CAKE. It discusses the theoretical background of automating and integrating construction management, an automation and integration approach that is based on project modeling, and a prototype system that implements this approach. This document has been submitted to the Automation in Construction journal for publication.

The second document was written in the beginning of the research project. It contains the goals of the project, a first analysis of the construction management process, and two example projects worked out on paper.

The third document contains some background and a manual for SME+, the extended Semantic Modeling Extension.

The fourth document contains a manual for SPACE CAKE, the System for Project mAnagement in Civil Engineering using Computer Aided Knowledge Engineering.

Bart Luiten
Martin Fischer
May 1995

SPACE CAKE

System for Project mAnagement in Civil Engineering using Computer Aided Knowledge Engineering

By Gijsbertus T. (Bart) Luiten

April 1995

Construction Engineering and Management Program
Department of Civil Engineering
Stanford University
Stanford, CA 94305-4020
USA
e-mail: Luiten@CE.Stanford.edu or Luiten@HBG.NL

Department of Civil Engineering
Delft University of Technology
Delft
The Netherlands

Contents

1 Introduction

This document describes the Scope, Time, and Cost Management System (also called **SPACE CAKE**, System for Project mAnagement in Civil Engineering using Computer Aided Knowledge Engineering). The document serves as a manual for developers that use the system and as a archive for future generations that have to live with the results of those developers.
The document first describes the architecture of the system. The main system of a generic model (cmim) and the five modules (ib, am, sa, sc, and ec) that are implemented in the system. For each of these models or modules the paper describes: the goal, the knowledge (or information structure) implemented, and a manual. It then shows how these five modules are combined in one model (comb). This combined model is specialized to specific types of building projects, i.e., concrete structures (cs) and apartment complexes (ac). The concrete structures model is used to demonstrate the whole system. The apartment complex model is an example of flow-based planning.
The last part of the document shows how you can start your own model or module and plug it in the system and how you can update the generic cmim model. It ends with a discussion of future work that can be done to complete this system.
In appendix A you can find some experiences I had while working with Kappa that can be useful for others too.

2 Architecture of the System

Figure 1 shows the five project management tasks and the information flows between them in an IDEF0 diagram.

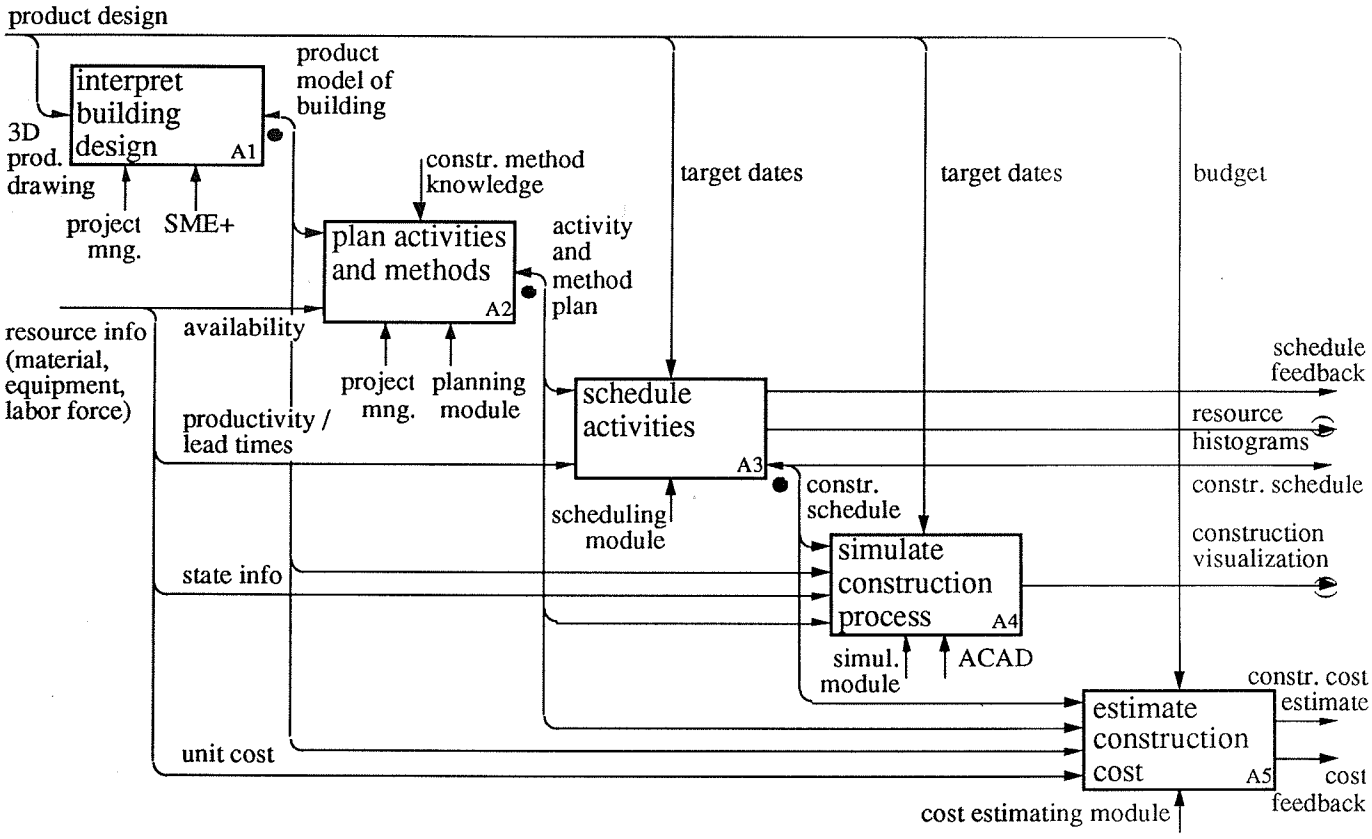

Figure 1 The five modules of the Scope, Time, and Cost Management System.

These project management tasks are supported with a several computer systems.
Most of the reasoning is implemented in Kappa in the so-called Scope, Time, and Cost Management System. Fig. 2 shows the four layers of this system.

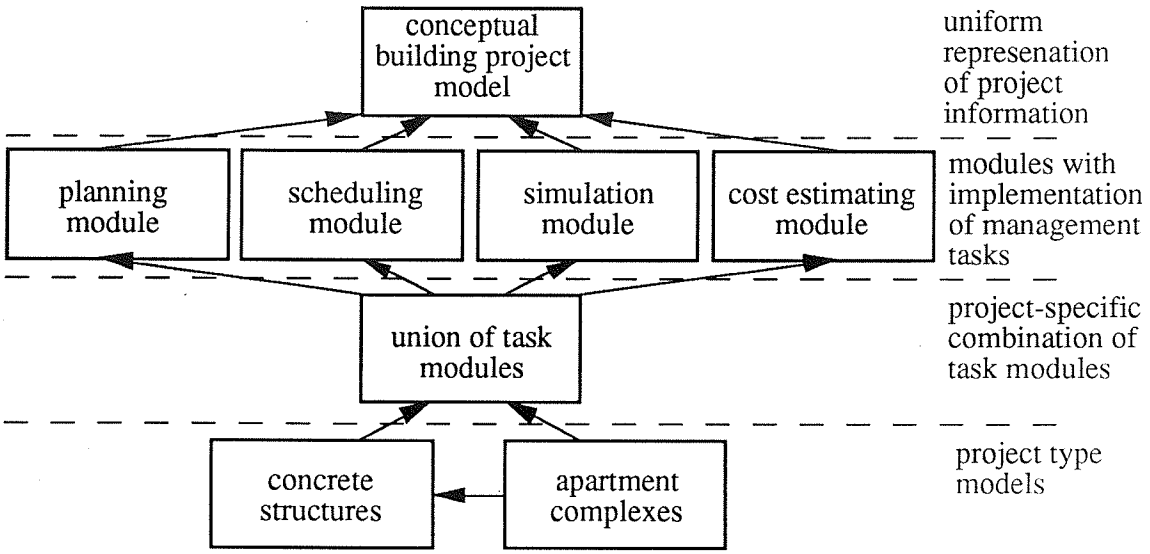

Fig. 2. Architecture of the four-layered Scope, Time, and Cost Management System.

A generic data structure applicable to all construction projects is implemented in cmim.

cmim generic model containing the Construction Management Information Model

The five modules of our system extend this generic data structure with functionality specific for a phase in construction management.

ib SME+ for Interpret a Building
am module for generation of the Activity-Method model
sa module for Scheduling the Activities
sc module for Simulation of the Construction process
ec module for Estimating the cost of Construction

The figure shows that the first module (ib) uses SME+, an AutoCAD extension that enables interpretation of 3D geometric objects (Clayton et al. 1994). The other modules use Kappa. For visualization of the simulation a home-made AutoCAD extension is used[1].

The functionality of these five modules is combined in the comb module.

comb model that combines the functionality of the 5 modules

Models applicable to specific types of building projects are implemented in:

cs model for the realization of Concrete Structures, and
ac model for the realization of Apartment complexes.

These two models specialize product, resource, and ConstructionMethod classes. Most of the knowledge for a type of building projects is implemented in the ConstructionMethods.

Notes for the programmer:

- To identify the origin of classes easily, each class should start with a two letters plus an underscore prefix (e.g., classes defined in cmim all start with "CM_").

[1] SME, SME+ and the visualization tool can be found in /home/users/luiten/SME and the Kappa directories in /home/users/luiten/pk.new, both on the Sundiver in CIFE.

- Each model or module is stored in its own directory with a name that corresponds to the model or module name. You can find those directories in /home/users/luiten/pk.new on the Sundiver file server (which can be reached from all SUN stations at CIFE).

3 CMIM: Construction Management Information Model

3.1 Goal

Define a generic information structure that can be used in the other modules and provide general functionality regarding this information structure.
For example, the Kappa implementation of this model enables the users to read and write data from STEP files. It also provides functionality to derive information from the generic data structure, such as a method to find the leaf activities (i.e., the lowest level of activities) in the activity-method tree.

3.2 Knowledge Implemented

Information Model
Figures 3 and 4 show respectively the hierarchy of cmim classes as implemented in Kappa and their relations in a NIAM diagram.

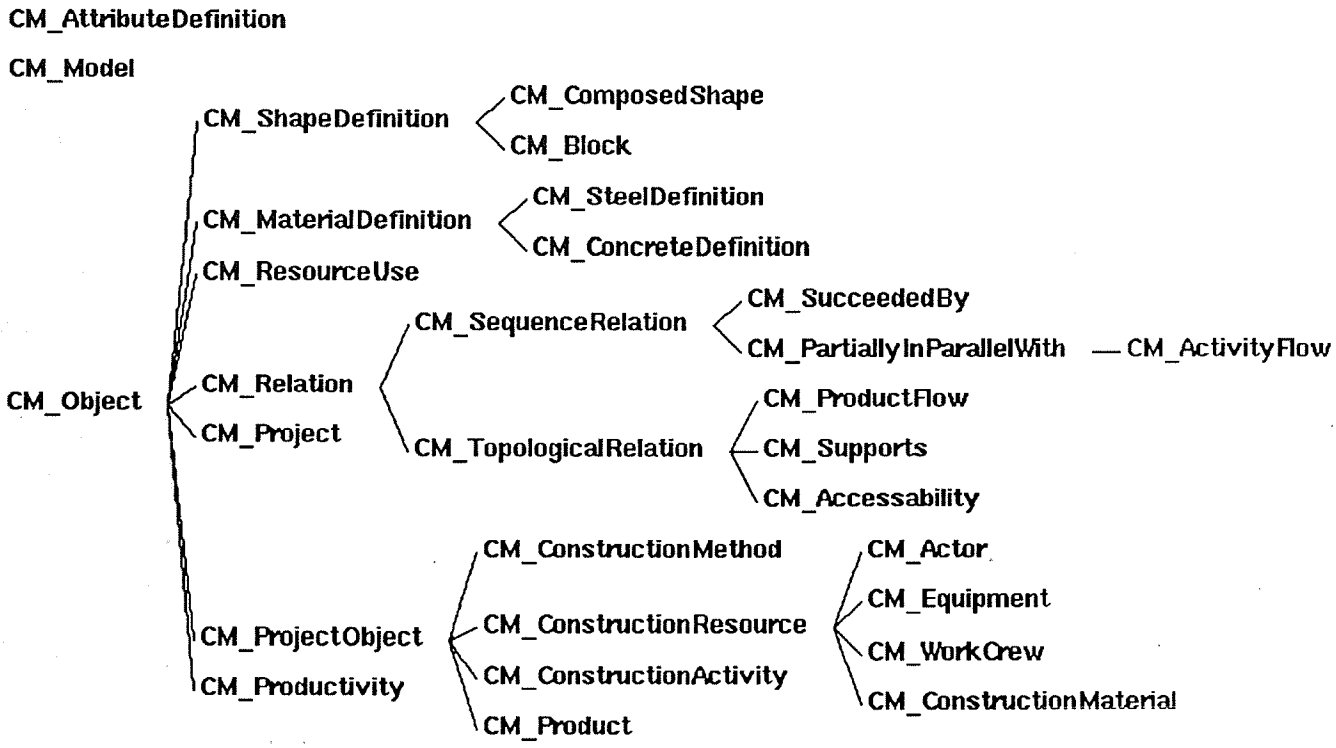

Figure 3 Class hierarchy in the construction management information model (cmim) as implemented in Kappa.

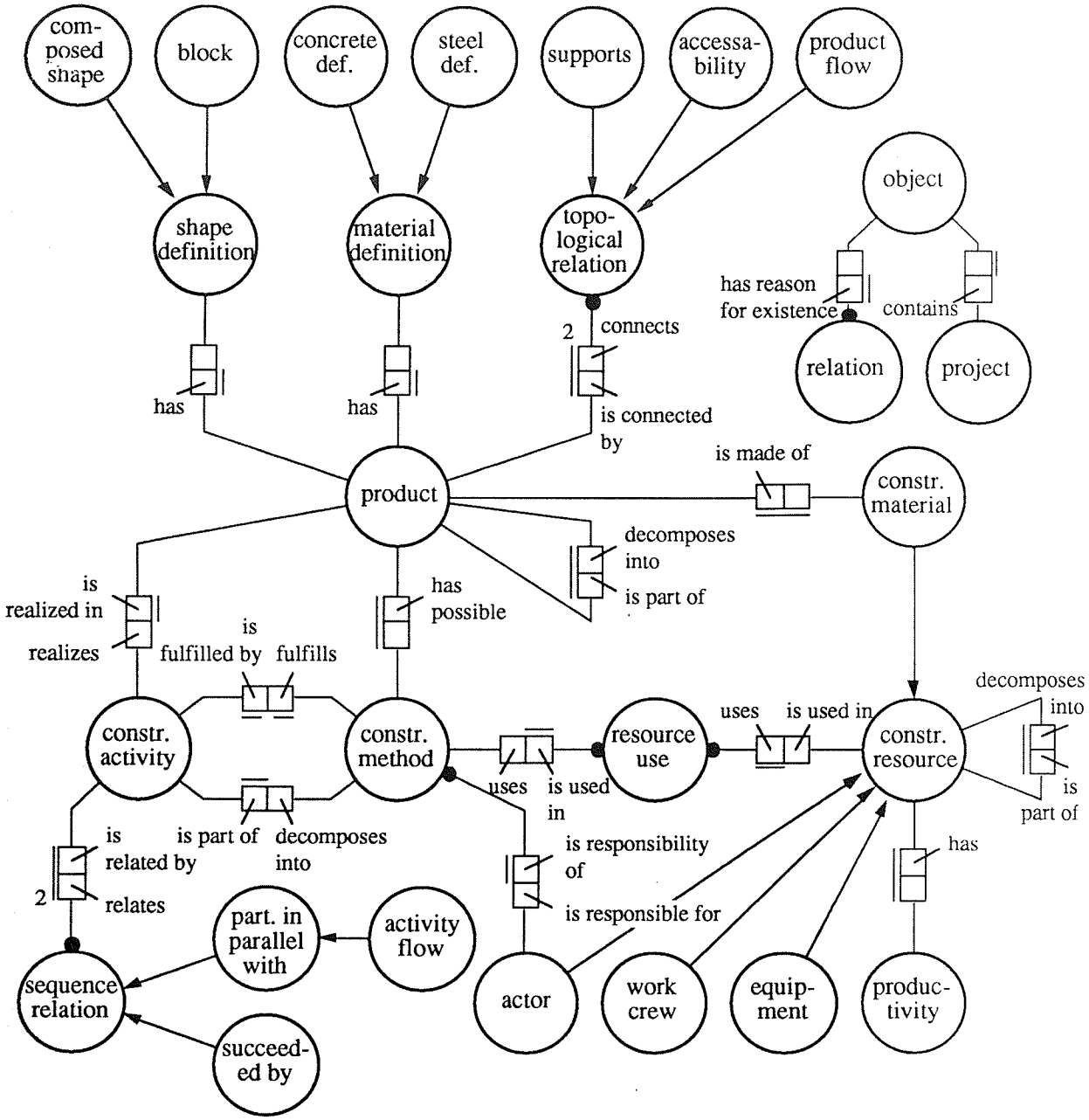

Figure 4 Cmim classes and their relations modeled in NIAM.

3.3 Manual

Read/write STEP files
To exchange information between the modules and to store information in a system independent way, information has to be stored in plain ASCII files. These ASCII files can be read and written by most computer applications and exchanged between different platforms. Because I did not feel like inventing yet another file format, I chose the STEP physical file format (ISO/TC184 1993).

With future extensions of the system it should be possible read and write data from (relational or object-oriented) databases, such as Oracle.

To read a STEP file:

```
instance of CM_Model:          ReadSTEPpf! (filename, ReadDefinition,
                               CleanClasses, ReadData, CleanObjects);
ReadDefinition, CleanClasses, ReadData, CleanObjects are integers; 0
means NO, 1 means YES;
use ReadDefinition = 1 and CleanClasses = 1 only if you read a STEP file
not produced by the system and that has another sequence of attributes;
use CleanObjects = 1 if you want to start over again;
use CleanObjects = 0 if you want to read data from a second database
(e.g., a resource database).
```

To write a STEP file (first read a file with the latest definition of attributes):

```
instance of CM_Model:          ReadSTEPpf! ("cmim/cmimdef.spf", 1, 1, 0,
                               0);
instance of CM_Model:          WriteSTEPpf! (filename, WriteDefinition,
                               WriteData);
WriteDefinition, WriteData are integers; 0 means NO, 1 means YES;
use WriteDefinition = 1 if you want to write a STEP file to be read by a
system that uses (or might use) another sequence of attributes.
```

FindClassWithName

To be sure that you always use the latest redefinition of a class (e.g., when you create a new instance) you can use FindClassWithName (in Kappa). Use the class name without the prefix. This function returns the class that is the latest redefinition found in the present models or modules (or, in other words, the class that does not have subclasses with the same name).

For example, to find the latest version of ConstructionActivity when you want to create a new activity instance that uses the latest redefinition, use something like (in Kappa):

```
?app = ObjectApp (?self);
?act_class = FindClassWithName ("ConstructionActivity");
?act = MakeInstance ("test_act", ?act_class, ?app);
```

Derive information from general data structure

The following Kappa methods are defined:

Find the top product (i.e., the highest level product) in a project (and set HasTopProduct):

```
instance of ..Project:         FindTopProduct! ();
```

Find the leaf activities of a activity-method tree (and set HasLeafConstructionActivities):

```
instance of ..Project:         FindLeafActs! ();
```

Find the predecessor (and automatically the successor) activities of all activities:

```
instance of ..Project:         FindPredecessorsOfAllActs! ();
```

Find the predecessor (and automatically the successor) activities of all activities:

```
instance of ..Project:         FindPredecessorsOfAllActs! ();
```

Find the part-activities of all activities:

```
instance of ..Project:         FindPartActsOfAllActs! ();
```

Find the start and finish activities (and set HasStartActivity and HasFinishActivity):

```
instance of ..Project:         FindStartFinishActs! ();
```

Find the predecessors of an activity (and set Predecessors):

```
instance of ..ConstructionActivity: FindPredecessors! ();
```

Show a graph of the scheduled use and the availability of a resource:

```
instance of ..ConstructionResource: ShowUseGraph! ();
```

4 IB: Interpret Building

4.1 Goal

Interpret a building from a 3D AutoCAD drawing to a symbolic product model. You can also aggregate objects here.

4.2 Knowledge implemented

- Interpreting components in a (3D) drawing and aggregating the components of a building into work packages (i.e., work packaging and zoning) by introducing spatial elements. In the current implementation there is no knowledge implemented to do this automatically. However, in the AutoCAD-SME+ environment the users is able to define zones, floors, building-blocks and buildings graphically.

4.3 Manual

Actions in AutoCAD-SME+ .

Load 3D drawing:

```
acad <filename> &
```

Load SME+:

```
(load "edsm")
```
[1]

Load model:

```
Semantics.Open: select <model>
```

Add spatial objects (such as floors and zones; don't forget one overall product):

```
See AutoCAD manual, use
Model.Primitives
```

Classify components and spatial objects (using the semantics pull down menu as in figure 5):

```
For each class:
Semantics.Preferences.DefaultFeatu
re (to set the default class) &
Semantics.AddFeature (to classify
the instances of that class)
```

Add non-graphical information (using the interpretation popup menu as in figure 6):

```
Semantics.Interpretation: set
attribute values for instances
```

Derive geometrical information (using the interpretation popup menu as in figure 6):

```
Semantics.Interpretation: click
"Derive Attributes" button
```

Write information to STEP file:

```
(DSMWriteSTEPpf)
```

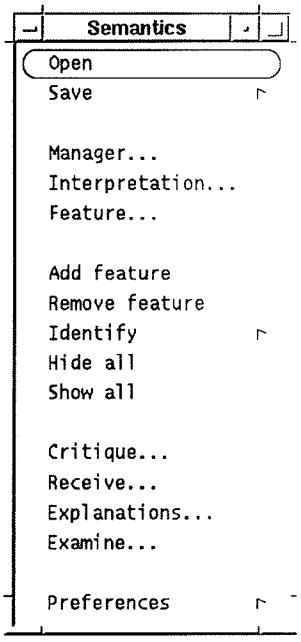

Figure 5 Semantics pull down menu in AutoCAD-SME+ (developed by Clayton et al. (1994)).

[1] When you copy the content of my acad.lsp file to yours, SME, SME+ , and the visualization tool will be loaded automatically.

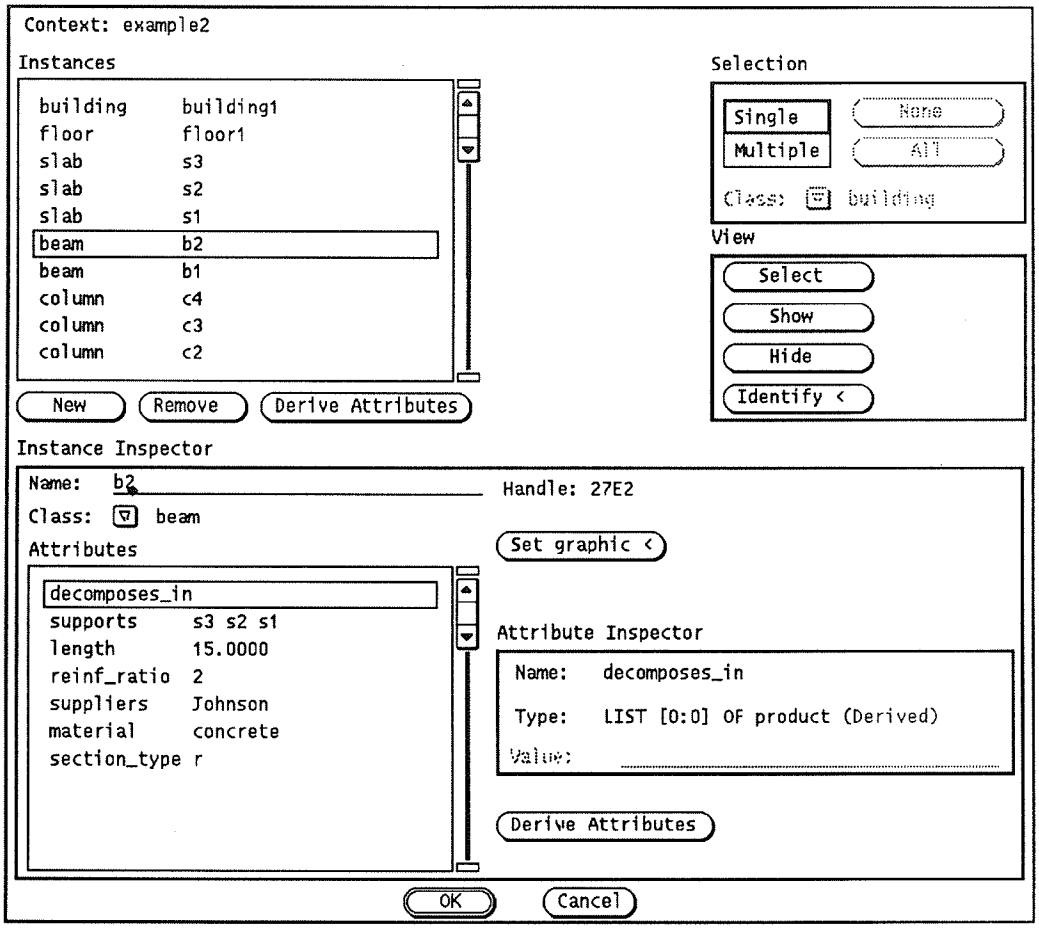

Figure 6 Interpretation dialog box in AutoCAD-SME+

5 AM: Generate Activity-Method model

5.1 Goal

Selection of ConstructionMethods and decomposition of the ConstructionMethods into Activities based on the Product, TopologicalRelations between the part-Products, available Resources, preferences of the user, and construction knowledge. If necessary for a ConstructionMethod, you can also decompose a Product further (e.g., to research how much value Activities of a very detailed level add) or add a decomposition layer (e.g., to divide components on a floor in zones). Resources are allocated to the selected ConstructionMethods.

5.2 Knowledge implemented

The following knowledge is implemented in the am classes shown figure 7:

- Selection of a ConstructionMethod for an Activity and allocation of Resources to the selected Method. Each Product class has a default list of possible ConstructionMethods in order of preference. The user can change the Methods in this list or the sequence of the Methods per instance or per class. Each ConstructionMethod has a Kappa method, CanFulfillActivity!, that assesses whether the ConstructionMethod can be used to fulfill

an Activity. This assessment is based on the Product (type, material, shape, parts), the Resources available, and the ConstructionMethod choices made on higher levels. The first ConstructionMethod in the list of possible ConstructionMethods of the realized product that passes the assessment is instantiated and linked to the Activity. When assessing the availability of the Resources, the Resources needed are allocated to the selected ConstructionMethod.
The default Kappa method, CanFulfillActivity!, assesses a ConstructionMethod as not appropriate for an Activity. For each subclass of ConstructionMethod the Kappa method has to be redefined. For the example in chapter 10, this is done for the ConstructionMethods that realize spatial or structural elements.

- Decomposition of the Product (if different from designed Product decomposition), definition of TopologicalRelations between part-Products (if different from designed TopologicalRelations), decomposition of a ConstructionMethod into part-Activities, and definition of the SequenceRelations between the part-Activities. Each ConstructionMethod has four Kappa methods that take care of these tasks: DecomposeProduct!, RelatePartProducts!, DecomposeActivity!, RelatePartActivities!, and a Kappa method that combines the four tasks: DecomposeIntoConstruction-Activities! Either the separate task Kappa methods or the combined Kappa method can be redefined for specific ConstructionMethods.
As default, DecomposeProduct! and RelatePartProducts! don't change anything to the designed Product decomposition structure. The default Kappa method DecomposeActivity! only looks at the parts of the realized product. For each part-product a corresponding activity is created. The default Kappa method RelatePartActivities! translates TopologicalRelations between the part-Products directly to SequenceRelations between the corresponding part-Activities. SupportedBy and Accessibility are translated to SucceededBy, ProductFlow to ActivityFlow. (NB In the scheduling module the ActivityFlow relations are translated to secondary relations on a lower level.)
When a more sophisticated decomposition is required, the Kappa method has to be redefined. This is certainly necessary for more detailed levels.

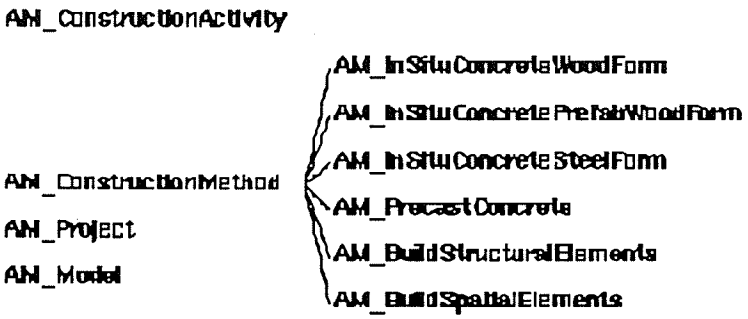

Figure 7 *Class hierarchy in the activity-method (am) module.*

5.3 Manual

Actions in Kappa:

Read output from ib module (this does not work, sorry):

```
instance of CM_Model:          ReadSTEPpf! (<filename>, 1, 1, 1, 1);
```

Read resource information:

```
instance of CM_Model:          ReadSTEPpf! (<filename>, 1, 0, 1, 0);
```

Initialize:

```
create instance of CO_Project;
instance of CO_Project:        FindTopProduct! ();
instance of CO_Project:        AddActivityToTopProduct! ();
```
Define activity-method tree recursively starting from highest level activity:
```
instance of CO_Project:        ElaborateTopActivity! ();
```
Inspect results:
```
instance of CO_Project:        FindPredecessorsOfAllActs! ();
Invoke a "Slot Graph" to view the "Predecessors" (or "Successors") slot
of any of the instances ConstructionActivity.
instance of CO_Project:        FindPartActsOfAllActs! ();
Invoke a "Slot Graph" to view the "DecomposesIntoConstructionActivities"
slot of any of the top-product.
```
Write model to output file:
```
instance of CM_Model:          ReadSTEPpf! (<filename>, 1, 1, 0, 0);
instance of CM_Model:          WriteSTEPpf! (<filename>, 0, 1);
```

6 SA: Schedule Activities

6.1 Goal

Schedule the activities resulting from the planning phase.

6.2 Knowledge implemented

- Calculation of the ExpectedDuration of the activities. The default Kappa method, CalculateDurationOfActivity!, of a ConstructionMethod bases the duration on the number of units to produce (e.g., the volume), the number of resource units available, and the productivity of the resource. The Kappa method calculates the duration of all resources used in a ConstructionMethod, and sets the duration of the activity the longest duration. (As a consequence, all resource re allocated to that method for the whole duration of the activity, even if the calculated duration for a single resource is shorter.) When more specialized calculation of the duration is required for a class of ConstructionMethods, the Kappa method CalculateDurationOfActivity! should be redefined.
- Calculation of the delay[1] of the SequenceRelations. In case of a SucceededBy relation, the Finish-Start delay of the connected Activities equals the DelayForNextAct of the first ConstructionMethod + DelayForNextActIfSupports of the first ConstructionMethod if the ReasonForExistence of the SucceededBy relation is a SupportedBy relation between the corresponding part-Products. In case of a PartiallyInParallelWith relation, the Start-Start delay is calculated by multiplying the ExpectedDuration of the first Activity with StartStartDelayInPercentageOfDuration of the relation.
- Derivation of the secondary SequenceRelations. Primary SequenceRelations between Activities are translated to Secondary SequenceRelations between their part-Activities. In case of a SucceededBy relation, the last part-Activities of the first Activity are connected to the first part-Activities of the second Activity with secondary SucceededBy relations (see figure 8). In case of an ActivityFlow relation, 'corresponding' part-Activities are connected with secondary SucceededBy relations (see figure 9). In the current implementation, corresponding is determined by comparing the types of the realized Products of the Activities.
  The only difference between primary and secondary SequenceRelations is their ReasonForExistence: primary relations exist because of TopologicalRelation or

[1] On second thought the word "float" would be better.

ConstructionMethods, secondary relations exist because of other SequenceRelations. This property is used in the Kappa method RemoveAllSecondaryRelations!, which can be used when you want to recalculate the secondary relations. It also means that even secondary SequenceRelations are translated to lower level SequenceRelations.

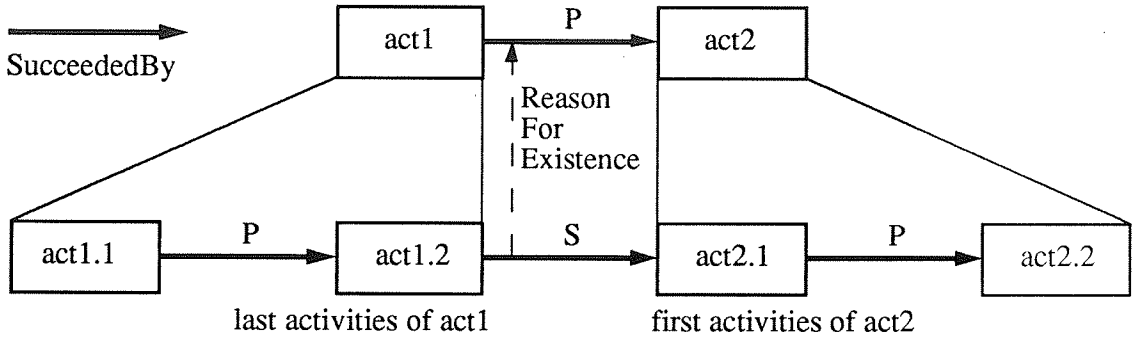

Figure 8 *The primary (P) SucceededBy relation between act1 and act2 is translated to secondary (S) SucceededBy relations between the last part-Activities of act1 and the first part-Activities of act2.*

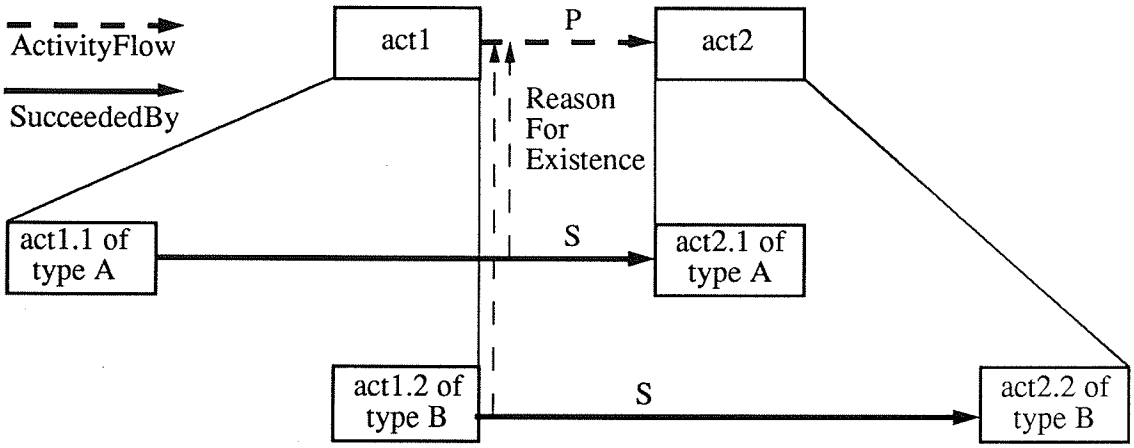

Figure 9 *The primary (P) ActivityFlow relation between act1 and act2 is translated to secondary (S) SucceededBy relations between corresponding part-Activities of act1 and act2.*

- Calculation of the critical path of the leaf activities, using the CPM forward and backward pass. This determines the early start, early finish, late start, and late finish of each leaf activity, without taking the limited availability of the selected resources into account.
- Rescheduling of the leaf activities in such a way that the availability of the selected Resources is not exceeded. This sets the scheduled start and scheduled finish of each leaf activity. Rescheduling consists of two steps:
  1 find the most critical resource (MCR). A resource is critical when at a certain point it is used more often than it is available. The MCR is the resource for which the total relative criticality (i.e. Sum {the period a resource is critical * quantity of resource used / quantity of resource available}) is highest. Rescheduling based on the MCR has *probably* the most effect on the schedule. (Kappa method: SetMostCriticalResource! which sets the MostCriticalResource slot of a project.)
  2 reschedule in such a way that the MCR is not critical anymore. As long as the MCR is not critical, nothing changes. At each point on which the MCR is critical, first all activities that do not use the MCR are left unchanged. Then, the those activities

with the highest priority and that use less MCR than available are left unchanged. All other activities that were supposed to start at that point in the schedule are deferred to the first point in the schedule when MCR comes available again. (Kappa method: ScheduleLeafActsBasedOnMCR!)
In the current version, the priority of an activity is determined by calculating the float relative to the duration of the project, using the scheduled finish and the late finish. An extension might be to incorporate other reasons for prioritizing an activity, such as the amount of MCR used. (Kappa method: PriorityOfAct (?act, ?ProjectDuration))

These two steps are repeated until no more critical resource is left. Rescheduling based on the MCR first *most likely* minimizes the number of iterations.

6.3 Manual

Actions in Kappa:

Initialize:

```
Set (or check) TopProduct TargetStartDate and TargetRealizationDate.
instance of CO_Project:        FindTopProduct! ();
instance of CO_Project:        AddStartFinishActs! ();
instance of CO_Project:        SetSecondarySequenceRelations! ();
instance of CO_Project:        FindLeafActs! ();
instance of CO_Project:        FindLeafPredsOfLeafActs! ();
instance of CO_Project:        ScheduleAllActsToProjectStart! ();
```

Schedule of leaf activities:

```
instance of CO_Project:        CalculateDurationOfLeafActs! ();
instance of CO_Project:        CalculateDelaysOfSequenceRelations! ();
instance of CO_Project:        ScheduleLeafActsBasedOnCPM! ();
instance of CO_Project:        ScheduleLeafActsToEarlyDates! ();
instance of CO_Project:        ScheduleLeafActsBasedOnResources! ();
```

Or step-by-step:

```
instance of CO_Project:        SetMostCriticalResource! ();
instance of CO_Project:        ShowResourceGraphs! ();
instance of CO_Project:        ScheduleLeafActsBasedOnMCR! ();
instance of CO_Project:        SetMostCriticalResource! ();
instance of CO_Project:        ShowResourceGraphs! ();
etc.
```

Display the criticality of all resources graphically (can only be used after SetMostCriticalResource! is executed, e.g., as part of ScheduleLeafActsBasedOnResources!):

```
instance of CO_Project:        ShowResourceGraphs! ();
```

Aggregate leaf activities to higher level activities:

```
instance of CO_Project:        AggregateLeafActs! ();
```

Write model to output file:

```
instance of CM_Model:          WriteSTEPpf! (<filename>, 0, 1);
```

7 SC: Simulate Construction process

7.1 Goal

Generate the input for a graphical simulation of the construction schedule.

7.2 Knowledge implemented

Generates the input for a graphical simulation of the construction schedule by identifying which products are when under construction or realized.

7.3 Manual

Actions in Kappa:

Read output from am module:

```
instance of CM_Model:          ReadSTEPpf! (<filename>, 0, 0, 1, 1);
```

Initialize:

```
instance of CO_Project:        Initialize! ();
```

Generate simulation input:

```
instance of CO_Project:        GenerateGraphicalSimulationInput! (?type);
                               ?type = "Scheduled", "Early", "Late", or
                               "Actual"
```

Export simulation input to AutoLISP readable file (NB the file name is in lower case character because LISP does not see the difference):

```
instance of CO_Project:        ExportSimulationDataToLISP! (?type);
                               ?type = "Scheduled", "Early", "Late", or
                               "Actual"
```

Or generate simulation input for all types and export to AutoLISP readable files in once:

```
instance of CM_Model:          GenerateAllLISPSimulationData! ();
```

Write model to output file:

```
instance of CM_Model:          WriteSTEPpf! (<filename>, 0, 1);
```

Actions in AutoCAD:

Load 3D drawing:

```
acad <filename> &
```

Load simulation (or put this in your acad.lsp file):

```
(load "simulation")
```

Ask for input file, load it and start simulation dialog box:

```
(LoadSimulation)
```

Startup simulation dialog box (see figure 10):

```
(ShowSimulationDB)
and the rest speaks for itself
```

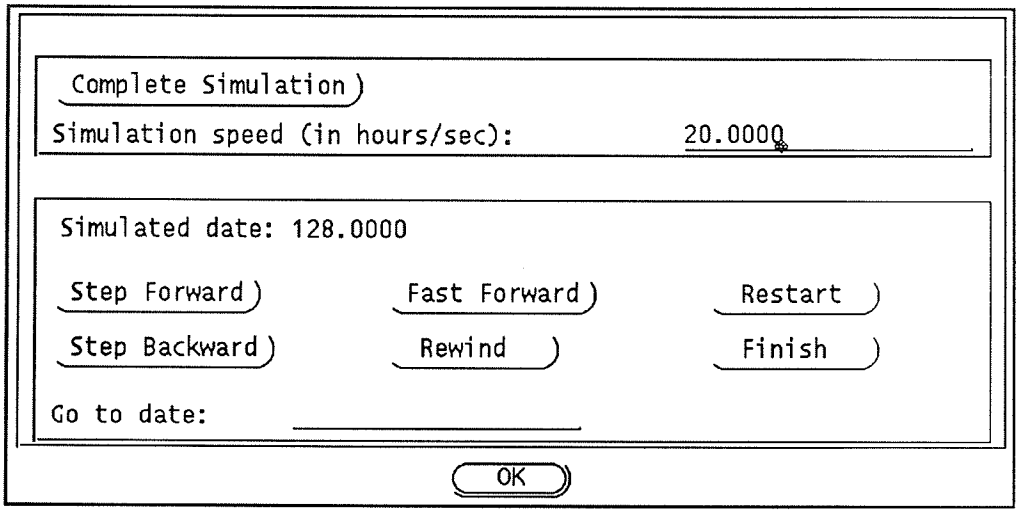

Figure 10 Simulation dialog box in AutoCAD

8 EC: Estimate Construction cost

8.1 Goal

8.2 Knowledge implemented

8.3 Manual

9 COMB: Combine modules

9.1 Goal

Combine the functionality of the modules.

9.2 Knowledge implemented

No new knowledge implemented.

9.3 Manual

Use multiple inheritance to combine the functionality. Basically, comb contains the same classes as cmim, that all inherit from the corresponding cmim class. In case a module redefined one of cmim classes, the corresponding comb class also inherits from that redefined class in the module.

10 CS: Concrete Structures

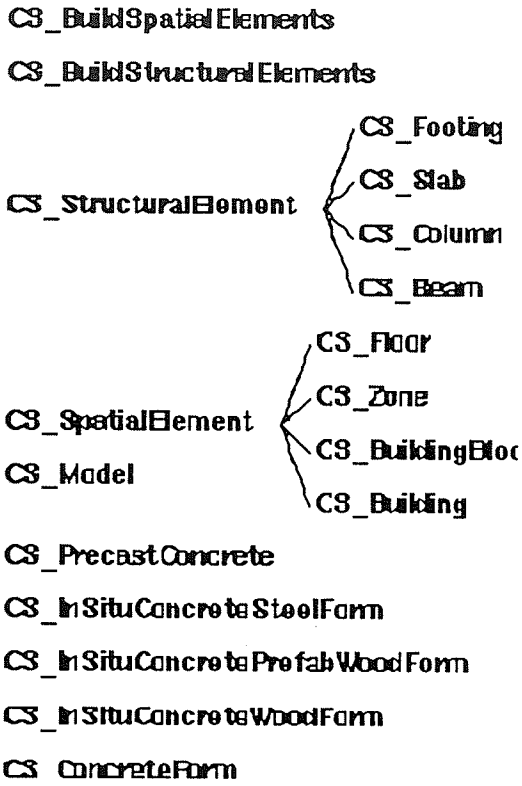

Figure 11 Class hierarchy in concrete structures (cs) module for Example 2.

10.1 Goal

Define information structure for concrete structures as a specialization of the generic cmim. This model is an example that shows how the cmim model can be specialized for a specific type of construction projects.

10.2 Knowledge implemented

Information structure (see figure 11) + ConstructionMethod selection knowledge (redefinition of Kappa method CanFulfillActivity!).

Spatial elements:

The selection of a ConstructionMethod for spatial elements is very simple and does not imply much reasoning. There are two alternatives: RealizeSpatialElements, in case the product involved decomposes into spatial elements, and RealizeStructuralElements, in case the product involved decomposes into structural elements.

Decomposition of the ConstructionMethods into lower level activities uses the default decomposition following the parts of the product.

Structural elements:

The selection of a ConstructionMethod for structural elements first focuses on the choice between precast or in-situ concrete, and then, for in-situ concrete, on the choice of formwork. The selection is based on the types of resource that are available. The default list of possible ConstructionMethods in order of preference is: PrecastConcrete, InSituConcreteSteelForms, InSituConcretePrefabWoodForm, InSituConcreteWoodForm. Note that these choices are made on the element level, which implies that they are (yet ??) independent of choices for other elements.

Decomposition of the ConstructionMethods into lower level activities uses the default decomposition following the parts of the product. Because the structural elements are not decomposed any further, the ConstructionMethods are not decomposed in lower level activities.

10.3 Manual

A very simple structure is modeled to demonstrate the whole system and the concrete structures module (see figure 12). This simple test structure is taken through the five module of the system and thereby demonstrates most aspects of it.

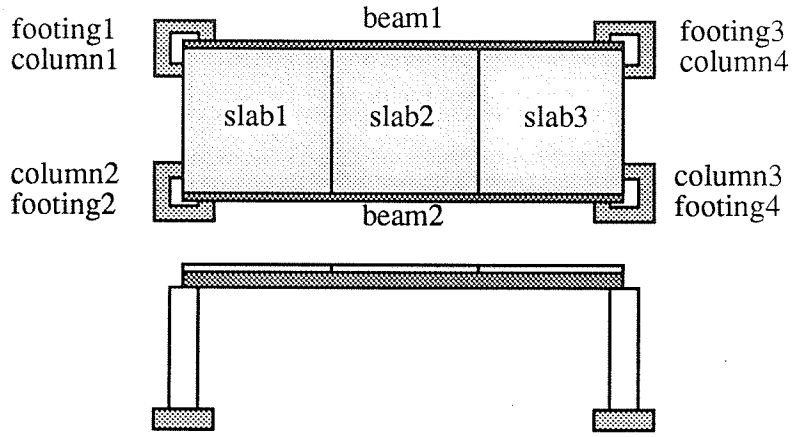

Figure 12 *Simple test case to demonstrate system.*

Example 2: Interpret Building

Actions in AutoCAD-SME+:

Load 3D drawing in /home/users/luiten/SME directory (see figure 13):

```
acad example2.dwg &
```

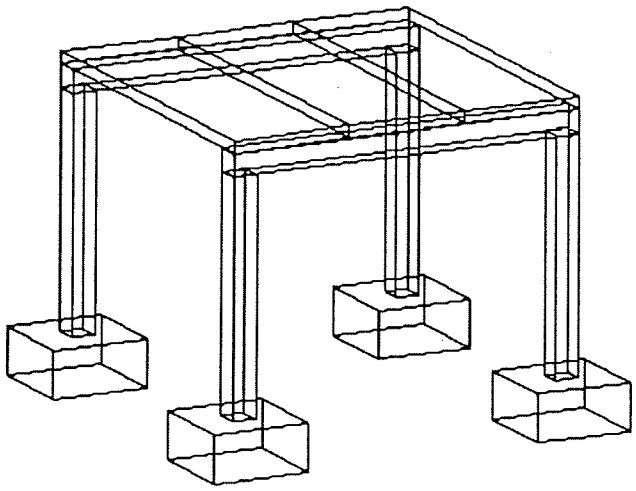

Figure 13 *Original 3D AutoCAD model of Example2 as it results from design.*

Load SME+:

```
(load "edsm")
```

Load model:

```
Semantics.Open: select example2
```

Add spatial objects (see figure 14):

```
Add floor1, building1 using Model.Primitives
```

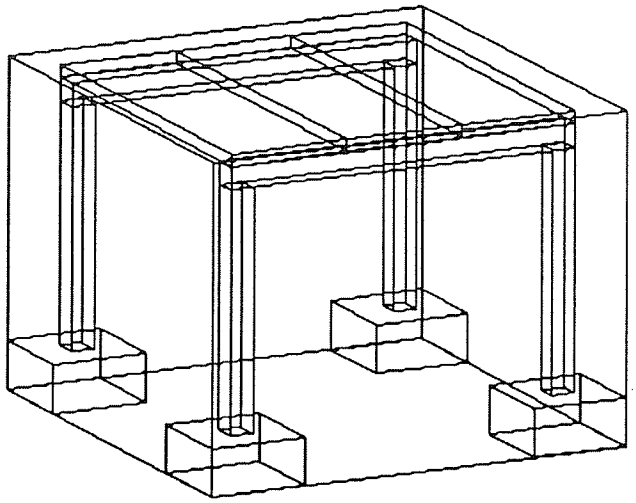

Figure 14 *3D AutoCAD-SME+ model of Example2 with spatial objects such floors and zones.*

Classify components and spatial objects:

```
For each class: Semantics.Preferences.DefaultFeature (to set the default
class) & Semantics.AddFeature (to classify the instances of that class)
```

Add not graphical information:

```
Semantics.Interpretation: set attribute values for instances
```

Derive geometrical information:

```
Semantics.Interpretation: Derive_Attributes
```

Write information to STEP file:

```
(DSMWriteSTEPpf)
```

Example 2: Generate Activity-Method model

Actions in Kappa:

Read output from ib module (see figure 15):

```
cs1_model.ReadSTEPpf!
("cs/DesignEx2.spf", 1, 1, 1, 1);
```

Read resource information:

```
cs1_model.ReadSTEPpf!
("cs/resources.spf", 1, 0, 1, 0);
```

Initialize:

```
create instance of CO_Project
and name it Exmaple2;
Example2.FindTopProduct! ();
Example2.AddActivityToTopProduct!
();
```

Define activity-method tree recursively starting from highest level activity:

```
Example2.ElaborateTopActivity! ();
```

Inspect results (see figure 16):

```
Example2.FindPredecessorsOfAllActs
! ();
Invoke a "Slot Graph" to view the
"Predecessors" (or "Successors")
slot of any of the instances ConstructionActivity.
```

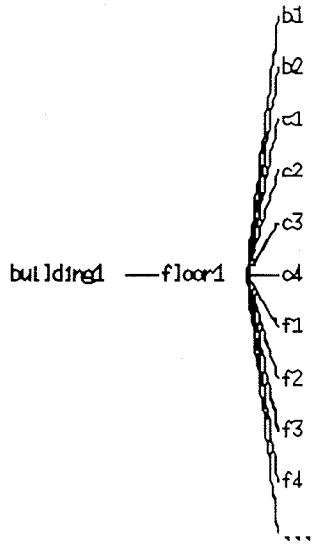

Figure 15 *Product decomposition hierarchy as defined in AutoCAD-SME+.*

```
Example2.FindPartActsOfAllActs! ();
Invoke a "Slot Graph" to view the "DecomposesIntoConstructionActivities"
slot of any of realize_building1.
```

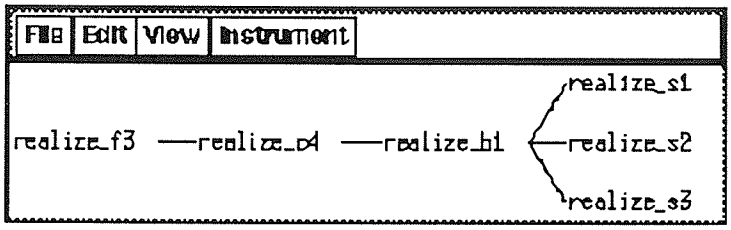

Figure 16 *Some ConstructionActivities of Example2 with their Successor relations generated in the am module.*

Write model to output file:

```
cs1_model.ReadSTEPpf! ("cmim/cmimdef.spf", 1, 1, 0, 0);
cs1_model.WriteSTEPpf! ("am/amEx2.spf", 0, 1);
```

Example 2: Schedule Activities

Actions in Kappa:

Initialize:

```
Set TopProduct TargetStartDate and TargetRealizationDate to 0 and 400.
Example2.FindTopProduct! ();
Example2.AddStartFinishActs! ();
Example2.SetSecondarySequenceRelations! ();
Example2.FindLeafActs! ();
Example2.FindLeafPredsOfLeafActs! ();
Example2.ScheduleAllActsToProjectStart! ();
See figure 17 for the leaf activities and their successor relations.
```

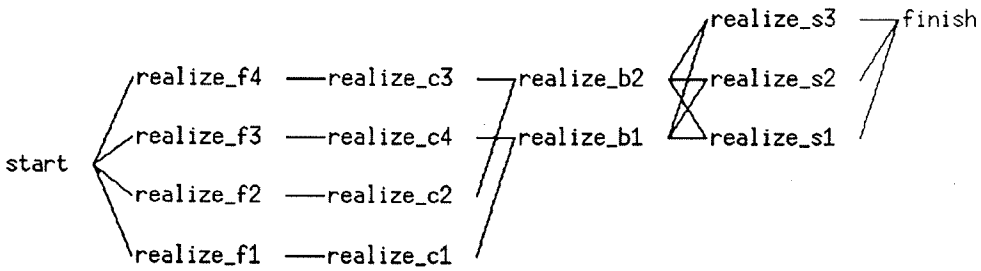

Figure 17 *The Leaf ConstructionActivities of Example2 with their Successor relations generated in the sa module.*

Schedule of leaf activities:

```
Example2.CalculateDurationOfLeafActs! ();
Example2.CalculateDelaysOfSequenceRelations! ();
Example2.ScheduleLeafActsBasedOnCPM! ();
Example2.ScheduleLeafActsToEarlyDates! ();
Example2.ScheduleLeafActsBasedOnResources! ();
Or step-by-step:
  Example2.SetMostCriticalResource! ();
  Example2.ShowResourceGraphs! ();
  Example2.ScheduleLeafActsBasedOnMCR! ();
  Example2.SetMostCriticalResource! ();
```

```
    Example2.ShowResourceGraphs! ();
    etc.
```
This results in RealizationDate = 449, which is larger than the TargetRealizationDate. As shown in the step-by-step calculation, ConcreteWorkCrew is the first MCR, and thus the limiting factor. When its MaxUnitsAvailable is changed from 1 to 2, and the scheduling is done again, the RealizationDate = 226, which is smaller than the TargetRealizationDate.

Aggregate leaf activities to higher level activities:
```
Example2.AggregateLeafActs! ();
```
Write model to output file:
```
csl_model.WriteSTEPpf! ("sa/saEx2.spf", 0, 1);
```

Example 2: Simulate Construction process

Actions in Kappa:

Initialize:
```
Example2.Initialize! ();
```
Generate simulation input for all types and export to AutoLISP readable files:
```
Example2.GenerateAllLISPSimulationData! ();
```
Write model to STEP file:
```
csl_model.WriteSTEPpf! ("sc/scEx2.spf", 0, 1);
```

Actions in AutoCAD:

Load 3D drawing (in /home/users/luiten/SME directory):
```
acad example2.dwg &
```
Ask for input file, load it and start simulation dialog box:
```
(LoadSimulation)
```
Startup simulation dialog box (see figure 10):
```
(ShowSimulationDB)
and the rest speaks for itself (result see figure 18)
```

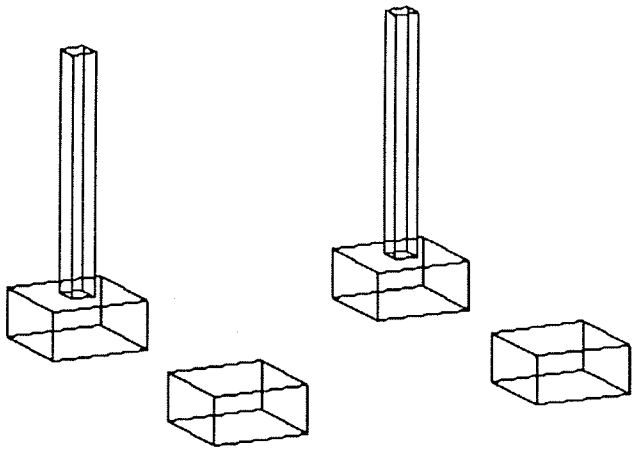

Figure 18 Simulation of the construction process of Example2 in AutoCAD-SME+.

11 AC: Apartment Complexes

11.1 Goal

The model for apartment complexes shows the use of planning by defining Flow relations between Products and Activities.

11.2 Knowledge implemented

See figure 19

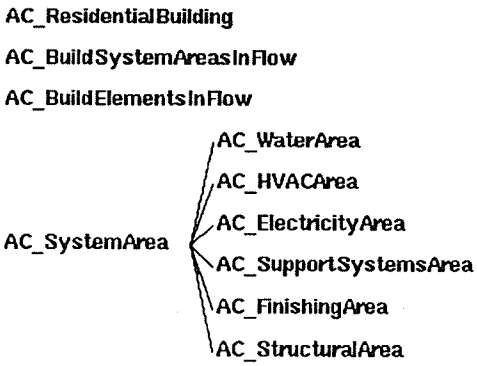

Figure 19 Class hierarchy in apartment complexes (ac) module.

11.3 Manual

See figure 20, 21, and 22.

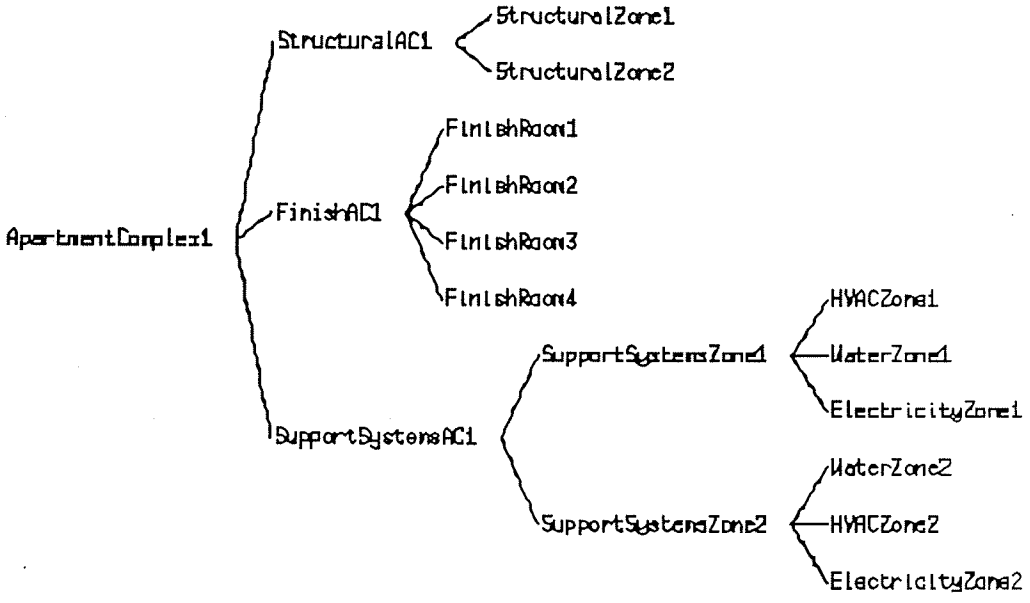

Figure 20 Product decomposition structure in ac1.

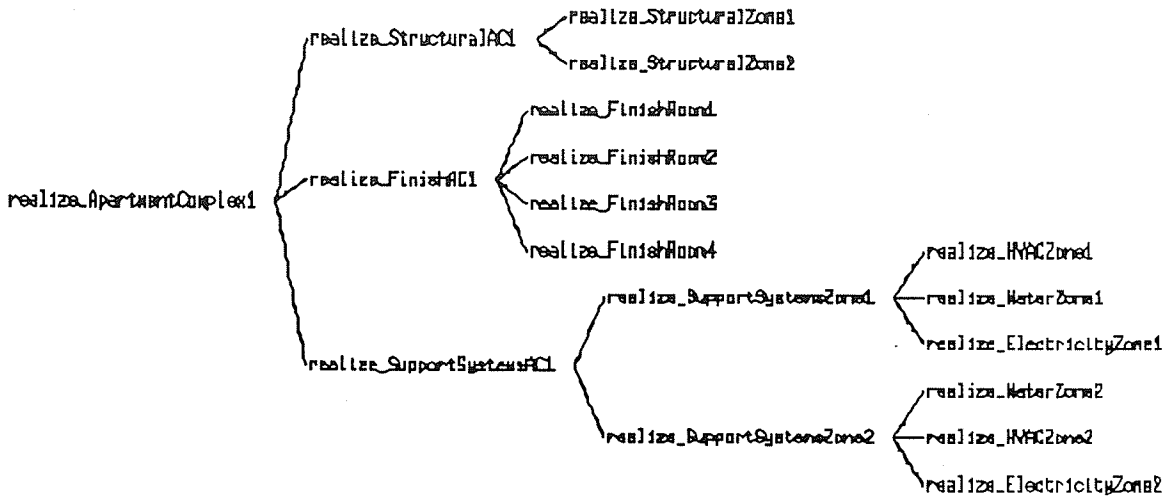

Figure 21 *Activity decomposition structure in ac1.*

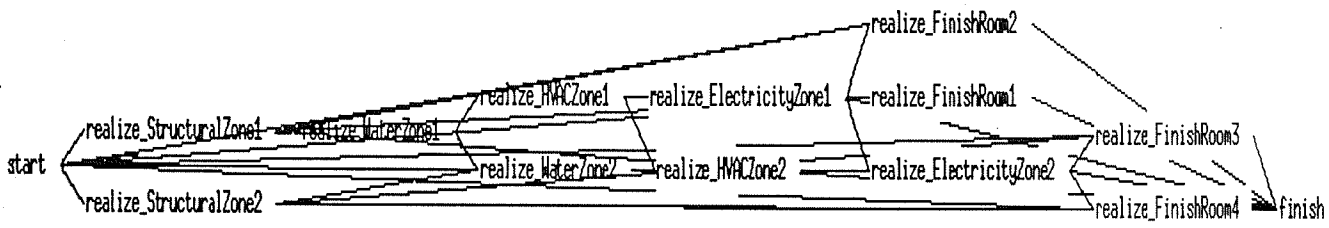

Figure 22 *Sequence relations between leaf Activities in act1.*

12 Start your own module

```
Create new app in XX directory, where XX is the name of the new app and
the new prefix.
Save and unload.
Copy ~luiten/pk.new/cs/cs.app file to new_directory/XX.app and change
every "cs" for "XX".
Load.
Create an instance of CO_Model and name it XX_model.
Change Prefix in XX_Model to "XX_".
```

and you are ready to start your module.

13 Update the construction management information model (cmim)

Talk to Bart (bring coffee and something nice to eat).

14 Future activities

- Develop user-interface.
- Implement SME+ in object-oriented CAD environment.

- Experiment with using this framework for design management and overall project management.

- Experiment with decomposition of Products by a ConstructionMethod, e.g., as an implementation of a value-added evaluation tool.
- Experiment with adding a decomposition layer by a ConstructionMethod, e.g., as an implementation of zoning and work packaging.
- Make ConstructionMethod knowledge easy to implement for users in practice.
- Extend planning and scheduling with reasoning about the workspace.

- Link scheduling module with organizational modeling tools, e.g., with VDT.
- In stead of scheduling in the scheduling (sa) module, you could also implement a two-way link with a commercial scheduling application, e.g., with PrimaVera.

- Extend construction simulation with resources, and resource movements, product movements, and interference checking.
- In stead of visualizing the construction process in AutoCAD, you could also implement a link with a commercial visualization application, e.g., with Walkthru.

- Implement cost estimating module.
- In stead of estimating the cost in the cost estimating (ec) module, you could also implement a two-way link with a commercial cost estimating application, e.g., with Timberline.

- Test examples of real life size and with real life numbers.

Appendix A Work with Kappa

A.1 Compile part of the system

It is possible to compile one finished model or module and still use the interpreter for the models or modules that are under development. This speeds up the methods of the compiled model or module considerately. Do the following:

1 finish the app (model or module) you want to compile completely;
2 for all .pkc files in the app, type in the C-listener:
    proto <path.filename.pkc>;
    when asked for an output file: <path.filename.pkc.proto>;
3 compile the app by selecting Build Runtime from the App menu;
(BTW you only need the .pto and .pko files; if you get an error message but you have those files, it is no problem; you can remove the other files that are created.)
4 in .app file of the app you compiled:
a change all .ptk (in ProTalkFiles) and .pkc (in CFiles) files to .pto and .pko;
b add all <:filename.pkc.proto> files to Libraries;
5 reload app that use the compiled app.

NB1 step 2 and 4b are only necessary if you programmed in C.
NB2 In the compiled version you do not see the names of the arguments anymore. If you find a remedy, please, let me know.

A.2 Display graphs

1 Use the Images app of John Kunz (maybe a compiled version is faster!)
2 Use xgraph (see code in luiten/pk.new/cmim/cmim.ptk to display a resource graph)

A.3 Use SlotGraph probe to display data structures

See figures 15, 16, and 17 for examples.

A.4 Use inverse relations

This works very good and keeps your data consistent. When using inverse relations, you only need to save the relation in one direction, which saves reading and writing time.

A.5 Use .app file

When you use the .app file for saving the resource set of an application, it is easier to change the resource set manually than automatically by using the .krs file. This is especially convenient when you want to compile a model or a module.

Automated and Integrated Management of Scope, Budget, and Schedule

By Martin A. Fischer[1] and Gijsbertus T. (Bart) Luiten[2]

ABSTRACT: The building industry is facing increasing pressures to deliver facilities in shorter time, at reasonable cost, and with good quality. Construction management traditionally focuses on controlling the construction process, and computer tools have supported scheduling and cost estimating for decades. Planning and integration of product design, scheduling, and cost estimating are, however, not yet supported by computers in daily practice. In our ongoing research, we are developing an approach to integrate scope, budget, and schedule concerns and automate planning, scheduling and cost estimating. In this paper we first discuss limitations that hinder better computer support for project managers. The main limitation is the low level at which project information is represented. This means that integration of design and construction planning information and automated reasoning about, for example, planning are very difficult to implement. We propose and describe a higher level representation of project information that explicitly represents the relations between products, activities, construction methods, and resources. A prototype computer system shows that it is possible to implement our conceptual project model and support project management decisions.

INTRODUCTION

Management of scope, budget, and schedule are important project management tasks throughout the delivery of engineered facilities. Quality, safety, environment, maintainability, and operability are examples of other important project objectives. Effective project managers, together with other specialists, continuously monitor these project objectives by identifying and making tradeoffs. Such tradeoffs can be made through evaluation of design alternatives from as many viewpoints as desired and through comparisons of the achieved behavior of an alternative with its intended functions.

Experienced project managers recognize tradeoffs and balance project objectives based on their intuition and rules of thumb gained over many years of practice. While we don't envision the replacement of experienced project managers with software tools, we envision project management tools that support the integrated management of project objectives to a much higher degree than the tools available today. Such tools will allow rapid testing of as many alternatives as needed to align client requirements and proposed solutions. They will form a basis for individual and organizational learning on a project and from project to project. They will shorten the time needed to identify and fulfill project objectives or increase the degree to which these objectives will be achieved. They will shorten the time needed to become an expert project manager. They will alter the way in which architecture, engineering, and construction professionals work and communicate with each other.

Generally speaking, construction management software tools should consist of modeling, automation, integration, and visualization components (see Fig. 1). The modeling

[1] Asst. Prof., Dept. of Civ. Engrg., Stanford University, Stanford, CA 94305-4020; Ph.: 415 725 4649; Fax: 415 725 8662; E-mail: Fischer@CE.Stanford.EDU

[2] Postdoctoral Fellow, Dept. of Civ. Engrg., Stanford University, Stanford, CA 94305-4020, and Dept. of Civ. Engrg., Delft University of Technology, Delft, The Netherlands; Ph.: 415 723 9340; Fax: 415 725 8662; E-mail: Luiten@CE.Stanford.EDU

components should support the modeling of products (i.e., the facilities and their elements) and processes at various levels of detail throughout the project life cycle. The automation components should automate tasks for different phases and disciplines to provide rapid feedback on design alternatives. The integration component should ensure a seamless information flow between the users and the automated tasks, and the visualization components should display results to the users as graphically as possible.

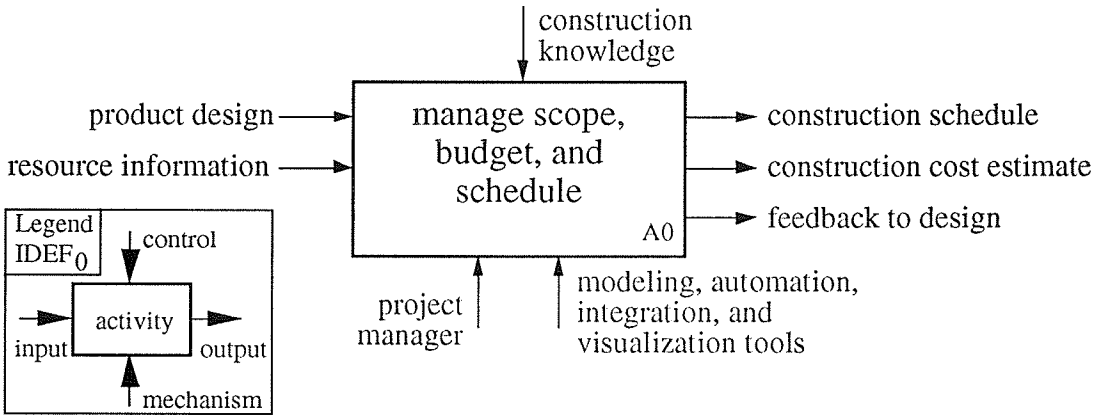

Fig. 1. Project management modeled in IDEF$_0$ (SofTech 1981).

Fig. 2 shows construction management tasks and information flows as observed in current practice. Project managers start with estimating the cost of constructing a facility, often with the support of an estimating tool. When their firm is awarded the project, they manually define and plan the construction activities, which they schedule with scheduling tools. Given the effort it takes to develop a project schedule today, it is not surprising that firms only develop detailed schedules once they have been awarded a project. Thus, a cost estimate is developed first based on unit costs, and a schedule is developed much later based on available resources, production rates, project objectives, climate and other environmental conditions, etc. Often, no explicit link is established between a project's budget and schedule even though concepts, such as work packaging, support both cost estimating and scheduling. Feedback on cost and schedule implications of design decisions frequently arrives only after design has been completed and the design budget expended. The two main problems that hinder a better and faster generation of estimates and schedules are (1) the manual planning process and (2) the paper-based communication with design and between construction management tasks. Other problems are the lack of graphical visualization and lack of formalized construction knowledge. As a consequence of these problems, there is generally little time for what-if analyses and optimization of product and process design.

We have developed an approach that offers a solution to these project management problems. Based on this approach, we implemented a prototype software system that supports integrated management of scope, budget, and schedule for a building project. Our research approach has two main objectives:

- to integrate project management concepts to ensure a consistent representation and flow of information among project management tasks and tools, and
- to formalize project management knowledge to support automation of planning, scheduling, and cost estimating tasks.

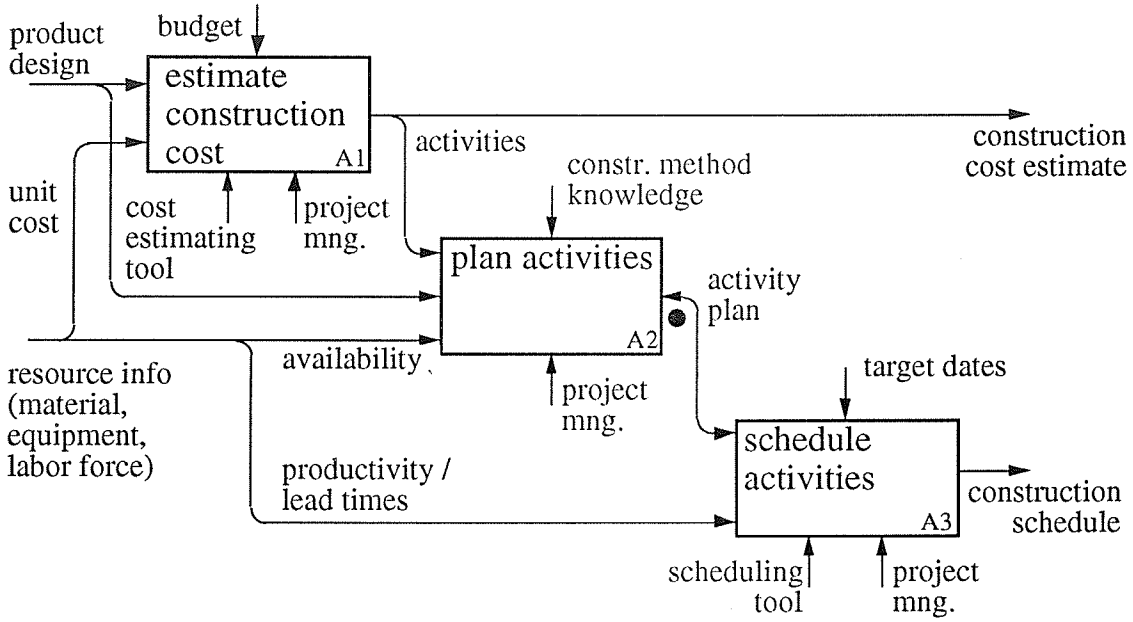

Fig. 2. Current project management tasks, mainly with paper-based communication and manual construction planning.

In this paper, we describe our research approach. We first discuss limitations of current project management approaches and present our basis for integration: a neutral representation of project management information using project modeling techniques. We then show how this neutral representation supports the reasoning necessary to automate the generation of plans, schedules, and estimates. We demonstrate the functionality of the prototype with a small example. We conclude with a summary of theoretical, educational, research, and practical challenges for the development and adoption of automated and integrated project management tools.

INTEGRATION: NEUTRAL PROJECT REPRESENTATION

In this section we first describe the limitations of current representations of project information and then introduce our neutral conceptual project model.

Limitations of current approaches to representing project information

Stuckenbruck (1981) defines project management as ".... a one-shot, time-limited, goal-directed, major undertaking, requiring the commitment of various skills and resources." The one-shot and time-limited aspects of project management have led to the development of general and flexible management concepts that can be applied to many different types of projects. However, these concepts do not support the goal-directed nature of project management and often lead to "reinventions of the wheel" because plans, schedules, and cost estimates are often developed from scratch for each project.

The following paragraphs discuss four fundamental limitations of state-of-the-art project management tools. These limitations are not limited to project management software. Rather they serve as examples of the general challenges for developing integrated software tools. With this paper we hope to make a contribution to these issues from the construction perspective.

1 Relations between different types of information are not modeled explicitly.

To integrate scope, time, and cost aspects of a project alternative, a project management system must *dynamically share semantic-rich* representations of the product, construction methods, and resources. Current representations, e.g., 2D CAD drawings, a STEP-based product model (ISO/TC184 1993a), or an activity network, only focus on one type of information and do not necessarily consider the relation with other types of information. For example, the relations between building components and scheduled activities typically only exist in the mind of the project manager. Some commercial project management tools now support file-based, static exchange of some scope, schedule, and cost information between applications (e.g., the integration of Timberline and Primavera software). To fully integrate and automate scope, budget, and schedule considerations, we need to explicitly represent the relations between different types of information. Cherneff et al. (1991) provide an early example of a system that supports the sharing of information between CAD systems and scheduling systems.

2 Scheduling is activity-based.

Birrell (1980) observed that: "In construction there are: (1) the resources required to execute the work; and (2) the end product to be constructed." However, all techniques commonly used to represent a construction plan and schedule (e.g., Gantt Chart, Critical Path Method (CPM), Line-of-balance chart) rely on the abstract notion of an *activity*. Birrell continues to argue that CPM is inadequate to plan the completion of a facility through a set of resources because it does not "consider each work squad as a continuous flow" which might lead to inefficient work plans. Nevertheless, CPM schedules have become the accepted technique to represent project processes. Line-of-balance charts show the relation between the product and the resources that construct it, but abstract this relationship to one number: the productivity of the resources. To allow the user to approach planning and scheduling from the traditional activity perspective—resources are needed because activities need them (Fondahl 1962)—or from a resource perspective—activities happen because resources perform them (Waugh 1990)—we need to complement the abstract notion of an activity with the explicit representation of *construction resources*.

Later in his paper, Birrell makes the interesting observation that "... any construction process is made up of a finite set of tasks from an existing feasible set of tasks carried out by the construction industry." Some researchers have attempted to classify actions performed by construction resources (Darwiche et al. 1989), but no generally accepted list exists. Instead of trying to predefine all the activities that will be performed on construction projects, we need to represent *construction methods* that allow easy customization of the descriptions of activities, resources, and components and their relationships (Navinchandra et al. 1988; Aouad and Price 1993; Jägbeck 1994).

3 Schedules are not easily extensible to different levels of decomposition and detail.

To be useful for construction management, a software system must support decomposition and aggregation of construction work by different categories (zones, contracts) and levels. This is known as the *flexibility* problem of the use of data models (Froese 1992). Decomposition in current tools is often static in nature and reasons for a particular breakdown are not made explicit.

Moreover, to be useful from conceptual to detailed design, a project management system must support the continuously growing amount of information. It must represent and reason about design, construction methods, and resources at various levels of decomposition and detail. It must support seamless and dynamic transition between these levels. This is known as the *extensibility* problem of data models (Phan 1993). Since current systems usually

support only one project phase, they are not useful in helping manage the changing levels of detail as a project progresses.

4 Reasons behind dependencies between activities are not represented.

CPM-based methods were developed because the Gantt Charts used over the first 60 years of this century did not represent a project's logic explicitly. Several researchers have discussed the limitations of the basic precedence relationships used in CPM: difficulty to specify certain time relationships and the lack of an explicit representation of the logic behind the job logic. Hendrickson et al. (1989) propose a unified activity network model that supports the specification of sixteen different time relationships between activities. Echeverry et al (1991) and Kähkönen (1993) summarize reasons for activity dependencies. Explicit representation of the *reasons for activity dependencies* enables computers to reason on a higher level. For example, only if it is known that the reason for a certain dependency is the limited availability of work crews, it is possible to remove the dependency automatically once crews have been added.

Conceptual project model

Much like typical CAD tools are not design tools, but merely represent a design graphically and facilitate the manipulation of abstract design primitives (i.e., lines, surfaces, solids), so are today's scheduling tools not planning tools, but simply allow (graphical) representation and manipulation of scheduling primitives (i.e., activities and their dependencies). If we are to integrate project management tasks, we have to raise the semantic level of the information represented and manipulated to the level of project models–much like the level of product information during design has to be raised to product models to integrate design tasks (Tolman 1991; Teicholz and Fischer 1994). Therefore, the core of our integration approach is a conceptual project model, i.e., a neutral representation of all information used during project management, including design, activity networks, schedules, resource plans, construction visualization information, and cost estimates. Neutral means that the representation of information is independent of the participants and applications that use the information.

The basic approach to project modeling is to define classes for sets of objects with similar characteristics. Characteristics are either relationships with other objects, such as the 'decomposes-into' relationships between structures and components, or attribute values, such as the shape and material properties of products. The object-oriented paradigm (Meyer 1988) adds the notion of methods to a class of objects. Methods derive relationships or calculate values. For example, a method for an activity can derive what construction methods are applicable, or a method can determine the weight of a component from its shape and material attributes. Several researchers have developed approaches for project modeling (Björk 1991; Luiten and Tolman 1991; Froese 1992; Gielingh and Suhm 1993; Luiten et al. 1993).

We apply this project modeling approach to construction management. Fig. 3 shows the class hierarchy we implemented in Kappa (IntelliCorp 1993), an object-oriented programming environment. In this diagram, classes are connected with their superclasses on the left and subclasses on the right. A subclass is a subset of the objects of a superclass. This subclass inherits all characteristics and methods from its superclasses and adds new ones that are typical for that subset of objects. With this class hierarchy it is possible to classify construction management information, e.g., information about resources, activities, and products.

Integration of management tasks needs more than a neutral classification of information: it also requires a neutral representation of the relationships between the classes. Fig. 4 shows the main relationships between the classes from Fig. 3 in a NIAM diagram (Nijssen and Halpin 1989). In NIAM, a class is represented by a circle and a relationship by a box on a

line between classes. Class-inheritance relationships are represented by an arrow from a subclass to a superclass.

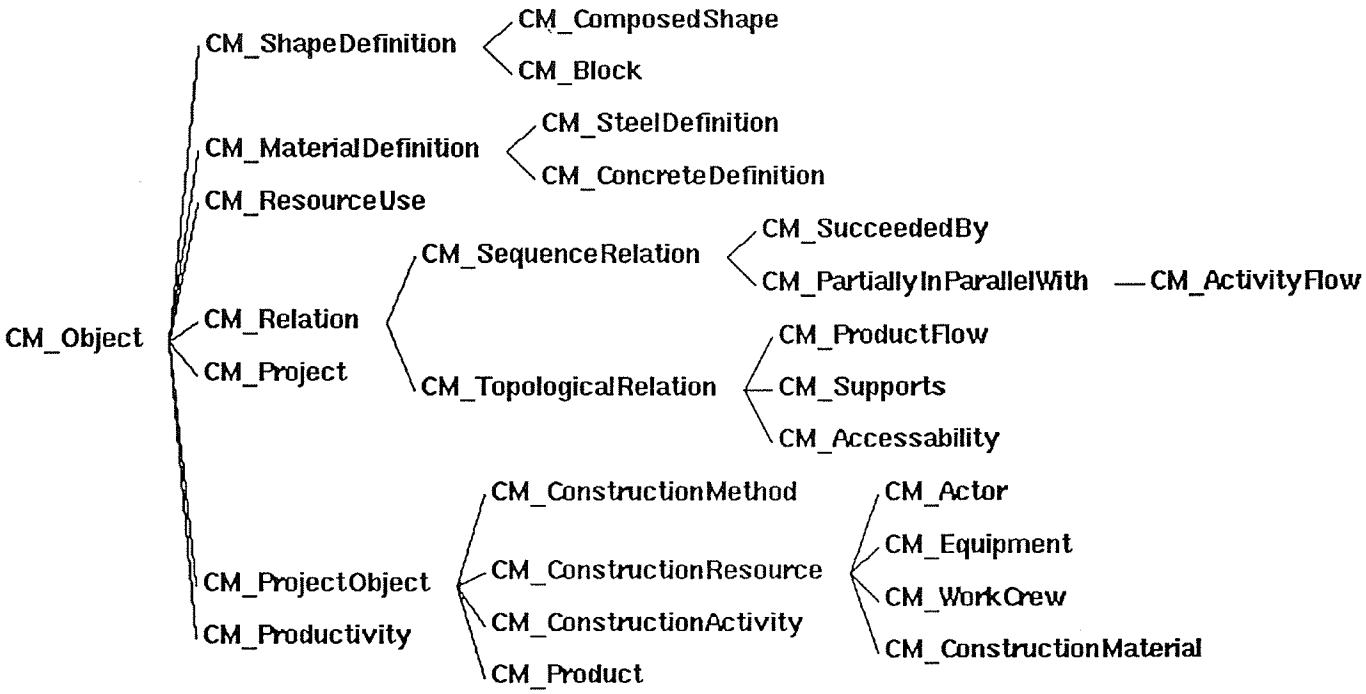

Fig. 3. Neutral representation of project information; class hierarchy as implemented in Kappa (IntelliCorp 1993).

The conceptual project model of Fig. 4 overcomes the limitations summarized above. In the figure, the classes and relationships that address these limitations are indicated with numbers corresponding to the limitations in the text. The model represents and relates information on a high semantic level and thus explicitly relates products, activities, construction methods, and resources. This addresses limitation 1. For example, with the model the shape of a product can be generally defined as a block with length, width, and height. From this general definition not only a 3D graphic can be derived for visualization, but also the volume and the surface area can be computed for cost estimation. In addition, the model represents resources and construction methods, thus overcoming limitation 2. With respect to limitation 3, the relationships between activities and construction methods allow flexible reasoning about the schedule and cost estimate at different levels of detail. The model also allows extension of a specific project model with more details in the course of a project. To resolve limitation 4, the model explicitly represents the reasons behind dependencies between products and between activities.

The relationship between activities and construction methods needs some elaboration. Product models often distinguish between function, form, and behavior of a product (Clayton et al. 1995). For example, in the FU-TS decomposition as proposed by Gielingh (1988), a functional requirement of a product is modeled in a functional unit (FU) and the shape and material definitions in a technical solution (TS). The technical solution contains knowledge that evaluates whether the predicted behavior of a solution corresponds to the required functionality. A similar construct is used in our model. Activities are modeled as functions that have to be performed and construction methods as solutions for these functions. For

example, the activity 'build-wall' with the requirement 'optimal duration and cost' can be fulfilled by the construction method 'mason-on-site' and its allocated resources. As in FU-TS decomposition, a construction method (a solution) decomposes into activities (requirements) at a lower level.

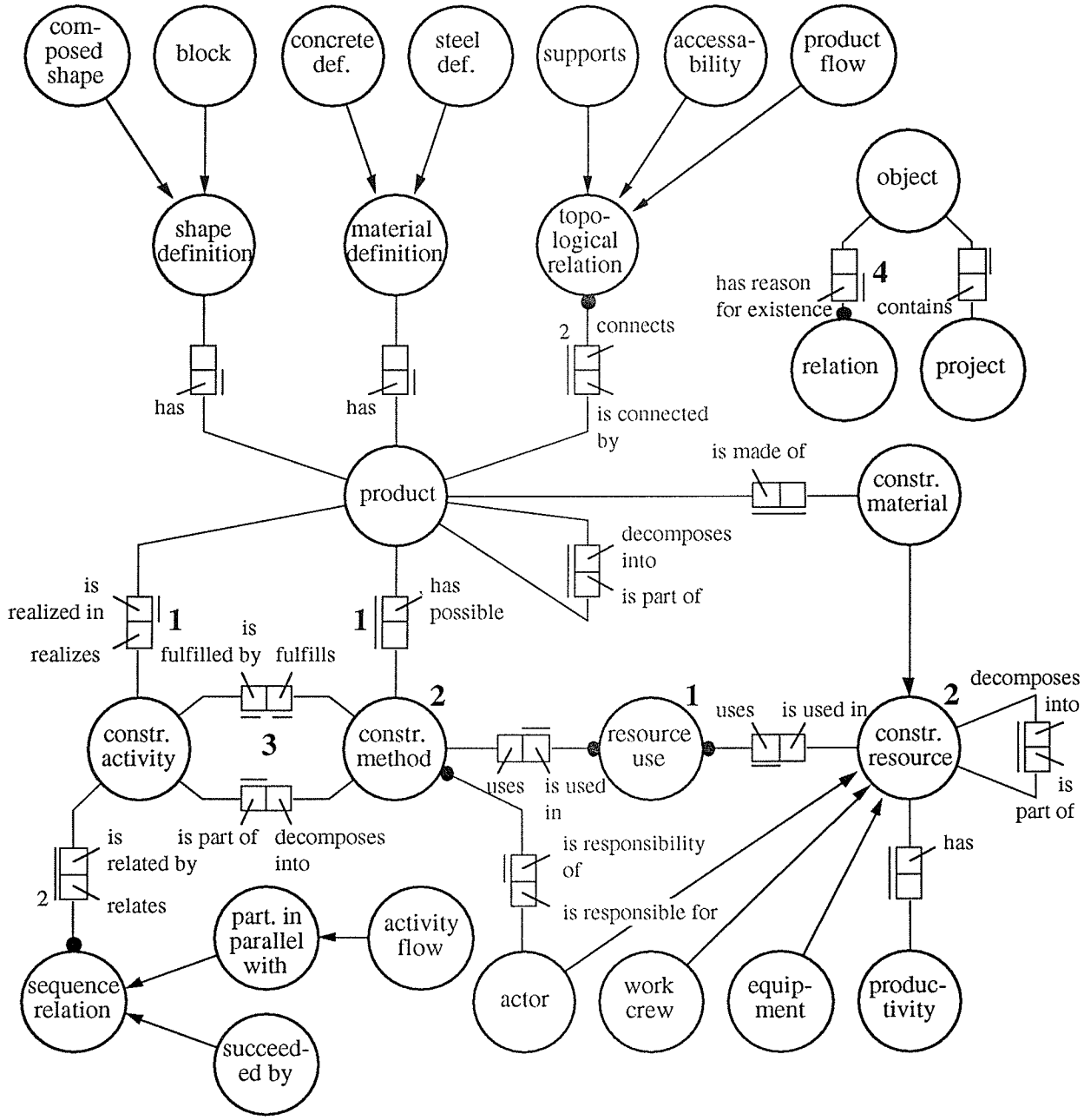

Fig. 4. Neutral representation of project information; classes and their relations modeled in NIAM (Nijssen and Halpin 1989).

The project classes *and* their attributes and relationships form the conceptual project model. This conceptual model is very abstract because it is intended to be valid for all

projects. For use on a particular type of project, e.g., concrete structures, this abstract model can be specialized to a project type model (PtM) (see Fig. 5). A PtM defines subclasses of the conceptual project model classes that have characteristics and knowledge valid for that type of projects (Tolman 1991). For a specific project, e.g., realizing the concrete structure for a new building of the School of Engineering at Stanford, the PtM is instantiated to a specific project model that contains information for that project in a neutral way. This means that objects are classified and related to each other in such a way that project management applications can derive information from the model and add information.

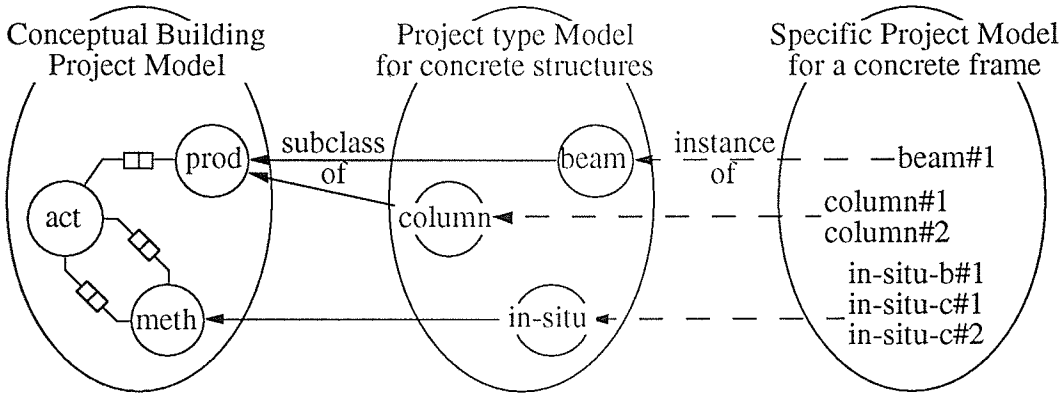

Fig. 5. Relations between the neutral conceptual project model, project type models, and specific project models.

AUTOMATION: REASONING ABOUT PROJECT INFORMATION

The neutral representation of project information is used in the prototype implementation of our system to automate reasoning about project management issues. In this section we discuss the architecture of our system and explain and illustrate the reasoning for each management task.

Overall system architecture

Fig. 6 shows five important construction management tasks and the information flows supported with our computer system. The construction manager first has to interpret the design to form a mental image of the building. With this image he/she chooses construction methods, activities, and resources. Once these activities are scheduled, the construction process can be simulated and visualized. With the activities, allocated resources, and components the construction cost can be calculated. In our computer environment, the first task, interpreting the design, is supported with SME+ (Clayton et al. 1994), an extension of AutoCAD. The other tasks are supported with modules developed in Kappa. We use AutoCAD to visualize the simulated construction process. The modules of this system will be explained in more detail in the upcoming sections.

Because several researchers are working on this system we chose to support each task with a separate module. Since we plan to automate more detailed tasks in the future, e.g., time-cost tradeoff, or might want to replace existing implementations, the system should be easily extensible. To ensure interoperability of these modules and the different systems, each module must be based on the conceptual model we discussed in the previous section. For these reasons, we chose a layered system architecture as shown in Fig. 7.

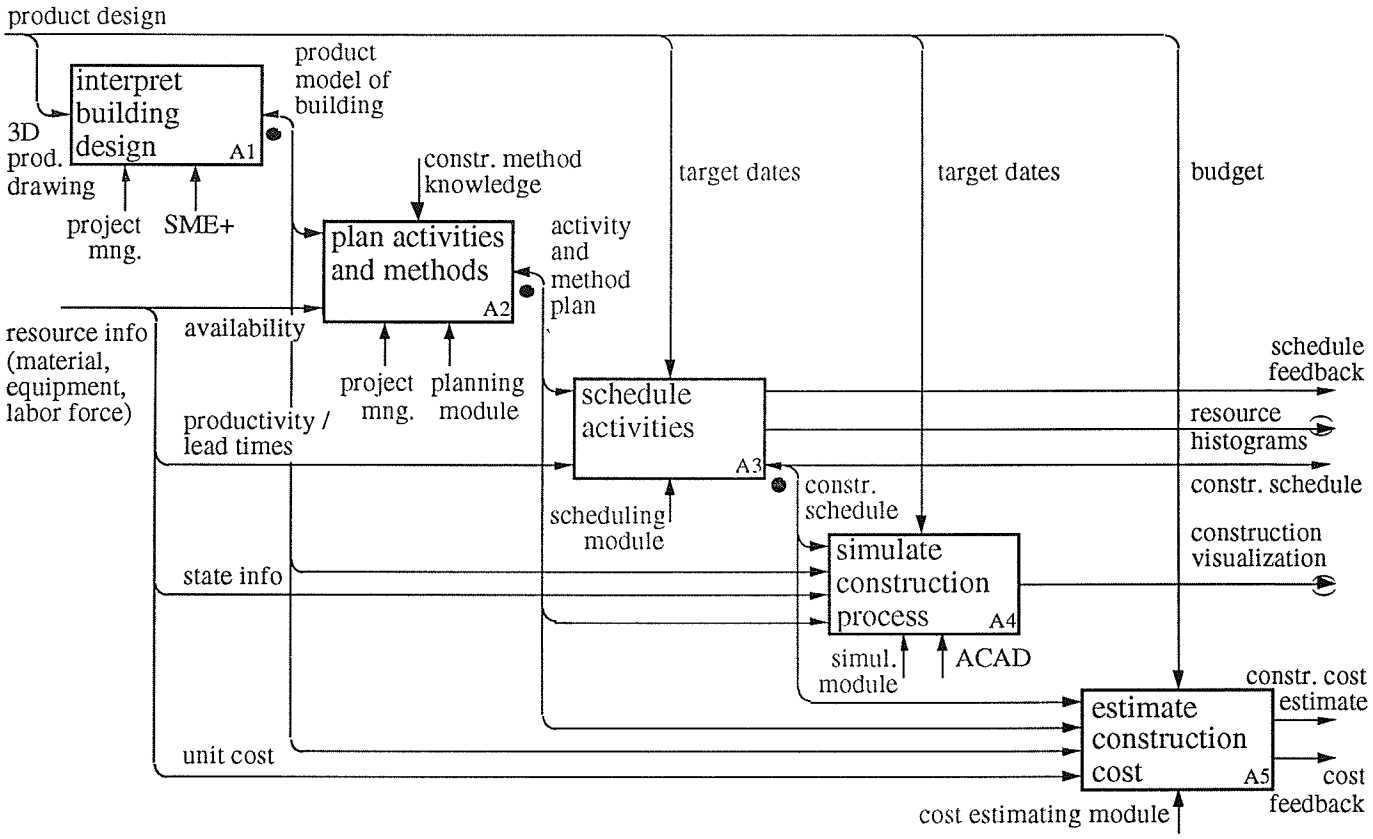

Fig. 6. Five tasks of construction management supported with SME+ (Clayton et al. 1994), the Scope, Budget, and Schedule Management System, and AutoCAD.

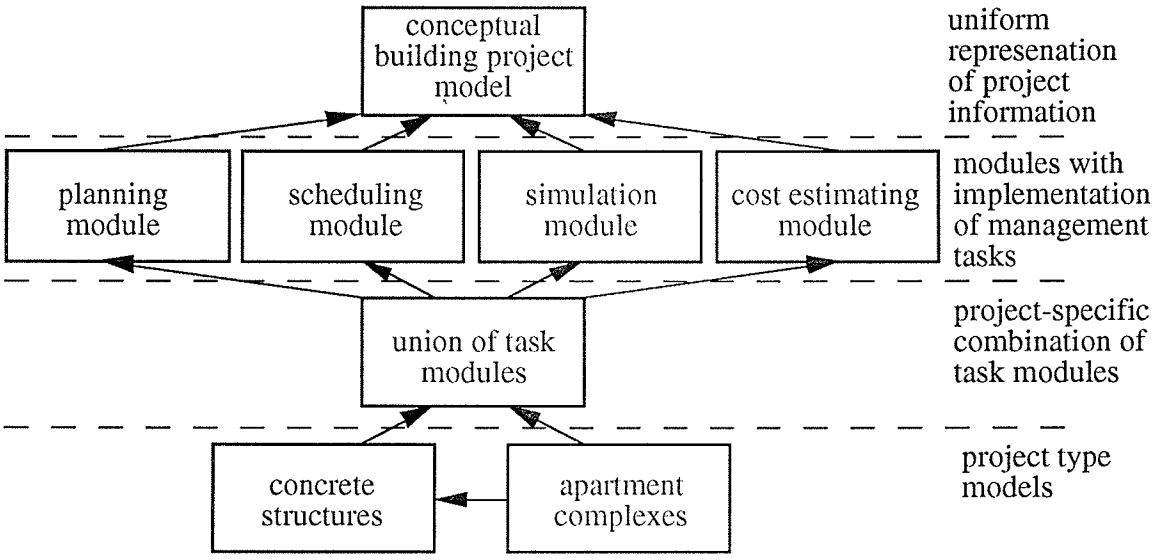

Fig. 7. Architecture of the layered Scope, Budget, and Schedule Management System.

Modules in a layer are always based on modules in layers above. This means that each class in a module is a subtype of classes of the layers above. The conceptual building project model is implemented in the top layer. This module also provides general functionality such as communication with other systems, e.g., SME+. The modules for the management tasks are implemented in the second layer. The third layer combines these modules into one system. It is this third layer that will provide the 'plug and play' capability needed for a system to be useful across multiple projects and organizations. The combination layer will make it easy for project participants to plug in their personal planning or scheduling module. The fourth layer contains the project type models for particular types of projects.

Reasoning in the modules for the main project management tasks

This section discusses the reasoning that is automated in each module and illustrates the reasoning for a simple test case.

Test case: a small concrete structure

To test our system we use a small concrete structure with thirteen elements (see Fig. 8). For this structure, we show how our system supports construction planning, scheduling, and visualization. The system provides an integrated environment that allows its users to address the following kinds of questions. What is the minimum project duration? What are the resource requirements for the minimum duration? How long would the project take with a reduced level of resources? How many resources are needed to complete the project in a given time? Today's project management systems do not support integrated analysis of these questions.

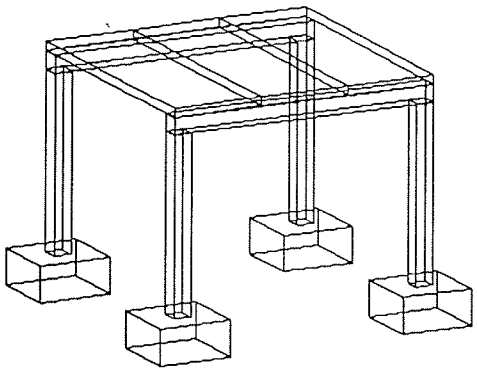

Fig. 8. Design of a simple concrete structure in 3D.

Interpret building design

To automate reasoning about the design, product information has to be represented in a product model that integrates with our conceptual building project model. In the future, designers will probably design with product models that can then be mapped to our conceptual model. Currently, however, designers only design in 2D or 3D drawings. Clayton et al. (1994) developed the Semantic Modeling Extension (SME), an extension of AutoCAD 12 that allows project managers to classify geometrical entities in a drawing into predefined component classes. We extended SME to SME+ with object-oriented properties, such as class inheritance and attributes. SME+ calculates geometrical attributes, such as volumes and surface areas, and derives relations between objects, e.g., 'decomposes-into' and 'is-supported-by'. The user can specify values for non-geometrical attributes, such as the material of a product. SME+ can export this product information to a STEP file (ISO/TC184 1993b) that can be read by other modules.

For interpretation of the design of the test case, the user adds two new entities to the drawing. These entities represent the aggregated structure for the whole building and the first floor. With the SME+ user interface (see Fig. 9), the user classifies the geometrical entities as building, floor, footing, column, beam, or slab objects and defines materials for the elements. SME+ then calculates geometric attributes and relations between objects and exports the resulting product model to a STEP file.

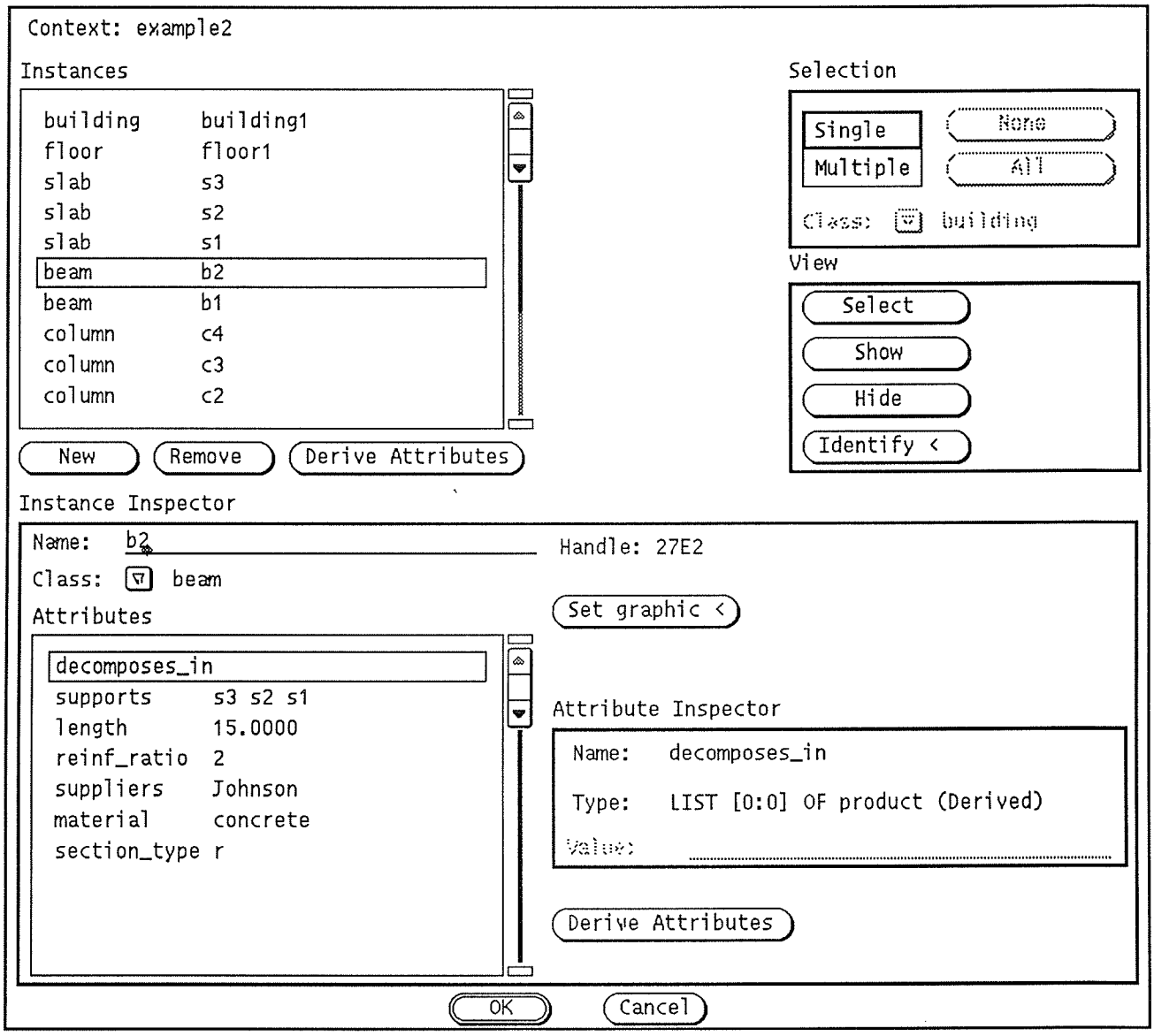

Fig. 9. SME+ user interface for classifying drawing entities into objects and specifying attribute values.

Plan activities and methods

The planning module supports the project manager in choosing construction methods, allocating resources, and decomposing methods into lower level activities. For each level of

decomposition and for each product class, the user defines a list of potential construction methods and ranks them according to his/her preferences. The reasoning starts with the creation of an abstract activity for the realization of the highest level product, e.g., 'realize-building' (see Fig. 11). The system selects the highest ranked construction method that is appropriate for that activity. Generally, the selection of a construction method is based on available resources, the product, and earlier choices. When an appropriate construction method is found, the system allocates resources to it.

The next step is to elaborate the construction method. This means that both the product and the activity are decomposed into related lower level products and activities. For products, the default decomposition is to follow the product decomposition as defined by the designer. This results in a list of part-products. The relations between the part-products are the 'support' and 'accessibility' relations. When a project manager wants to introduce construction zones, he/she can deviate from the default decomposition and add an additional product decomposition layer. For construction methods, the default decomposition is to directly follow the product decomposition. The relations between the part-activities directly correspond to the relations between the products. For example, a 'support' relation is translated to a 'succeeded-by' relation. In addition to these default settings, the user may specify the decomposition or the definition of the relations for a specific construction method. For example, when a tilt-up method is used to build a wooden frame, the sequence of the activities does not necessarily correspond to the 'support' relations of the columns and beams. Or, when defining the part-activities for realizing a concrete element on site, the activities should not only include activities that correspond directly to the part-products—such as 'place-concrete' and 'place-reinforcement'—but also auxiliary activities—such as 'place-form', 'cure-concrete', and 'remove-form'. After elaboration of a construction method, the process of selection and elaboration is repeated for each of the part-activities.

Companies can use this planning module to formalize and bring on line their construction knowledge. A company can define PtM modules for its kinds of projects and implement construction knowledge that reflects its way of working with specialized construction methods.

For the test case, the planning module reads the STEP file with the product model from the first module. It also reads a STEP file with the resources that are available for this project (Fig. 10). Based on the default preferences for construction methods and default activity decomposition, the planning module develops an activity decomposition tree that corresponds directly to the product decomposition (see Fig. 11). The construction methods selected are 'in-situ-concrete-wooden-form' for the footings, 'in-situ-concrete-prefab-wooden-form' for the columns, 'in-situ-concrete-prefab-steel-form' for the beams, and 'precast-concrete' for the slabs. These selections are based on the available resources and their capabilities.

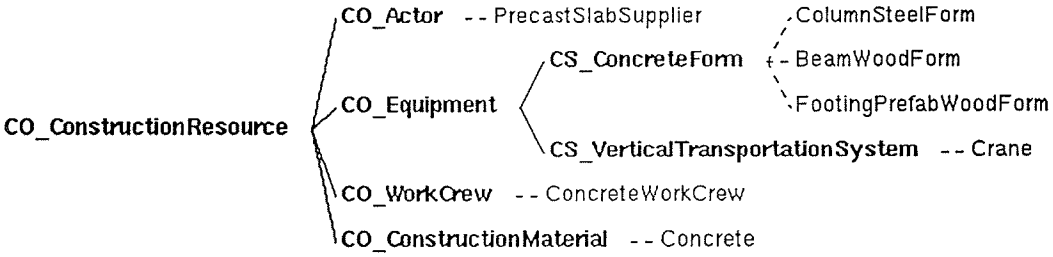

Fig. 10. Available resources.

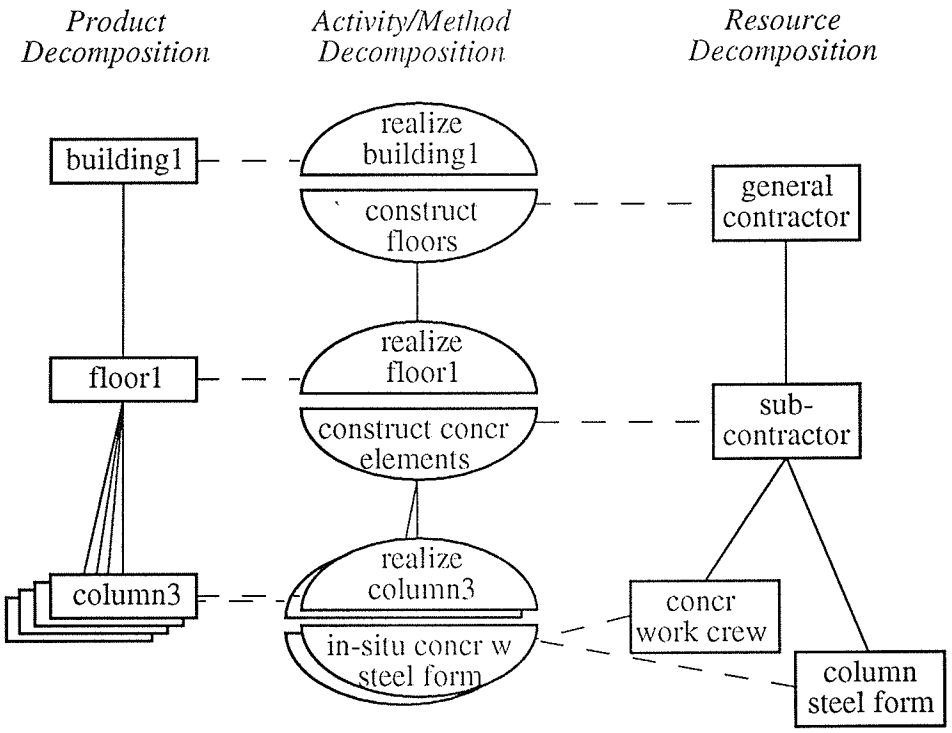

Fig. 11. Decomposition of products, activities/methods, and resources.

Schedule activities

After the activities, their dependencies, and the construction methods with resources are defined, the activities on the lowest decomposition level (leaf-activities) can be scheduled. In contrast to planning, the scheduling module does not only care about resource types, but also about resource availability.

First, the duration of each leaf-activity and the lag between leaf-activities are calculated. The duration of a leaf-activity depends on the construction method and the productivity of the allocated resources. The lag between leaf-activities depends on the construction methods for the activities and the reason for the precedence relation. For example, when a 'succeeded-by' relation exists because of a 'supported-by' relation and the preceding construction method is 'build-concrete-element-in-situ', the lag between the two activities is longer because the concrete has to cure first. Furthermore, the 'succeeded-by' relations between activities on higher levels are propagated to the lowest level. For example, a construction manager defines that one zone succeeds another. These precedence relations between zones are maintained at the level of leaf-activities.

Now, a first approximation of the total schedule duration is calculated using the Critical Path Method (CPM), with a forward and backward pass (Fondahl 1962). This determines the early-start, early-finish, late-start, late-finish dates and the critical path. This initial CPM analysis does not consider resource availability.

The second scheduling step takes resource availability into account. The scheduled start and finish dates of each leaf-activity are set to the early-start and early-finish dates from the initial CPM analysis. Since scheduled dates for an activity are only realistic if resources are available to start and complete the activity within its scheduled dates (Fondahl 1991), the system checks for which resources demand exceeds availability. It then calculates a measure of criticality for each resource in the context of the overall project. We define the criticality of

a resource as the sum over time of the duration for which a resource exceeds availability limits multiplied by the ratio of resource units required to units available. The resource with the highest criticality is leveled first. At a certain point in time, several activities might compete for this most critical resource. Activities that are already underway at this point in time continue. Unused resource units are then allocated to the activity with the smallest total float that does not exceed resource limits. Once the most critical resource has been leveled, the system recalculates the resource criticality for the other resources and levels the next most critical resource. Starting leveling with the most critical, or most constraining resource minimizes the number of iterations. More importantly, however, it shows the user which resources are likely to cause schedule delays during construction.

When all leaf-activities are scheduled, the results can be aggregated to higher level activities. Because every activity on each decomposition level is fulfilled by a construction method, the system is always able to produce a schedule, even if different branches are decomposed to different levels. This makes it possible to use this system from the early design phases on when only few details are known and allows continuous extension of detail.

The reasoning mechanisms in this scheduling module are well formalized and can thus be automated completely. This allows project managers to play with different levels of resource availability to optimize resource utilization and project duration. It also allows quick evaluation of alternative construction methods. The results of the scheduling module can easily be used as input to existing scheduling applications such as Primavera.

For the test case, we would like to investigate how many concrete work crews will be required to finish the project in 400 hours (50 working days). As shown in Fig. 12, the system automatically adds start and finish activities and considers sequence relations from higher decomposition levels when sequencing the leaf-activities. The first calculation of activity durations and lags and the scheduling of all activities results in a project duration of 174 hours (or approximately 22 working days). However, when we look at the resource histogram for the resource 'work crew', we see that its availability (i.e., 1 unit available) is exceeded by far (i.e., 4 units required; Fig. 13). Thus, the system needs to level the resources. It realizes that the resource 'work crew' is the most critical or constraining resource. After leveling, the project lasts 457 hours, i.e., longer than our target duration of 400 hours. To the reduce project duration, the user can add a work crew. The finish time now drops to hour 242. This example shows how construction managers can rapidly explore a number of alternatives, optimize resource usage, or respond quickly to changed conditions or project objectives. This is only possible because the software system is based on a high-level, semantically-rich project model.

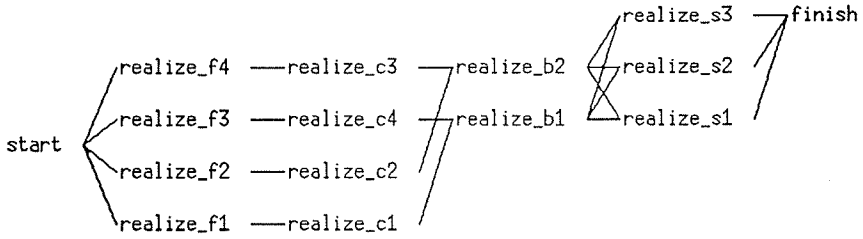

Fig. 12. Network of leaf-activities.

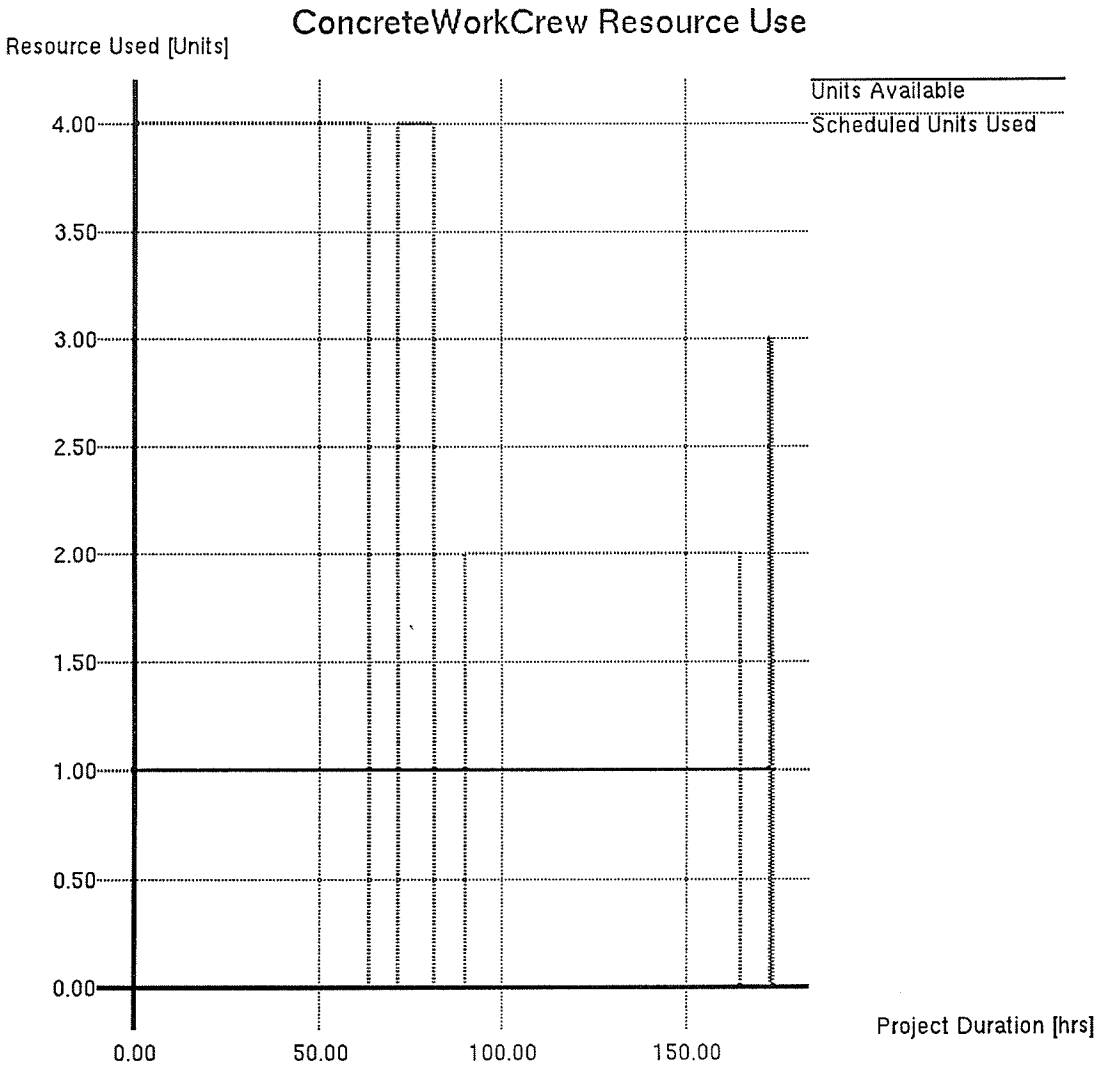

Fig. 13. Resource histogram for concrete work crew before leveling.

Simulate construction process

To present the results of the planning and the scheduling modules graphically, the construction process can be simulated and visualized in a 4D-CAD application (3D plus time). In a Kappa module we create the input for a visualization tool in AutoCAD. The visualization shows the realization of the different components over time. The AutoCAD tool allows the project manager to animate the complete construction process or walk through it step by step. Thanks to the integrated product-activity-method-resource model, the relationships between activities and components are created and maintained automatically. Thus, the construction sequence can be replayed and visualized immediately after the scheduling module has completed its reasoning. In contrast, current 4D-CAD tools require manual linking of components and activities (Collier and Fischer 1995).

Fig. 14 shows the user interface and the state of the building half way through construction for the test case. When looking at the simulation, the user would discover that the sequence of the elements is random because the system only considers dependency

relationships that follow from the 'support' relations. For example, footing construction starts with the footing in the front, continues on the left, then the right, and finishes in the back. A more circular sequence would probably save relocation time for the work crews and their equipment. To improve the automatically generated schedule we are considering the definition of product-flow relations between components in a next version of the system.

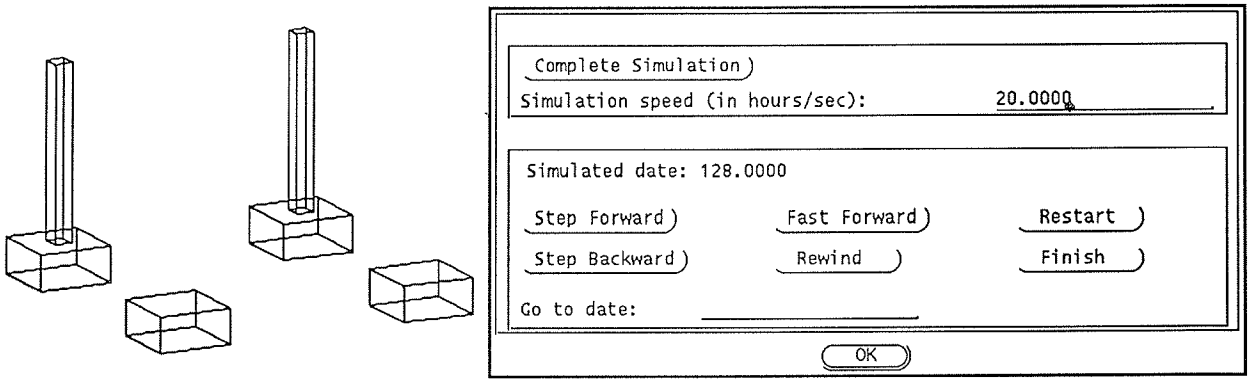

Fig. 14. Simulation of construction process with simulation dialog box.

Estimate construction cost

Based on the design, schedule, and detailed resource cost information, it is a straightforward task to calculate the direct and indirect construction costs of a project. So far, we have only implemented procedures that calculate direct cost. Thus, the cost calculation for the test case does not provide new insight into the efficiency of the plan (based on the chosen construction methods) and the schedule (based on resource availability).

CONCLUSIONS

Project management systems based on integration approaches like the one presented in this paper open the opportunity to make continuous cost and schedule feedback an integral part of design. On one hand, designers will be able to cycle through the 'design-planning-scheduling-estimating' process as often as they like and will thus be able to improve the constructibility of their designs. On the other hand, project managers will be able to work out more alternatives in detail because the repetitious portion of their work has been automated. These improvements will help the building industry understand and control the building process better and deliver facilities on time, at reasonable cost, and with good quality.

To advance the field of automated and integrated project management, contributions to theory, education, research, and practice are needed.

Theoretical challenges include:

- Formalization of knowledge about construction methods, resources, planning, scheduling, and cost estimating
- Increased understanding of the information flow required to support integrated project management and of the relations between project management concepts
- New insights into the nature and importance of activity dependencies
- Validation of the idea of supporting software integration with a shared model

Educational challenges include:
- Tools to simulate consequences of design and construction management decisions
- Systems that foster understanding of project management concepts

Research challenges include:
- Modular research platforms to test new project management concepts and to add new, refined reasoning modules

Practical challenges include:
- Improved resource management
- More efficient building projects
- Improved organizational and individual learning through the integrated representation of a project from the start
- Rapid feedback on alternatives
- Faster process and better elucidation of client requirements
- Improved alignment of project objectives and solutions

It is our research goal to continue to contribute to these challenges with observation, model building, implementation, and testing as reported in this paper.

ACKNOWLEDGMENTS

We thank Delft University of Technology, the Fulbright Scholar Program, Stanford's Center for Integrated Facility Engineering, and Stanford's Office of Technology Licensing for their support.

APPENDIX 1. REFERENCES

Aouad, G. and Price, A.D.F. (1993). "An Integrated Computer Model to Aid the Planning of Concrete Structures." *Microcomputers in Engineering*(8), 27-44.

Birrell, G.S. (1980). "Construction Planning—Beyond the Critical Path." *Journal of the Construction Division, ASCE*, 106 (3), 389-407.

Björk, B.C. (1991). "A Unified Approach for Modelling Construction Information." *Building and Environment*, 27 (2), 173-194.

Cherneff, J., Logcher, R., and Sriram, D. (1991). "Integrating CAD with Construction-Schedule Generation." *Journal of Computing in Civil Engineering, ASCE*, 5 (1), 64-84.

Clayton, M.J., Fischer, M.A., Kunz, J.C., and Fruchter, R. (1995). "Behavior Follows Form Follows Function: A Theory of Design Evaluation." *Second ASCE Congress on Computing in Civil Engineering*, Atlanta, ASCE, New York, NY, USA (accepted for presentation).

Clayton, M.J., Fruchter, R., Krawinkler, H., and Teicholz, P. (1994). "Interpretation Objects for Multi-Discipinary Design." *Artificial Intelligence in Design '94*, Gero, J.S. and Sudweeks, F. (eds.), Kluwer Academic Publishers, the Netherlands, 573-590.

Collier, E. and Fischer, M.A. (1995). *Four-Dimensional Modeling in Design and Construction*. Technical Report 101, CIFE, Stanford University, CA, USA.

Darwiche, A., Levitt, R., and Hayes-Roth, B. (1989). "OARPLAN: Generating Project Plans by Reasoning about Objects, Actions and Resources." *AI EDAM*, 2 (3), 169-181.

Echeverry, D., Ibbs, W., and Kim, S. (1991). "Sequencing Knowledge for Construction Scheduling." *Journal of Construction Engineering and Management, ASCE*, 117 (1), 118-130.

Fondahl, J. (1962). *A Non-Computer Approach to the Critical Path Method for the Construction Industry*. Technical Report 9, 2nd. Ed., Construction Institute, Stanford University, CA, USA.

Fondahl, J.W. (1991). "Development of the construction engineer: Past progress and future problems." *Journal of Construction Engineering and Management, ASCE*, 117 (3), 380-392.

Froese, T.M. (1992). *Integrated Computer-Aided Project Management Through Standard Object-Oriented Models*. Ph.D. thesis, Stanford University, CA, USA.

Gielingh, W.F. (1988). *General AEC Reference Model (GARM)*. BI-88-150, TNO Building and Construction Research.

Gielingh, W.F. and Suhm, A.K. (eds.) (1993). *IMPPACT Reference Model: An Approach for Integrated Product and Process Modelling of Discrete Parts Manufacturing*. Vol. 1, Research Reports ESPRIT: Project 2165 - IMPPACT, Springer-Verlag, Darmstadt, Germany.

Hendrickson, C. and Zozaya-Gorostiza, C. (1989). "Unified Activity Network Model." *Journal of Computing in Civil Engineering, ASCE*, 3 (2), 192-200.

IntelliCorp (1993). *Kappa*. IntelliCorp, Inc., Mountain View, CA, USA.

ISO/TC184 (1993a). *Part 1: Overview and fundamental principles*. Industrial automation systems and integration - Product data representation and exchange, Draft International Standard, ISO, Geneva, Switzerland, ISO DIS 10303-1.

ISO/TC184 (1993b). *Part 21: Clear Text Encoding of the Exchange Structure*. Industrial automation systems and integration - Product data representation and exchange, Draft International Standard, NIST, Gaithersburg, USA, ISO DIS 10303-21.

Jägbeck, A. (1994). "MDA Planner: Interactive Planning Tool Using Product Models and Construction Methods." *Journal of Computing in Civil Engineering, ASCE*, 8 (4), 536-554.

Kähkönen, K. (1993). *Modeling Activity Dependencies for Building Construction Project Scheduling*. VTT Publications 153, VTT, Technical Research Center of Finland.

Luiten, G.T., Froese, T.M., Björk, B.C., Cooper, G., Junge, R., Karstila, K., and Oxman, R. (1993). "An Information Reference Model for Architecture, Engineering and Construction." *Management of Information Technology for Construction*, Singapore, Mathur, K.S., Betts, M.P., and Wai Tham, K. (eds.), World Scientific Publishing Co. Pte. Ltd, 391-406 (CIB W78).

Luiten, G.T. and Tolman, F.P. (1991). "Project Information Integration for the Building and Construction Industries." *Fourth IFIP TC5 Conference on Computer Applications in Production and Engineering: Integration Aspects*, Bordeaux, France, Doumeingts, G., Browne, J., and Tomljanovich, M. (eds.), Elsevier Science Publishers B.V., 469-476.

Meyer, B. (1988). *Object-oriented Software Construction*. Series in Computer Science, Hoare, C.A.R. (eds.), Prentice Hall, Cambridge, UK.

Navinchandra, D., Sriram, D., and Logcher, R.D. (1988). "GHOST: Project Network Generator." *Journal of Computing in Civil Engineering, ASCE*, 2 (3), 239-254.

Nijssen, G.M. and Halpin, T.A. (1989). *Conceptual Schema and Relational Database Design: A fact oriented approach*. Prentice Hall.

Phan, D.H.D. (1993). *The Primitive-Composite (P-C) Approach: A Methodology for Developing Sharable Object Oriented Data Representations for Facility Engineering Integration*. Ph.D. thesis, Stanford University, CA, USA (also published as CIFE Technical Report 85A).

SofTech (1981). *Function Modeling Manual ($IDEF_0$)*. Vol. 4, Integrated Computer-Aided Manufacturing (ICAM), Architecture Part II, SofTech, Inc., Waltham, USA.
Stuckenbruck, L.C. (1981). *The Implementation of Project Management: The Professional's Handbook*. Project Management Institute, Addison-Wesley Publishing Company, Reading, PA, USA.
Teicholz, P. and Fischer, M.A. (1994). "Strategy for Computer Integrated Construction Technology." *Journal of Construction Engineering and Management, ASCE*, 120 (1), 117-131.
Tolman, F.P. (1991). *Integration Core Model for Product Modelling*. C90-PMP-02, CAM-I (also published as TNO report B-91-0885).
Waugh, L.M. (1990). *A Construction Planner*. Ph.D. thesis, Stanford University, CA, USA.

Automated and Integrated Management of Scope, Time, and Cost

This document outlines a system that uses product, resource, and construction method information and knowledge to link time and cost aspects to the scope of a project. This document serves as a research map for the development of the proposed system.

Overview of System

Purpose

1 The proposed system integrates scope, time, and cost aspects of a construction project and automates the generation of schedules and cost estimates. The system gives feedback to designers and construction managers on time and cost implications of decisions about design alternatives, construction methods and resources.

Representation

2 The central notions in the system are *product*, *resource* and *construction method*; the abstract notion of an *activity* connects these concrete notions.

3 The system supports scheduling and cost estimating at several levels of decomposition and detail from conceptual to detailed design. It supports scheduling and estimating, even when different parts of the building are designed at different levels of detail. The system allows use of decomposition mechanisms (e.g., zoning, work packaging) to break down construction work into manageable units.

4 To link scope to time and cost aspects, the system represents construction methods and resources at appropriate levels of decomposition.

Reasoning

5 Based on a given design and user-defined or preset choices for methods and resources, the system automatically develops an acceptable schedule and cost estimate.[1] Users (e.g., construction managers) should try to improve upon this base case by using their knowledge to make better choices than the system does. Through what-if analyses, users are able to assess multiple alternatives rapidly.

User Interface

6 Menus guide the interaction of users with the system. This interaction is graphical wherever possible (e.g., a user can define construction zones graphically in a 3D CAD model).

7 The output of the system uses natural idioms of construction management, such as CPM[2] schedules, resource histograms, estimating spreadsheets, 3D schedule animation. Feedback to design is given at all levels of decomposition.

Testing

8 The system is tested with synthetic examples (see Section XX) and with an ongoing construction project (foundations and structural system of Central Plant of San Mateo County Health Center) (see Section XX).

Limitations

9 The implementation will be a prove-of-concept prototype.

10 The system is not an automated design system. It does not generate a rough preliminary design to estimate cost, like Haztimator (Oralkan et al. 1994).

---

[1] This base case can be used for "going around the circle" (Fischer and Kunz, 1993).

[2] CPM stands for Critical Path Method. See, e.g., (Barrie and Paulson 1992)

Research Challenges

1 To integrate scope, time, and cost aspects of a project alternative, the system must *dynamically share semantic-rich* representations of the product, construction methods and resources. Current representations only focus on one type of information, e.g., 2D CAD drawings or a STEP-based product model, and do not necessarily consider the relation with other types of information (e.g., the relations with activities in a CPM schedule). Current electronic sharing of construction information is often limited to the static exchange of files (e.g., the integration of Timberline and Primavera).

2 To be useful from conceptual to detailed design, the system must support the continuous growing amount of information in a project. It must represent and reason about design, construction methods and resources at various levels of decomposition and detail. It must support seamless and dynamic transition between these levels. This is known as the *extensibility* problem of data models (see, e.g., Phan 1993, Froese 1992). Current systems usually support only one design phase.

3 To be useful for construction management, the system must support decomposition and aggregation of construction work by different categories (zones, contracts) and levels. This is known as the *flexibility* problem of the use of data models (Froese 1992). Current decomposition tools (e.g., fragnets in Primavera's P3) are static in nature and do not make the reasons for a particular breakdown explicit.

4 To generate schedules and cost estimates automatically, the system must formalize planning, scheduling, and estimating processes. Current scheduling and cost estimating tools require manual input and have no notion of the process and the use of information.

The system processes

A-0 Automated and Integrated Management of Scope, Time, and Cost in a building project

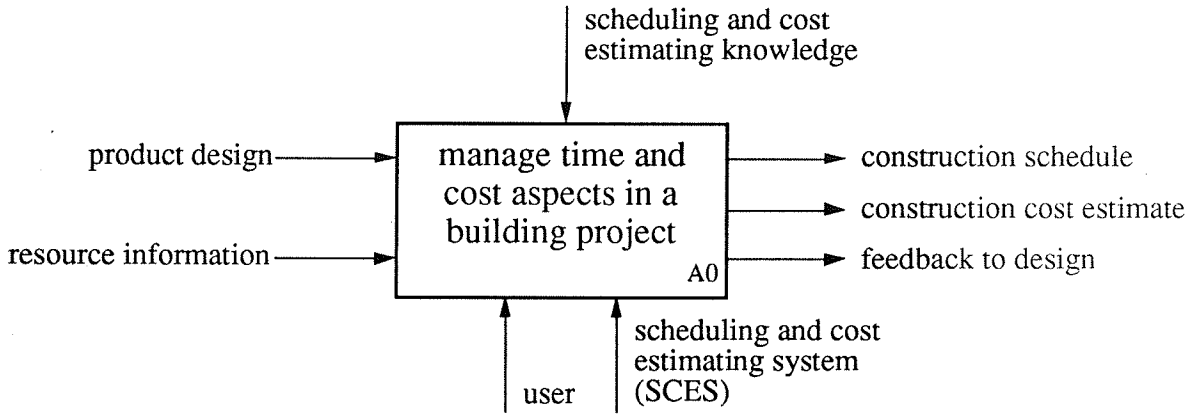

Figure 1 *A-0: Automated and Integrated Management of Scope, Time, and Cost in a building project; the main process is Manage time and cost aspects in a building project*

A0 Manage time and cost aspects in a building project

Based on the product design, resource information, and scheduling and cost estimating knowledge the *time* and *cost* aspects of a building project are managed. The designed product is the *scope* of construction. This process results in a construction schedule, a cost estimate, and feedback to design. The process is executed by the user, supported with a scheduling and cost estimating system (SCES).

Viewpoint of the IDEF0 model
Developer of an automated and integrated system to manage scope, time, and cost in a building project.

Goal of the IDEF0 model
Identify requirements for a scope, time, and cost management system based on sub-processes and information flows.

A0 Manage time and cost aspects in a building project

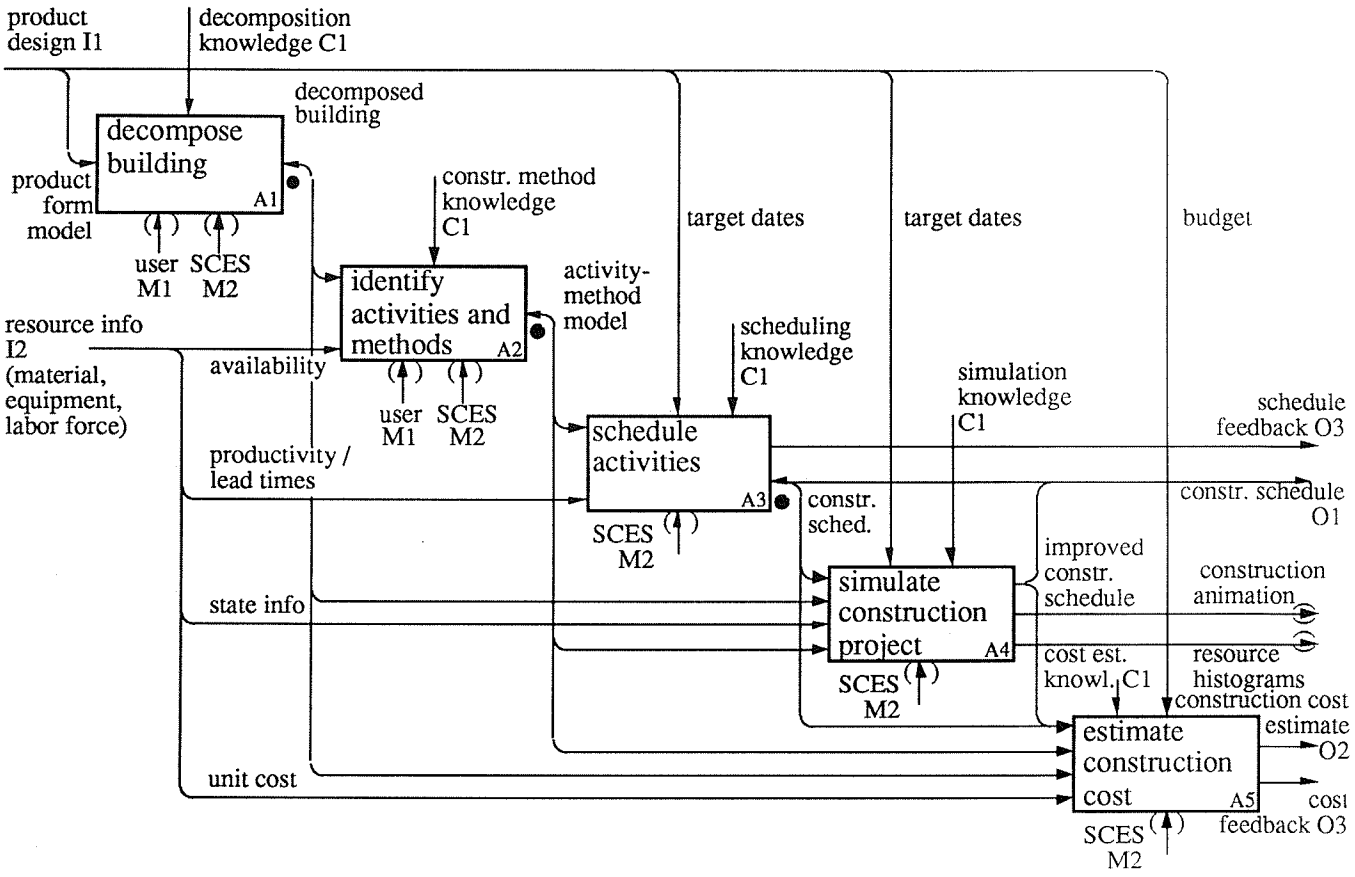

Figure 2 A0: Manage time and cost aspects in a building project

A1 Decompose building
Decomposes the building into construction areas based on the decomposition of the product as designed and general construction decomposition knowledge. A construction area is a part of the building that is considered as one unit in construction management at a certain level of detail. These parts can be determined by discipline (work packaging) or by a horizontal or vertical division of the building (zoning). Also the relations between the different parts are defined based on topological relations. This process results in the building decomposed from the construction management point of view. This decomposed building serves as a kind of master plan for the other management processes.

A2 Identify activities and methods
Determines the construction activities in a building project and the methods to realize these activities. The activities are based on the decomposed building (resulting from A1) and the chosen

methods. The methods are also based on the available resources. This process is done top-down: for the highest level activity (e.g., realize building) a method is selected; this method is decomposed into (related) more detailed activities for which again methods are selected; this is repeated until the activities and methods for the most detailed (or bottom) products are determined. Activities are related with predecessor/successor relationships. This process results in an activity-method model.

A3 Schedule activities
Based on the activity-method model, productivity and lead time information of the resources, and general scheduling knowledge (e.g., CPM knowledge), scheduling properties of the lowest level (or bottom) activities are determined. Scheduling properties are, e.g., early-start, total-float, and late-finish. These scheduling properties are aggregated for higher level activities. This process results in a construction schedule and feedback to design. The dates in the schedule (i.e., the behavior of (parts of) the building with respect to time) can be compared to the target dates (a functional requirement defined in the design or by project management).

A4 Simulate construction
Based on the construction schedule and construction simulation knowledge, the construction process is simulated using a similar approach as in (Waugh 1990). Waugh *generates* a construction schedule based on resource and product state limitations. Here the construction schedule (resulting form A3) is *checked* against the resource and building states. In case of conflicts, the construction schedule is updated. Issues are, e.g., resource leveling and time-cost trade-off (see also Axworthy 1990). This process results in an improved construction schedule, input for construction animation, and resource histograms.

A5 Estimate construction cost
Based on the (improved) construction schedule and/or the activity-method model, unit cost information, and general cost estimating knowledge, direct cost of the bottom activities are determined. When a complete schedule has been determined the costs are calculated by summing resource costs, otherwise costs are approximated with heuristic knowledge (e.g., cost per square feet). The direct costs are aggregated for higher level activities. Cost calculation is completed with calculating indirect costs for the project. This process results in a construction cost estimate and feedback to design. The costs of product parts (i.e., the behavior of a product with respect to costs) can be compared to the budgets (i.e., the functional requirements on the product).

I1 Product design
The product as designed by the designer(s). The product design consists of the form model of the product, target dates for the construction schedule and budgets for the cost estimate. The form model of the product consists of geometry, material, and topological relations.

I2 Resource information
Information on available construction resources, such as materials, equipment and labor force. Resource information consist of information on availability, productivity , lead times, resource states, and unit costs.

C1 Scheduling and cost estimating knowledge
Scheduling and cost estimating knowledge is either available in the heads of the project managers or formalized in the computer system. The more knowledge is formalized in the system, the better the system is able to develop a good schedule and cost estimate. Scheduling and cost estimating knowledge consists of building decomposition knowledge, construction method knowledge, scheduling knowledge, simulation knowledge, and cost estimating knowledge.

O1 Construction schedule
Input for existing (commercially available) software for presentation of resulting construction schedules (e.g., Primavera's P3).

O2 Construction cost estimate
Input for existing (commercially available) software for presentation of the resulting construction cost estimate (e.g., Timberline).

O3 Feedback to design
Makes results available for designers.

M1 User
The user (designer, construction manager) that executes the scheduling and cost estimating activities. The user is supported by a scheduling and cost estimating system.

M2 Scheduling and cost estimating system (SCES)
For "going around the circle" the system executes these processes by itself, for construction management the system supports the user to execute them.

A1 Decompose building

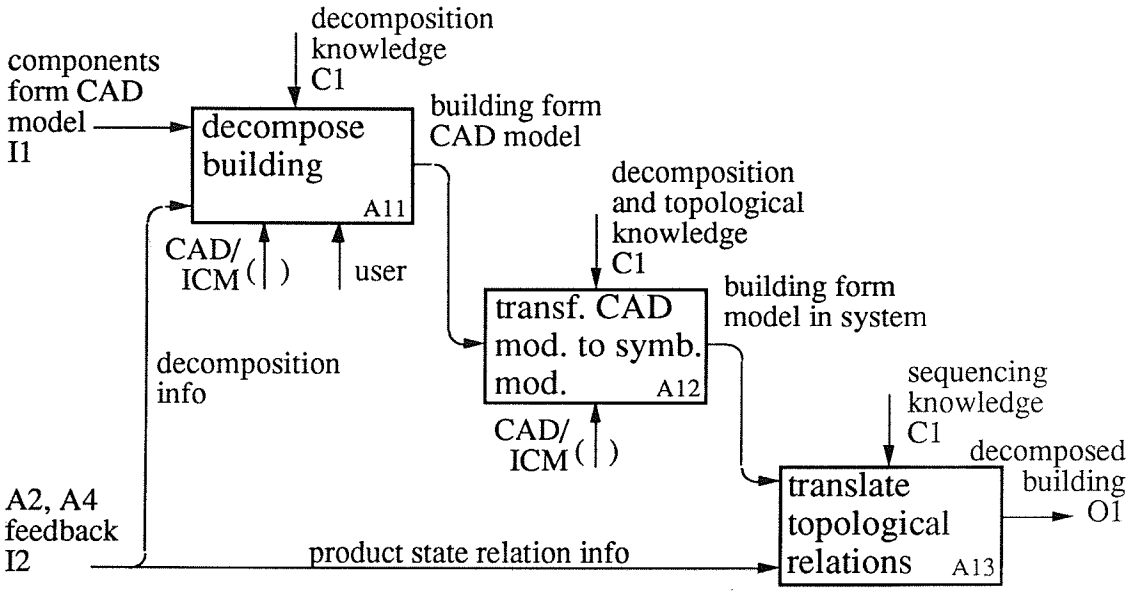

Figure XX A1: Decompose building

A2 Identify activities and methods

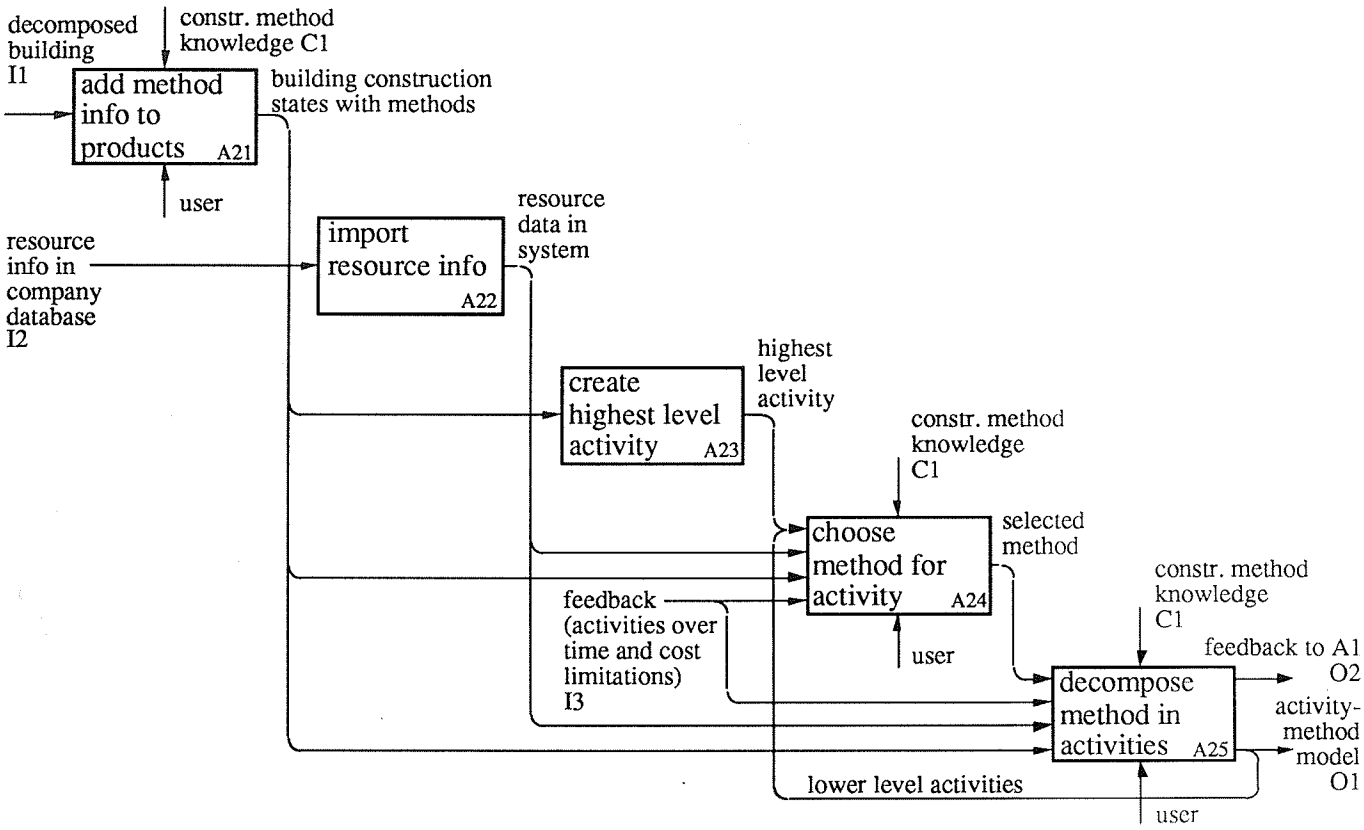

Figure XX A2: Identify activities and methods

A3 Schedule activities

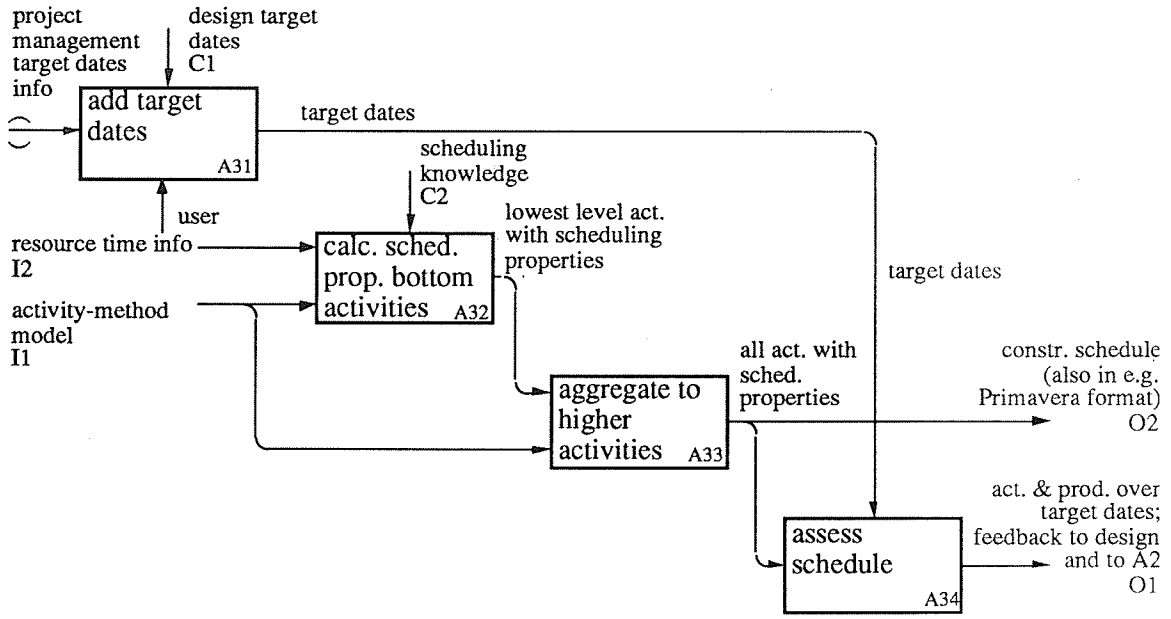

Figure XX A3: Schedule activities

A4 Simulate construction project

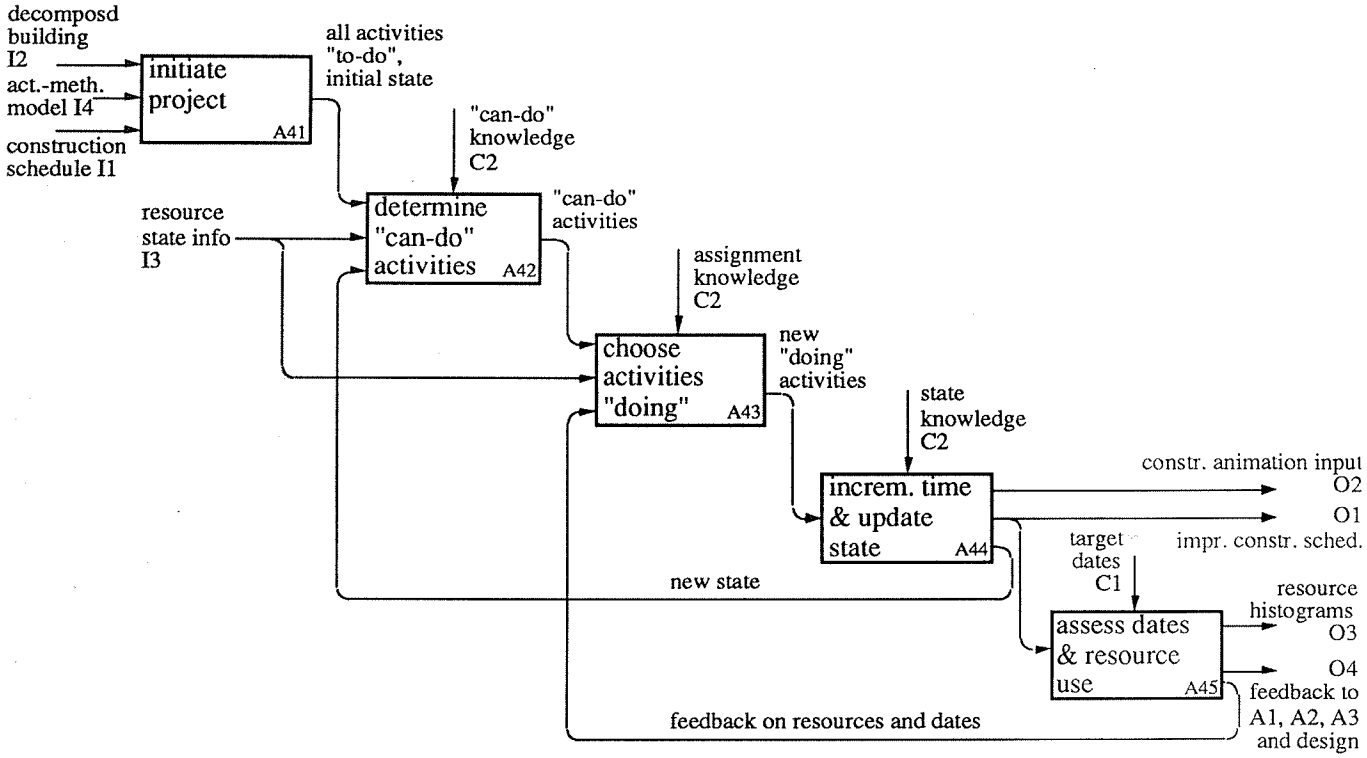

Figure XX A4: Simulate construction project

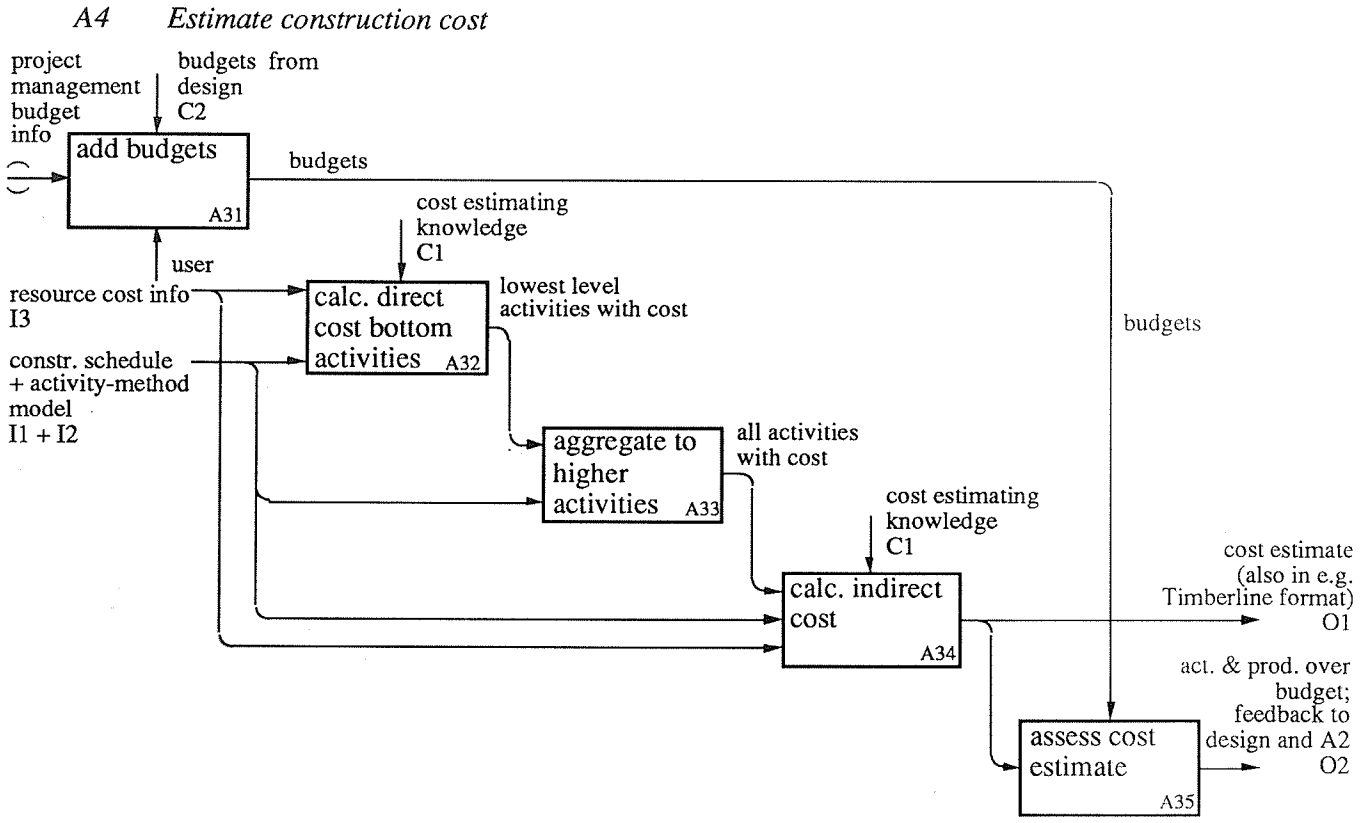

Figure XX A5: Estimate construction cost

The modules

Module 1 Determine construction state requirements of building
This module is deals with the planning part of scheduling.

1 import form model (product + components, decomposition, topological_relations (supports, encloses, ...), shape, material); (save CAD handle for feedback to design)
2 product top-down: define state_relations for product for all decomposition levels based on topological_relations

Module 2 Determine activity-method model
This module is the core of the system with most of the reasoning.
Basic information structure: an activity is_fulfilled_by a method; a method consists_of activities.

1 add possible_methods to products (default when product form model is imported and/or user-defined); change sequences of methods, if deemed necessary
2 import general material and other resource data from company database
3 add start_construction, finish_construction activities to project and an activity to the most_abstract_product (e.g., realize_building_1 to building_1, or realize_structural_system_1 to structural_system_1 if you only want to schedule structural_system_1) and relate the three top activities
4 activity/method top-down: choose method (default first possible_method of the product for which resource requirements are fulfilled by available resources)
5 if enough information:

a use method knowledge and state_relations to decompose method in lower level activities with time_relations, resources, delta_product, delta_resource, actor, etc.; default: all activities that are not state_related are done at the same time, available resources impose limitations
(knowledge about method decomposition is stored in programming code; users are able to alter a decomposition or to define a new one)

b for each lower level activity: go to 4 (so each activity has a selected method, otherwise you cannot calculate, e.g., expected_duration)

else: stop

Module 3 *Schedule activities*

This module only calculates the time consequences of choices in 2 using CPM algorithms.

1 input target dates for products (if externally defined then already imported with product form model)

2 for all the activities on the lowest level of decomposition:
if enough information:

a calculate expected_duration based on method

b CPM forward pass
find bottom-level activities that are not preceded by other activities, set ES = project.start_date, EF = ES + expected_duration
for all successors: find largest EF of predecessors, set ES = largest EF, EF = ES + expected_duration

c CPM backward pass
find bottom-level activities that are not followed by other activities, set LF = project.most_abstract_product.target_realization_date, LS = LF - expected_duration, TF := LS - ES, FF := smallest ES of all successors - EF
for all predecessors: find smallest LS of successors, set LF = smallest LS, LS = LF - expected_duration, TF := LS - ES, FF := smallest ES of all successors - EF
on_critical_path := TF = 0

d resource leveling algorithm: determine scheduled_start (SS), scheduled_finish (SF), etc.

e if problems (e.g., resources already used, leveling not possible): enlarge project duration

else: estimate duration for method at higher decomposition level

3 aggregate scheduled dates (+ duration, TF, FF) to higher level activities and products

4 if product.realization_dates > product.target_dates: feedback to A2 and/or design

5 feedback results to design

6 export bottom level activities + activity aggregation information + CPM data to PrimaVera for documentation

7 export bar chart information to AutoCAD for graphical feedback in same application as used in design

Module 4 *Simulate construction*

This module assesses the results of 2 and 3 by simulating construction activities, using a similar approach as Lloyd Waugh. In first version, only asses schedule, give feedback information and report problems. In later version, knowledge can be added to improve the schedule and to solve the problems.

1 initialize project: set simulated_date := actual_date; for all project.objects: simulated_state := actual_state; initialize in_progress and done list based on actual date

2 put all bottom level (= most detailed) activities in todo list of project

3 until project.state = finished (simulated_date >= project.finish_date)

a check which of the in_progress activities are finished at simulated_date (for each activity in progress: if SF <= simulated_date ...)
change activity.state to "finished", move to done list, update product.state

b find todo activities that should start at simulated_date: for each activity, check resource availability, product state, method.can_be_done
if not OK; send message (or, in later version: improve schedule one way or another)
change activity.state to in_progress, move to in_progress activities, update product state

c set new relevant simulated_date (smallest (SF of in_progress activities and SS of todo activities))

3 asses simulation:
check product.target_dates
for each resource: asses use (time idle, productivity, etc.) e.g. in time-use chart
export product + relevant dates to AutoCAD for graphical animation (send back all products for which a CAD id is known with SS, SF, and on_critical_path)

Module 5 Estimate construction cost

This module only calculates the cost consequences of choices in 2.
(NB this must be elaborated in much more detail to include overhead, fixed resource costs, resource cost/use, resource cost/time in project, etc., see, e.g., MOCAII)

1 input budget for products (if externally defined then already imported with product form model)
2 if all components and activities known: Sum resource cost (material is also a resource)
else if all components known: Sum product cost * indices
else estimate on higher abstraction level of product
3 aggregate cost to higher level resources, activities and products
4 asses cost results: if product.budget > product.expected_cost: feedback to A2 and/or design
5 feedback results to design: send back all products for which a CAD id is known with cost

General issues

1 do I need to add uncertainty (reliability, range) to the schedule and the estimate? something like PERT?
2 what product information do I need from design (especially the decomposition structure)?
3 how do I get that information from the AutoCAD 3D component model?
4 how to add a new decomposition level into an existing level (e.g., how to add a construction zone between a floor-level and the component level)?
5 in module 4: also improving the schedule?
6 how to model productivity of a resource: as a function of (product and time) or (method and time) or (method, product, time)? (productivity of a resource is based on unlimited availability of other resources ?)

Implementation plan

Follow chronological sequence of scenario as much as possible

1.1 set up main classes (project, product, shape, material, topological and state-relations) with main information structure
1.2 add general messages etc. to add information to information structure
1.3 import product form model from ASCII file (this gives experience with what product information is needed and how to deal with it; later this file can be derived from the AutoCAD 3D model)
1.4 reason about topological relations, translate to state-relations
1.5 test for different levels of detail of design

2.1 add main classes (method, activity, resource, resource_use, time_relations) with main information structure

2.2 add general messages etc. to add information to information structure
2.3 import resource data from ASCII file
2.4 add new methods with can_be_done and decomposition knowledge
2.5 add methods to products
2.6 test for different levels of detail of design

etc.
etc.

ProKappa characteristics
1 implement all modules in one ProKappa application
2 inverse relations can be kept update in ProKappa
3 use Value_type for type checking
4 use standard OO as much as possible (= separation class/instance, attributes, messages: no rules, no monitors)
5 use ProKappa as much as possible, be aware of backtracking (also use it? it becomes a bid messy)
6 graphical display in ProKappa is rather slow; this should be no problem in end-user version
7 What happens with redefinitions of attributes, messages?

The manual computer

Test cases (also usable for prototyping):
1 conceptual design: only floor plan and number of floors are known
2 definite design: idem + structure is decomposed in elements

Test case 1 conceptual design

building_1 decomposes in: spatial_system_1
structural_system_1
architectural_system_1
foundation_system_1
heating_system_1 etc.

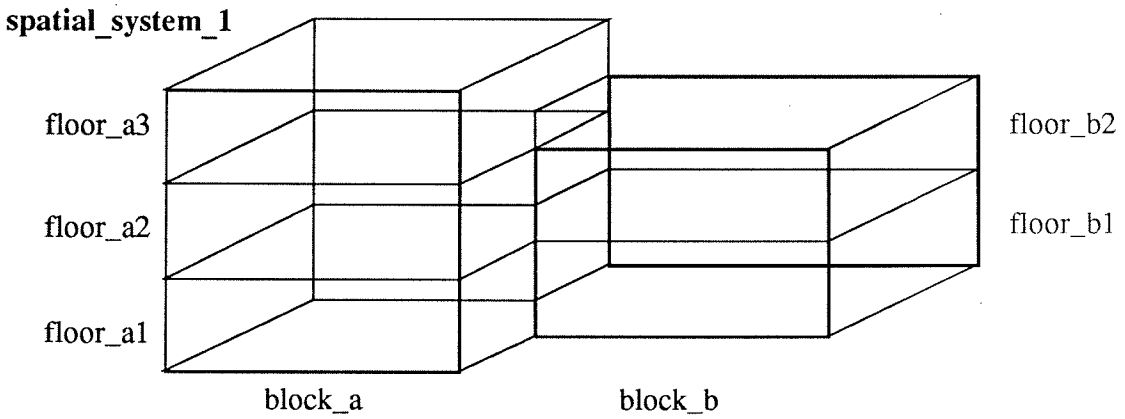

figure 2 *test case 1: graphical representation of product form model of the spatial system of an office building during conceptual design; the model includes shape and material information, and decomposition and topological relations; it also includes the top level decomposition of a building in its systems*

Module 1:

2 translate topological relations to state-relations:
foundation_system_1 realized_before structural_system_1
structural_system_1 realized_before architectural_system_1
structural_system_1 realized_before heating_system_1

Module 2:

1 add possible_methods to products
building_1.possible_methods := (sequential_systems, sequential_floors)
foundation_system_1.possible_methods := (concrete_piles, footings)
structural_system_1.possible_methods := (in_situ_concrete, steel, precast_concrete)
archtitectural_system_1.possible_methods := (prefab_elements)
heating_system_1.possible_methods := (prefab_elements)
2 resource information: not relevant for this level
3 add activity "realize_building" to building_1
4 choose method: sequential_systems
5a decompose method in activities with relations
realize_foundation_system_1 sequential realize_structural_system_1
realize_structural_system_1 sequential realize_architectural_system_1
realize_structural_system_1 sequential realize_heating_system_1
5b go to 4 for each lower level activity
realize_foundation_system_1.method: concrete_piles, no decomposition
realize_structural_system_1.method: in_situ_concrete, no decomposition
realize_architectural_system_1.method: prefab_elements, no decomposition
heating_system_1.method: prefab_elements, no decomposition

Module 3:

1 set target dates
building_1.target_start_date := July 1, 94
building_1.target_realization_date := July 1, 96
2a calculate duration of bottom-level activities, based on chosen method (which are not decomposed):
realize_foundation_system_1.expected_duration := 6 months
realize_structural_system_1.expected_duration := 8 months
realize_architectural_system_1.expected_duration := 6 months
heating_system_1.expected_duration := 12 months
2bc CPM forward and backward pass

table 1 *CPM information of bottom level activities in months for test case 1*

	D	ES	EF	LF	LS	TF	FF
realize_found_system_1	6	7/94	1/95	1/95	7/94	0	0
realize_structu_system_1	8	1/95	9/95	9/95	1/95	0	0
realize_archite_system_1	6	9/95	3/96	9/96	3/96	6	0
heating_system_1	12	9/95	9/96	9/96	9/95	0	0

2ef resource leveling not relevant, scheduled_dates = early dates
3 aggregate:

table 4 *aggregated CPM information in months for test case 2*

	D	ES	EF	LF	LS	TF	FF
realize_building_1	26	7/94	9/96	9/96	7/94	0	0

4 problem: realize_building_1.scheduled_finish > building_1.target_realization_date

solutions:
a change building.method to sequential_floors (i.e., structural_system still sequential with architectural_system and heating_system, but now on level of floors: concurrent construction); do module 1 and 2 again, but now use spatial system information to decompose systems and system level activities also on system-floor level
b change structural_system method to precast concrete: duration = 6 months

5 feedback results to design: resulting preferred structural system
6 PrimaVera not relevant
7 export bar chart information to AutoCAD

Module 4:

1 simulated_date := July 1, 1994; for all project.products: simulated_state := "designed"
2 simulation
a simulated_date := July 1, 1994: no activities finished
b realize_foundation_system_1: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, foundation_system_1.simulated_state := under_construction
c new relevant simulated_date := realize_foundation_system_1.SF = January 1, 1995
a simulated_date := January 1, 1995: realize_foundation_system_1.SF <= simulated_date => realize_foundation_system_1.simulated_state := finished, foundation_system_1.simulated_state := realized
b realize_structural_system_1: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, structural_system_1.simulated_state := under_construction
c new relevant simulated_date := realize_structural_system_1.SF = September 1, 1995
a simulated_date := September 1, 1995: realize_structural_system_1.SF <= simulated_date => realize_structural_system_1.simulated_state := finished, structural_system_1.simulated_state := realized
b realize_architectural_system_1: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, realize_architectural_system_1.simulated_state := under_construction
realize_heating_system_1: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, realize_heating_system_1.simulated_state := under_construction
c new relevant simulated_date := realize_architectural_system_1.SF = March 1, 1996
a simulated_date := March 1, 1996: realize_architectural_system_1.SF <= simulated_date => realize_architectural_system_1.simulated_state := finished, architectural_system_1.simulated_state := realized
b no new activities to start
c new relevant simulated_date := realize_heating_system_1.SF = September 1, 1996
a simulated_date := September 1, 1996: realize_heating_system_1.SF <= simulated_date => realize_heating_system_1.simulated_state := finished, realize_heating_system_1.simulated_state := realized
b no new activities to start
c no new relevant simulated_date; project.simulated_state := finished;

3 asses simulation:
check product.target_dates: same problem as in Module 2
no resources to asses
export CAD ids + relevant dates to AutoCAD

Module 5: Estimate construction cost:

1 set budget
building.budget := 1.000.000

2 not all components and activities known
not all components known
estimate on higher abstraction level of product:

table 3 *cost table for test case 1*

	calcuation procedure	values	cost
foundation_system_1	area * cost/area		
structural_system_1	volume * cost/volume		
architectural_system_1	volume * cost/volume		
heating_system_1	volume * cost/volume		

3 building.expected_cost = ...
4 building.budget > building.expected_cost: feedback to A2 and/or design
5 export CAD ids + cost

Test case 2: structural system in definite design

building_1: spatial_system_1: floor_1
architectural_system_1
foundation_system_1 footing1,2,3,4
heating_system_1
structural_system_1: column1,2,3,4
beam1,2
slab1,2,3

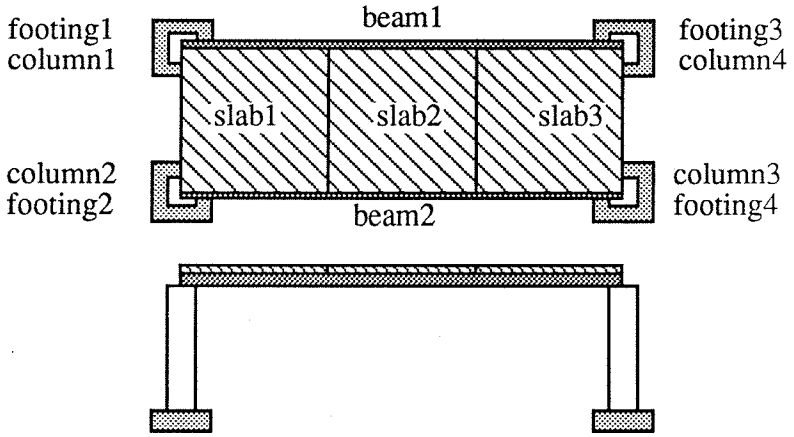

figure 3 *test case 2: spatial, foundation and structural system of a very simple building detailed until components*

Module 1:
1 import elements + decomposition structure + topological_relations
2 translate topological_relations to state_relations
foundation_system_1 realized_before structural_system_1
footing*i* realized_before column*j*
beam*i* realized_before column*j,k*
slab*i* realized_before beam*j,k*

Module 2:

1 add possible_methods to products
building_2.possible_methods := (sequential_systems, sequential_floors)
foundation_system_1.possible_methods := (in_situ, precast)
structural_system_1.possible_methods := (sequential_types, sequential_zones)
footing*j*.possible_methods := (in_situ_steel_form, in_situ_prefab_wood_form, in_situ_wood_form)
column*j*.possible_methods := (precast, in_situ_steel_form, in_situ_wood_form)
beam*j*.possible_methods := (precast, in_situ_steel_form, in_situ_wood_form)
slab*j*.possible_methods := (precast, in_situ_steel_form, in_situ_wood_form)

2 resource information:
2 column_steel_forms, productivity/unit: 1 column/week
? beam_wood_form, productivity/unit: 1 beam/week (NB ? denotes unlimited)
3 footing_prefab_wood_form, productivity/unit: 1 footing/week
1 precast_slab_supplier
1 crane, capacity/unit = 30 ton, productivity/unit: 4 precast_element/day
1 concrete_work_crew, productivity/unit: 4 footings/week if in_situ_wood, 4 colums/week if in_situ_steel, 2 colums/week if in_situ_wood, 1 beam/week if in_situ_wood, 1 slab/week if in_situ_wood, 1 slab/day if assemble
? concrete

3 add activity "realize_building" to building_2

4 choose method: sequential_systems

5a decompose method in activities with relations
realize_foundation_1 sequential realize_structure_1

5b go to 4 for each lower level activity
realize_foundation_1.method: in_situ
decompose in realize_footing*i* with time_relations (only 3 footings at the same time: limited by availability of footing_prefab_wood_forms)
realize_footing*i*.method: in_situ_wood_form (no steel_form available)
realize_structure_1.method: sequential_types
decompose in: realize_columns, realize_beams, realize_slabs
realize_columns.method: in_situ_steel_form (no precast_supplier available)
decompose in realize_column*i* with time_relations (time_relations limited by number of available resources: only 2 column_forms available thus only 2 columns at the same time!)
realize_column*i*.method: in_situ_steel_form
realize_beams.method: in_situ_steel_form (no precast_supplier available)
decompose in realize_beam*i* with time_relations (only one beam at the same time: limited by productivity of work crew and availability of beam_wood_forms)
realize_beam*i*.method: in_situ_wood_form (only two
realize_slabs.method: precast
decompose in realize_slab*i* with time_relations (only 1 slab at the same time: limited by productivity of work crew)
realize_slab*i*.method: assemble

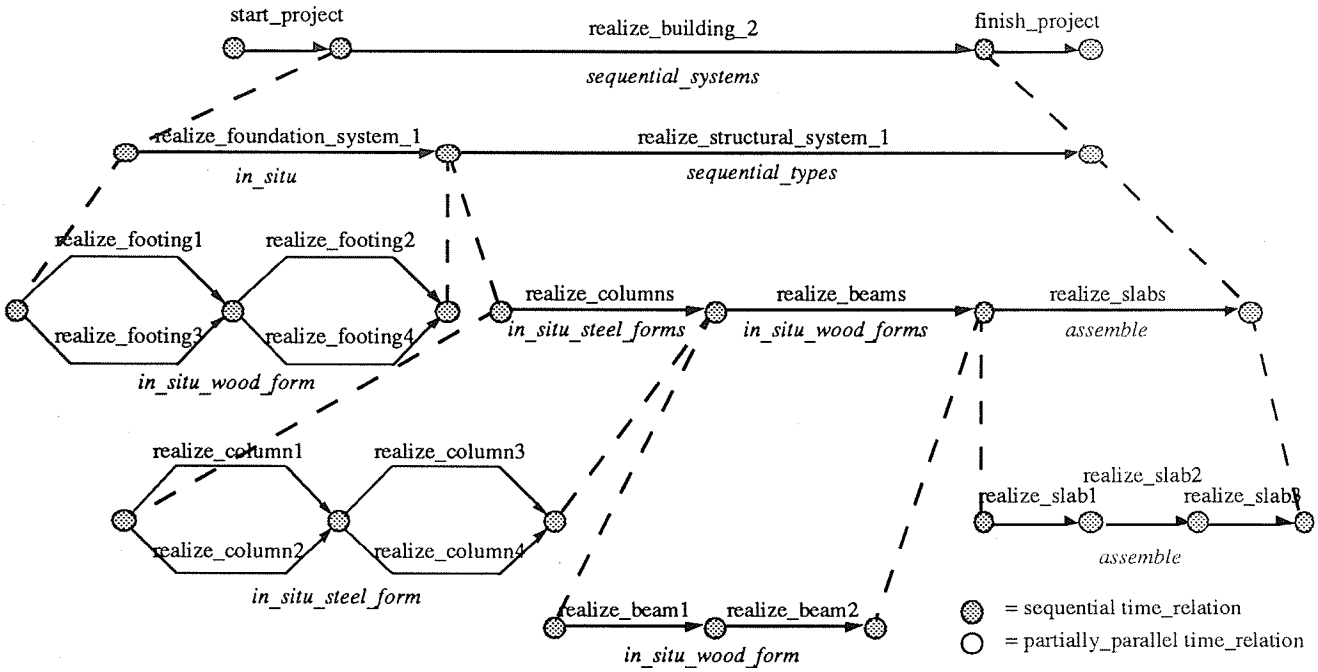

figure 4 *activity-method model of test case 2, represented with activity on arrow diagram; plain text denotes an activity, italic text denotes the chosen method*

Module 3: CPM

1 set target dates
  building_2.target_start_date := August 1, 94 = 0 weeks
  building_2.target_realization_date := September 8, 94 = 5 weeks

2a calculate duration of bottom-level activities, based on chosen method:
  realize_footing*i*..expected_duration := 1.0 week
  realize_column*i*.expected_duration := 1.0 week
  realize_beam*i*.expected_duration := 1.0 week
  realize_slab*i*.expected_duration := 0.2 week

2bc CPM forward and backward pass

table 4 *CPM information of bottom level activities in weeks for test case 2*

	D	ES	EF	LF	LS	TF	FF
realize_footing1	1.0	0	1.0				
realize_footing2	1.0	0	1.0				
realize_footing3	1.0	0	1.0				
realize_footing4	1.0	1.0	2.0				
realize_column1	1.0	2.0	3.0				
realize_column2	1.0	2.0	3.0				
realize_column3	1.0	3.0	4.0				
realize_column4	1.0	3.0	4.0				
realize_beam1	1	4.0	5.0				
realize_beam2	1	5.0	6.0				
heating_slab1	0.2	6.0	6.2				
heating_slab2	0.2	6.2	6.4				
heating_slab3	0.2	6.4	6.6				

2ef resource leveling not relevant, scheduled_dates = early dates

3 aggregate:

table 5 *aggregated CPM information in weeks for test case 2*

	D	ES	EF	LF	LS	TF	FF
realize_found_system_1	2.0	0	2.0				
realize_columns	2.0	2.0	4.0				
realize_beams	2.0	4.0	6.0				
realize_slabs	0.6	6.0	6.6				
realize_structu_system_1	4.6	2.0	6.6				
realize_building_2	6.6	0	6.6				

4 problem: realize_building_2.scheduled_finish > building_1.target_realization_date
solutions:
- a add extra resources: footing_prefab_wood_form or 2 column_steel_forms
- b add extra work crew for realization of beams
- c do not use sequential_types method but sequential_supported_by method (a method in which the sequence is determined by the topological_relations between individual elements)
- d precast columns and/or beams

5 feedback results to design: resulting dates, slabs precast, no other design alterations required

6 export bottom level activities + activity aggregation information to PrimaVera

7 export bar chart information to AutoCAD

Module 4:

1 simulated_date := 0; for all project.products: simulated_state := "designed"

2 simulation
- a simulated_date := 0: no activities finished
- b realize_footing_1: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, footing_1.simulated_state := under_construction
  realize_footing_2: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, footing_2.simulated_state := under_construction

realize_footing_3: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, footing_3.simulated_state := under_construction
c new relevant simulated_date := realize_footing_1.SF = 1.0
a simulated_date := 1.0: realize_footing_1.SF <= simulated_date => realize_footing_1.simulated_state := finished, footing_1.simulated_state := realized
simulated_date := 1.0: realize_footing_2.SF <= simulated_date => realize_footing_2.simulated_state := finished, footing_2.simulated_state := realized
simulated_date := 1.0: realize_footing_3.SF <= simulated_date => realize_footing_3.simulated_state := finished, footing_3.simulated_state := realized
b realize_footing_4: resource availability OK, product state OK, method.can_be_done OK => simulated_state := in_progress, footing_4.simulated_state := under_construction
c new relevant simulated_date := realize_footing_4.SF = 2.0
a simulated_date := 2.0: realize_footing_4.SF <= simulated_date => realize_footing_4.simulated_state := finished, footing_4.simulated_state := realized
... etc. ...
b no new activities to start
c no new relevant simulated_date; project.simulated_state := finished;

3 check product.target_dates: same problem as in Module 2
asses resources:

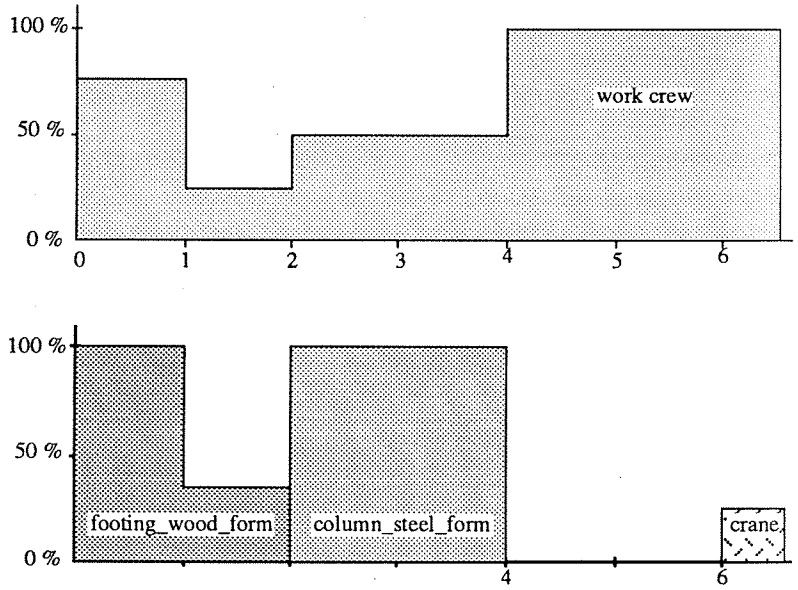

figure 4 *time-use graphs for resources of test case 2; form work is limiting in the first four weeks of the project and the work crew in the last two and a half weeks*

export CAD ids + relevant dates to AutoCAD

Module 5: Estimate construction cost:
1 set budget
building.budget := 100.000
2 all components and activities known: estimate cost from resources

table 6 *cost table for test case 2*

	calcuation procedure	values	cost
column_steel_form	weeks * cost/week		
beam_wood_form	area * cost/area		
footing_prefab_wood_form	area * cost/area		
precast_supplier	contract		
crane	time * cost/time		
concrete_work_crew	time * cost/time		
concrete	volume * cost/volume		

3 aggregate
4 asses cost results
5 export CAD ids + cost

Demonstration case: foundation and structural system of utility building of San Mateo Hospital

Eric's 3D CAD model contains all components at the level of structural elements. The CAD model is linked with a schedule. The components are organized in layers. Each layer contains a group of elements which are realized in one activity of the schedule.
I think that we should not use the extra information we can derive from the layers, because these layers are normally not available after design. Therefore, we must find other ways to extract information (e.g., on aggregation/decomposition of elements) from the 3D model.
With other words, we must see the 3D model as a set of elements (with form and material specifications) as it results form design. To these design 3D models, project managers have to add decomposition and other information they think is relevant.

The demonstration utility building consist of three blocks: low level, middle level and top level. Each block only has one floor. The foundation consists of concrete piles, footings, beams, walls, and slabs. The structure consists of steel beams and columns. Joints are not modeled.
The dimensions of the concrete elements reflect the design. The steel elements have arbitrary dimensions.

Design information needed by the construction manager:
1 elements with shape (either as shape_definitions, such as blocks, or as calculated attributes, such as area, volume) and material information
2 topological relations
3 basic aggregation/decomposition information: spatial system (building, blocks, floors, rooms) with shape and function
4 target dates and budgets (mainly for higher level products)
5 ? information about construction methods as intended by designer
6 design functional/design attributes that can be used for scheduling and cost estimating during conceptual design (e.g., when a building has an office function you know the approximate cost per square foot, or with detailed soil information you can make better estimates for foundation cost and duration)

In the current version the 3D model does not contain all this design information. To complete the design, symbolic information has to be added to the 3D model, e.g., classification of all elements, material specifications to all elements, steel profile type information to steel elements, and reinforcement percentages to concrete elements. (Maybe we can use the layering information of Eric's model to shorten this time consuming activity that normally should be done by designers.

We do not focus on design but on construction planning!) Graphical infornation that has to be added is mainly the basic aggregation/decomposition information.

Construction management information that should be added to the 3D design by the construction manager:

1 extra aggregation/decomposition information (e.g., needed when a floor will be build in three phases)
2 cost estimating and scheduling attributes (??)
3

Possible ways to add information to or to derive information from a CAD drawing:

1 draw spatial elements (e.g., building blocks or floors) in a separate layer; use AutoCAD functionality to derive which elements are enclosed by these spatial elements; use this as a basis for decomposition of the building during construction.
2 use AutoCAD functionality to derive topological relations between elements (e.g., supports)
3 use AutoCAD functionality (from geometry domain of Marc) to derive geometrical characteristics, e.g., volume, area, length
4 use ICM features to classify graphical objects (feature_class_name = ProKappa class name
5 use material definition available in AutoCAD
6 use AutoCAD attributes
7 use data field in ICM feature

SME+ = SME with attributes and inheritance

Bart Luiten, August 1994

Background

To develop schedules and cost estimates in an object-oriented environment, design information must be available in a symbolic model. To a large extend, this model can be derived from a 3D geometric model, which is developed during design by architects and engineers. However, much design information is not represented in the geometric model, but is still important for scheduling and cost estimating. Examples are material information, topological relations, and design intent. With the current version of SME (Clayton et al. 1994) it is possible to identify and classify geometric objects. However, this is not enough. It should be possible to add attributes to the classified objects.

There a various ways to add attributes to an AutoCAD 3D model:

1 use AME material definition: only limited number of predefined attributes possible that all deal with material properties, not changeable per AutoCAD entity
2 use AutoCAD attributes: only possible to add basic attributes (string, real, integer, etc.), necessitates much work in the drawing (e.g., new blocks creating)
3 extended entity data: only accessible with LISP functions (is not a major problem)
4 add attributes to SME features
5 .....

Advantages of option 4 are:

1 in line with SME interpretation concept (per interpretation certain attributes are relevant); I think it is a logical extension
2 powerful (only limited by LISP, dialog language, etc. , not by AutoCAD)
3 can be made pretty flexible (depending on the implementation, but still ...)
4 user interface can be easily integrated with SME feature interface

After discussing this with Mark Clayton, we decided to use option 4 and to implement it the expedient way, i.e., make a (rapid, dirty) prototype version by adding new functionality to existing SME functionality. With the new SME+ system it is possible to formalize non geometric design information, which should be enough for developers of systems that reason about design information. If future users are pleased with the functionality, it might be incorporated in a new version of SME. This new version can be either in AutoLISP or in an object oriented environment (e.g., C++ of AutoCAD 13). Until then, the new SME+ system can be used in parallel with the current SME system.
The following describes how SME+ can be used, and shortly how it is implemented. The document ends with a short evaluation of the ideas behind SME and SME+.

How to use it

This section describes how SME+ can be used by the developers of conceptual models and end-users. The example model is called decomp. The files mentioned in this section are stored in `/home/users/luiten/SME`. The appendix contains some of the example files and a list of defined AutoLISP methods.
To comply better with the OO paradigm, the terms *class* and *instance* are used, in stead of *class* and *feature*.

Initializing the new system
To initialize the system type `(load "/home/users/luiten/SME/edsm")` in the AutoLISP command line. This loads all the required files: `edsm.lsp`, `eiman.lsp`, `egeometry.lsp`, and `edsmdialog.lsp`.
To use the example decomp interpretation, copy the class definition file `/home/users/luiten/SME/decomp.eiob` to your working directory, open the decomp model in the SME Interpretation Manager, and type `(load "/home/users/luiten/SME/decomp")`.
NB the file `"/home/users/luiten/SME/acad.lsp"` automatically loads all LISP files needed. This file is loaded when you start AutoCAD from `"/home/users/luiten/SME"`. Copy the contects of this file your `acad.lisp` file.

Define classes, attributes and inheritance structure
There is no user interface for this part (yet?, depends on whether people really want to use it). New classes and attributes can be defined in the `<model>.eiob` file (see, e.g., `decomp.eiob` in the appendix). A conceptual model is defined in the following syntax[1]:

```
conceptual_model    = ([class])
class               = (class_id [attribute] [parent])
attribute           = (attribute_id (type lower_bound upper_bound kind))
type                = {STRING, BOOLEAN, INTEGER, REAL, CHARACTER, class_id}
lower_bound         = number
upper_bound         = number
kind                = {ATTRIBUTE, DERIVED_ATTRIBUTE}
parent              = class_id
```

A model consists of classes. Each class has an id, a list of new attributes, and a list of parents. A class inherits attributes from its parents (which can inherit attributes from their parents). When a new attribute has the same id as an inherited attribute, the inherited attribute is overwritten with the new attribute.
An attribute has an id, a defined type, bounds, and a kind. The type of an attribute can be a basic type (STRING, BOOLEAN, INTEGER, REAL, or CHARACTER), or a reference to an instance of class_id. The bounds of an attribute define the number of values that can be defined. The lower_bound defines the minimum number, the upper_bound the maximum number. An upper_bound of 0 means (per definition) that the maximum number of values is not limited. An upper_bound not equal to 1 implies a list of values, an upper_bound of 1 implies a single value. The kind of an attribute defines whether the attribute is derived from the 3D model (`DERIVED_ATTRIBUTE`) or set by the end-user (`ATTRIBUTE`). The difference is that derived attributes cannot be set with the end-user interface (see below).
The attributes `class_id` and `instance_id` are required for each class that does not have parents (see appendix).

The defined classes can be loaded from the `<model>.eiob` file into the system by using the command: `DSMLoadClasses`. The class definitions can be stored again in that file with: `DSMWriteClasses`. The classes can also be stored in EXPRESS format in a `<model>.xpr` file: `DSMWriteEXPRESS`. (NB The EXPRESS file cannot be loaded.)

When defining classes, be sure that there is an interpretation in the parallel SME system with the same name and that the SME classes have the same ids as the new classes.

---

[1] syntax rules: ( ) = a list (also use the parentheses), [ ] = repeat contents (without brackets), { } = choose one of the items between the brackets, <variable_id> = replace the brackets and the variable_id by the value of that variable

Implement LISP methods to derive (geometric and topological) attributes
The programming user can define methods to derive geometric and topological attributes from the 3D model. The programming user creates a new file called `<model_id>.lsp` (see, e.g., `decomp.lsp` in the appendix). This file should, at least, contain the method `Derive<model_id>InstanceAttributes`, which has one argument `instance_id` (see, e.g., `DeriveDecompInstanceAttributes (instance_id)`). This method evaluates the class of the instance and executes the calculations corresponding to this class. The attribute values are set using the method: `DSMSetAttributeValue (instance_id, attribute_id, new_value)`. Be sure the new value conforms to the defined type and bounds of the defined attribute.

See appendix for a list of predefined geometric methods, used in the decomp interpretation.

Set attribute values of an instance
The end-user (e.g., the designer) uses an AutoCAD menu under `Semantics, Interpretation` to manipulate attribute values of the defined instances. In the upper part of the menu the instances are manipulated (created, deleted, selected, etc.). In the lower part the attributes of one instance are manipulated.
The upper part is nearly the same as in the current SME version, except for one new button: `Derive Attributes`. When this button is clicked, the attributes of the selected instances are derived.
The lower part is activated when the upper part is in the `Single` mode. When activated, it shows all information of the selected instance: `id`, `class`, `handle`, and attributes with values.
The `id` can be updated by typing in the `Name` field. The `class` can be updated by choosing from the popup list (this also updates the attributes and their values; if the new class has the attributes in common with the old class, the values of these attributes are copied). The `handle` of an instance can be updated by clicking the `Set Graphic <` button.
Attributes that are defined as ATTRIBUTE can be updated by selecting it and typing a new value in the `Value` field. The definition of the attribute is shown in the EXPRESS format in the `Definition` field. The kind of the attribute is shown in parentheses, in this case `(User defined)`. If the entered value does not correspond to this definition, the user is alerted and the attribute value is not changed. Type checking is limited to the basic types and to whether the instances exist in the interpretation. If multiple values are entered for a single value attribute, only the first value is used to set the value. Type checking does not (yet ?) include the bounds and whether an entered instance is really a 'child' of the defined class.
Attributes that are defined as DERIVED_ATTRIBUTE (`(User defined)` in the definition, the `Value` field not active) can be updated by clicking the `Derive Attributes` button in the lower part of the menu. This calculates all derived attributes of the selected instance.

The instances can be loaded into the system form a file called `<model>.edsm` (see, e.g., `decomp.edsm`) by using the command: `DSMLoadInstances`. The instances can be stored in the same file again with: `DSMWriteInstances`. The classes can also be stored in the STEP physical file format in a `<model>.spf` file by using `DSMWriteSTEPpf`. (NB The STEP physical file cannot be loaded.)

In the `<model>.edsm` file the instances are stored for use in applications that assess the design. For example, a construction scheduling application can read this file and generate a schedule using the design information. Each application can select the instances and attributes it needs. The instances are stored in the following format:

```
model              = ([instance])
instance           = (handle [attr_value_pair])
attr_value_pair    = (attribute_id attr_value)
attr_value         = string
there are two important attr_value_pairs for each instance:
```

```
("class_id" <class_id>)
("instance_id" <instance_id>)
```

When changing the interpretation (using functionality in the current SME system), the system automatically saves the current interpretation and reads the new interpretation (if existing).

Implementation

The new system is implemented in AutoLISP. The methods are structured in three main files: `edsm.lsp`, `edsmdialog.lsp`, and `eiman.lsp`. These files are extensions of existing SME files. For deriving geometric and topological attributes special functionality is defined in `geometry.lsp` (developed by Marc Clayton) and `egeometry.lsp`. See the appendix for lists of new functionality in these files.

Relation with current version of SME

The new system runs completely in parallel with the current version of SME. This means that there are always two class definitions and two instance (or feature) definitions in core. This makes the system a bid slow every now and than, and it might not be robust in all circumstances. Be sure to save the instances regularly (by typing `DSMWriteInstances`).
An interpretation in the current SME system can be translated to the new system by using: `TranslInstances` (if corresponding new classes are defined). This creates new instances of the corresponding classes with empty attributes.

Evaluation

In the current version of SME you have to define classes and instances for each interpretation (or application) you want to assess. With the opportunity to define attributes and to use class inheritance, it might be possible to use only one classification of the geometric objects. Each interpretation can add the attribute definitions it needs and it can neglect the attributes and instances it does not need. The only thing that changes when the end-user selects an interpretation, is which methods are used to derive attribute values.
Advantages of this approach are: only one classification necessary, the model is flexible to adding new applications, and the designer does not have to know what applications will make use of the symbolic model.
A disadvantage of this approach is that you have to agree upon one conceptual model (to a certain level).

Appendix

Example file: decomp.eiob (class definitions)

```
(
("product" (("class_id" ("STRING" "1" "1" "ATTRIBUTE")) ("instance_id"
("STRING" "1" "1" "ATTRIBUTE")) ("decomposes_in" ("product" "0" "0"
"DERIVED_ATTRIBUTE"))) nil)
("spatial_element" (("length" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")) ("width"
("REAL" "1" "1" "DERIVED_ATTRIBUTE")) ("height" ("REAL" "1" "1"
"DERIVED_ATTRIBUTE")) ("volume" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")))
("product"))
("building" nil ("spatial_element"))
("block" nil ("spatial_element"))
("floor" nil ("spatial_element"))
("zone" nil ("spatial_element"))
("structural_element" (("supports" ("product" "0" "0" "DERIVED_ATTRIBUTE")))
("product"))
("beam" (("length" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")) ("reinf_ratio"
("INTEGER" "1" "1" "ATTRIBUTE")) ("suppliers" ("STRING" "0" "0" "ATTRIBUTE"))
("material" ("STRING" "1" "1" "ATTRIBUTE")) ("section_type" ("CHARACTER" "1"
"1" "ATTRIBUTE"))) ("structural_element"))
("column" (("length" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")))
("structural_element"))
("slab" (("length" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")) ("width" ("REAL" "1"
"1" "DERIVED_ATTRIBUTE")) ("height" ("REAL" "1" "1" "DERIVED_ATTRIBUTE")))
("structural_element"))
)
```

Example file: decomp.xpr (class definitions in EXPRESS format)

```
SCHEMA decomp

  ENTITY product;
      class_id: STRING;
      instance_id: STRING;
    DERIVE
      decomposes_in: LIST [0:0] OF product = .... ;
  END_ENTITY;

  ENTITY spatial_element
    SUBTYPE OF (product);
    DERIVE
      length: REAL = .... ;
      width: REAL = .... ;
      height: REAL = .... ;
      volume: REAL = .... ;
  END_ENTITY;

  ENTITY building
    SUBTYPE OF (spatial_element);
    DERIVE
  END_ENTITY;

  ENTITY block
    SUBTYPE OF (spatial_element);
    DERIVE
```

```
  END_ENTITY;

  ENTITY floor
    SUBTYPE OF (spatial_element);
    DERIVE
  END_ENTITY;

  ENTITY zone
    SUBTYPE OF (spatial_element);
    DERIVE
  END_ENTITY;

  ENTITY structural_element
    SUBTYPE OF (product);
    DERIVE
      supports: LIST [0:0] OF product = .... ;
  END_ENTITY;

  ENTITY beam
    SUBTYPE OF (structural_element);
      reinf_ratio: INTEGER;
      suppliers: LIST [0:0] OF STRING;
      material: STRING;
      section_type: CHARACTER;
    DERIVE
      length: REAL = .... ;
  END_ENTITY;

  ENTITY column
    SUBTYPE OF (structural_element);
    DERIVE
      length: REAL = .... ;
  END_ENTITY;

  ENTITY slab
    SUBTYPE OF (structural_element);
    DERIVE
      length: REAL = .... ;
      width: REAL = .... ;
      height: REAL = .... ;
  END_ENTITY;

END_SCHEMA;
```

Example file: decomp.lsp (to derive instance attribute values)

```
(defun DeriveDecompInstanceAttributes (instance_id /
                                       class_id lwh)
; derive the form attributes for instance
  (setq class_id (DSMGetInstanceClassId instance_id))
  (cond
    ((equal class_id "building")
      (setq lwh (DSMSolidLWH (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "length" (car lwh))
      (DSMSetAttributeValue instance_id "width" (cadr lwh))
      (DSMSetAttributeValue instance_id "height" (caddr lwh))
      (DSMSetAttributeValue instance_id "volume"
                    (DSMSolidVolume (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "decomposes_in"
```

```
                    (DSMEnclosedBy instance_id
                          (DSMFilterClass "block" instances) 0.9))
      (if (= (DSMGetAttributeValue instance_id "decomposes_in") nil)
        (progn
          (DSMSetAttributeValue instance_id "decomposes_in"
                    (DSMEnclosedBy instance_id
                          (DSMFilterClass "floor" instances) 0.9))
          (if (= (DSMGetAttributeValue instance_id "decomposes_in") nil)
            (progn
              (DSMSetAttributeValue instance_id "decomposes_in"
                        (DSMEnclosedBy instance_id
                              (DSMFilterClass "zone" instances) 0.9))
              (if (= (DSMGetAttributeValue instance_id "decomposes_in") nil)
                (DSMSetAttributeValue instance_id "decomposes_in"
                    (DSMEnclosedBy instance_id
                       (DSMFilterClasses (list "column" "beam" "slab")
                                         instances) 0.9))
              )
            )
          )
        )
      )
    )
etc.
etc.
    ((equal class_id "zone")
      (setq lwh (DSMSolidLWH (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "length" (car lwh))
      (DSMSetAttributeValue instance_id "width" (cadr lwh))
      (DSMSetAttributeValue instance_id "height" (caddr lwh))
      (DSMSetAttributeValue instance_id "volume"
                    (DSMSolidVolume (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "decomposes_in"
                    (DSMEnclosedBy instance_id
                       (DSMFilterClasses (list "column" "beam" "slab")
                                         instances) 0.9))
    )
    ((equal class_id "column")
      (setq lwh (DSMSolidLWH (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "length" (caddr lwh))
      (DSMSetAttributeValue instance_id "supports"
                    (DSMSupports instance_id
                       (DSMFilterClasses (list "column" "beam" "slab")
                                         instances) 0.9))
    )
    ((equal class_id "beam")
      (setq lwh (DSMSolidLWH (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "length" (car lwh))
      (DSMSetAttributeValue instance_id "supports"
                    (DSMSupports instance_id
                       (DSMFilterClasses (list "column" "beam" "slab")
                                         instances) 0.9))
    )
    ((equal class_id "slab")
      (setq lwh (DSMSolidLWH (DSMGetInstanceHandle instance_id)))
      (DSMSetAttributeValue instance_id "length" (car lwh))
      (DSMSetAttributeValue instance_id "width" (cadr lwh))
      (DSMSetAttributeValue instance_id "height" (caddr lwh))
```

```
    )
  )
  nil
)
```

Example file: decomp.edsm (user defined instances)

```
(
("7DF" (("class_id" "beam") ("instance_id" "b12") ("decomposes_in" nil)
("length" 235.0) ("supports" nil) ("reinf_ratio" nil) ("suppliers" nil)
("material" nil) ("section_type" nil)))
("7E3" (("class_id" "beam") ("instance_id" "b11") ("decomposes_in" nil)
("length" 235.0) ("supports" nil) ("reinf_ratio" nil) ("suppliers" nil)
("material" "concrete") ("section_type" nil)))
("6D5" (("class_id" "column") ("instance_id" "c13") ("decomposes_in" nil)
("length" 100.0) ("supports" ("b12"))))
("6D8" (("class_id" "column") ("instance_id" "c12") ("decomposes_in" nil)
("length" 100.0) ("supports" ("b12" "b11"))))
("6D9" (("class_id" "column") ("instance_id" "c11") ("decomposes_in" nil)
("length" 100.0) ("supports" ("b11"))))
("3AA" (("class_id" "floor") ("instance_id" "floor21") ("decomposes_in" nil)
("length" 500.0) ("width" 500.0) ("height" 100.0) ("volume" 2.46094e+07)))
("3AC" (("class_id" "floor") ("instance_id" "floor12") ("decomposes_in" nil)
("length" 500.0) ("width" 500.0) ("height" 100.0) ("volume" 2.46094e+07)))
("3AB" (("class_id" "floor") ("instance_id" "floor11") ("decomposes_in" ("b12"
"b11" "c13" "c12" "c11" "c14" "c15" "c16" "b13" "b14")) ("length" 500.0)
("width" 500.0) ("height" 100.0) ("volume" 2.46094e+07)))
("3A6" (("class_id" "block") ("instance_id" "block2") ("decomposes_in"
("floor21")) ("length" 500.0) ("width" 500.0) ("height" 100.0) ("volume"
2.46094e+07)))
("1C8" (("class_id" "block") ("instance_id" "block1") ("decomposes_in"
("floor12" "floor11")) ("length" 500.0) ("width" 500.0) ("height" 200.0)
("volume" 5e+07)))
("34C" (("class_id" "building") ("instance_id" "building1") ("decomposes_in"
("block2" "block1")) ("length" 1000.0) ("width" 500.0) ("height" 200.0)
("volume" 1e+08)))
("1113" (("class_id" "column") ("instance_id" "c14") ("decomposes_in" nil)
("supports" nil) ("length" nil)))
("1114" (("class_id" "column") ("instance_id" "c15") ("decomposes_in" nil)
("supports" ("b13" "b14")) ("length" 100.0)))
("1115" (("class_id" "column") ("instance_id" "c16") ("decomposes_in" nil)
("supports" nil) ("length" nil)))
("1116" (("class_id" "beam") ("instance_id" "b13") ("decomposes_in" nil)
("supports" nil) ("length" nil) ("reinf_ratio" nil) ("suppliers" nil)
("material" nil) ("section_type" nil)))
("1117" (("class_id" "beam") ("instance_id" "b14") ("decomposes_in" nil)
("supports" nil) ("length" nil) ("reinf_ratio" nil) ("suppliers" nil)
("material" nil) ("section_type" nil)))
)
```

Example file: decomp.spf (instances in STEP physical file format)

```
HEADER;

blablablabla

END_SEC;

DATA;
```

```
#1=beam(beam,b12,(),235.0,(),nil,(),nil,nil);
#2=beam(beam,b11,(),235.0,(),nil,(),concrete,nil);
#3=column(column,c13,(),100.0,(#1));
#4=column(column,c12,(),100.0,(#1,#2));
#5=column(column,c11,(),100.0,(#2));
#6=floor(floor,floor21,(),500.0,500.0,100.0,2.46094e+07);
#7=floor(floor,floor12,(),500.0,500.0,100.0,2.46094e+07);
#8=floor(floor,floor11,(#1,#2,#3,#4,#5,#12,#13,#14,#15,#16),500.0,500.0,100.0,
2.46094e+07);
#9=block(block,block2,(#6),500.0,500.0,100.0,2.46094e+07);
#10=block(block,block1,(#7,#8),500.0,500.0,200.0,5e+07);
#11=building(building,building1,(#9,#10),1000.0,500.0,200.0,1e+08);
#12=column(column,c14,(),(),nil);
#13=column(column,c15,(),(#nil,#nil),100.0);
#14=column(column,c16,(),(),nil);
#15=beam(beam,b13,(),(),nil,nil,(),nil,nil);
#16=beam(beam,b14,(),(),nil,nil,(),nil,nil);

END_SEC;
```

Implemented methods in file edsm.lsp

```
; **** file write load defuns ****
(defun DSMWriteClasses ( /
(defun DSMLoadClasses ( /
(defun DSMWriteEXPRESS ( /
(defun DSMWriteInstances ( /
(defun DSMLoadInstances ( /
(defun DSMWriteSTEPpf ( /
; **** class defuns ****
(defun DSMCreateNewClass (class_id  first_parent_class_id /
(defun DSMDeleteClass (class_id / )
(defun DSMRenameClass (old_class_id new_class_id /
(defun DSMCreateNewAttribute (class_id attr_id attr_type lowerbound upperbound
attr_kind /
(defun DSMDeleteAttribute (class_id attr_id /
(defun DSMAddClassParent (class_id new_parent_id /
(defun DSMDeleteClassParent (class_id parent_id /
(defun DSMGetClassId (class)
(defun DSMGetDefinedClassAttributes (class_id /
(defun DSMGetClassAttributes (class_id /
(defun DSMGetClassParentIds (class_id /
(defun DSMGetClassWithId (class_id /
(defun DSMGetClassChildrenIds (class_id /
(defun DSMGetInstancesOfClass (class_id /
(defun DSMUpdateAllInstances ( /
; **** instance defuns ****
(defun DSMCreateNewInstance (instance_id handle class_id /
(defun NotExpDSMCreateNewInstance (instance_id handle class_id /
(defun DSMDeleteInstance (instance_id /
(defun DSMSetInstanceId (old_instance_id new_instance_id /
(defun DSMSetInstanceHandle (instance_id handle /
(defun DSMSetInstanceClass (instance_id class_id /
(defun DSMGetInstanceId (instance)
(defun DSMGetInstanceHandle (instance_id /
(defun DSMGetInstanceClassId (instance_id /
(defun DSMGetInstanceAttributes (instance_id /
(defun DSMGetInstanceWithId (instance_id /
```

```
(defun DSMGetInstanceWithHandle (handle)
(defun DSMUpdateInstanceAttributes (instance_id /
; **** attribute defuns ****
(defun DSMGetAttributeId (attribute)
(defun DSMGetAttributeDefinition (class_id attribute_id)
(defun DSMGetAttributeType (attribute_def)
(defun DSMGetAttributeLowerbound (attribute_def)
(defun DSMGetAttributeUpperbound (attribute_def)
(defun DSMGetAttributeKind (attribute_def)
(defun DSMIsUserDefinedAttribute (attribute_def)
(defun DSMIsDerivedAttribute (attribute_def)
(defun DSMAttributeIsList (attribute_def)
(defun DSMAttributeIsBasic (attribute_def /
(defun DSMGetAttributeValueString (instance_id attribute_id /
(defun DSMSingleAttributeValueToString (value attr_type /
(defun DSMSingleAttributeValueFromString (value_string attr_type /
(defun DSMGetAttributeValue (instance_id attribute_id /
(defun DSMSetAttributeValue (instance_id attribute_id new_value /
(defun DSMValueTypeOK (attribute_def attribute_value /
(defun DSMSingleValueTypeOK (attr_type single_value /
(defun DSMValueTypeOKString (instance_id attribute_id new_value_string /
; **** General Attribute Calculation defuns ****
(defun DeriveAttributes ( /
(defun DeriveInstanceAttributes (instance_id /
; **** list defuns ****
(defun remove (item alist /
(defun islast (item alist)
(defun hasitem (item alist)
; **** string defun ****
(defun prepare_string (new_string /
; **** temporary defuns needed for transition from old to new system ****
(defun translinstances ( /
(defun GetFeature (instance_id /
(defun cInstance ()
```

Implemented methods in file egeometry.lsp

```
(defun DSMFilterClass (class_id instances /
(defun DSMFilterClasses (class_ids instances /
(defun DSMInterference (instance_id instance_ids /
(defun DSMEnclosedBy (instance_id instance_ids accuracy /
(defun DSMSupports (instance_id instance_ids accuracy /
(defun DSMSolidCentroidPoint (handle /
(defun DSMIsAbove (handle other_handle /
```