Three Integrated Tools for Designing and Prototyping Object-Oriented Enterprise Models[*]

Ulrich Frank, Stefan Klein

Gesellschaft für Mathematik und Datenverarbeitung
Schloß Birlinghoven
Postfach 13 16
5205 Sankt Augustin 1
Germany

GMD research paper No. 689, 1992

Abstract

The paper presents an integrated environment for designing object-oriented enterprise models. The conceptual framework it is based on recommends a multi-perspective approach. For this purpose three main views on the enterprise are proposed: a strategic view, an organizational/operational view and an information system view. The environment that is introduced is intended to encourage the design of enterprise models on these three levels as well as interconnecting them. It features three tools each of which is related to one or more main levels of abstraction. The *Object Model Designer* guides conceptualization of an enterprise wide object model. Object models are represented using a graphical notation that is completed by a structured description of classes and associations. In order to facilitate user feedback fast prototyping is supported by generating code from class descriptions. The object model´s implementation demonstrates the automatic control of semantically rich integrity constrains as well as the benefits of inter-application communication using domain concepts as common references rather than technical ones. The *Office Procedure Designer* guides the description of office procedures which are conceptualized as ordered graphs of activity blocks. It provides means to analyze the effectiveness of business procedures and generates prototypical user-interfaces as a representation of a virtual procedure document. The *Value Chain Designer* is mainly a tool for representing and documenting business activities and for strategic scrutiny. Based on Porter´s value chain concept it provides a general framework as well as analytical categories to segment business units and to establish relations between different activities. It thus supports modelling of alternative future strategies.

Beside a detailed description of the tools listed above the paper presents a conceptual framework for the design of multi-perspective enterprise model. Furthermore it is demonstrated how the tools and the related methods interact during analysis and design and how the partial models managed by the tools are integrated.

Keywords

Object-Oriented Analysis, Object-Oriented Design, Enterprise Modelling, Business Planning, Fast Prototyping, Integrated Information Systems, Reusability

1 Introduction

Designing, implementing and using corporate information systems face numerous challenges, e.g. frictions between the different stages of system life-cycles should be avoided, the software architecture should support system adaptability, communication between different applications (within one organization as well as inter-organizational) should be possible on a high level of semantics, costs for development and maintenance should be reduced. Integration as well as reusability seem to be attractive orientations to meet these challenges. Both of them require comprehensive models of the enterprise in general and of its information system in particular.

For a number of reasons an object-oriented approach seems to be very suitable to build such models. The project "Computer Integrated Enterprise" [Frank/Klein 1992] that had been started in 1990 at the German National Research for Computer Science is dedicated to this subject. This paper reports on some of the results that have been accomplished so far. It presents a methodology as well as a set of related tools which are intended to support the design of enterprise models - particularly by

- providing a representation of organizations that is illustrative for business people and takes into account the requirements of object-oriented analysis and design at the same time.
- supporting identification, specification and refinement of objects (classes).
- contributing to strategic planning of the business and the information system.
- providing means to analyze and refine the effectiveness of business procedures.
- conveniently modelling and prototyping user-interfaces.

Generic enterprise models that fulfil the requirements of a wide range of firms are an attractive research vision. It is however not possible to develop such models from scratch. You have to start with one enterprise of a particular domain. The domain we started with is car insurance within an insurance company. The examples given below are taken from this domain.

2 Conceptual Framework

While the notion of enterprise models becomes more and more popular - within the research community (see for instance [Pröfrock et al. 1989], or [ESPRIT 1991]) as well as in the area of commercial software development and information system planning

([IBM], [Katz 1990]) there is no detailed consensus on how an enterprise model should look like. Enterprise models are supposed to provide a suitable foundation for integrating information systems.

2.1 Dimensions of Integration

Within the context of information systems the term integration is usually related to the different components of the system. Although this is an important issue there are other dimension of integration which should be taken into account as well:

- integrating the different phases of the software life-cycle
- integrating the different roles and perspectives of those who analyze, design and use an information system
- integrating the information system (and its development) with the organization (and its development)

Integration implies communication. For components to be able to communicate there has to be a *common semantic reference system*. In other words: they need to have corresponding interpretations of the symbols they interchange as well as common unique names for these interpretations. Data types, functions of an operating system or relations within a database are examples for such reference systems. The more semantics is incorporated in the concepts that can be referred to the higher is the level of integration. The amount of semantics itself depends on the number of permitted interpretations. A data type like an integer can be interpreted in numerous different ways - depending on what real world entity it represents. A concept however that directly represents a real world entity reduces the set of possible interpretations. Is there any indication for the appropriate amount of semantics? It seems to be desirable to provide concepts that incorporate enough semantics not to bother any of the involved components with the need to reconstruct meaning for further processing. For instance: defining a concept “account” rather than only providing more general concepts that could be used to implement an account in a convenient way. If you then include a certain graphical representation of an account into a document the document processor should know the semantics of an account - which would improve the chances for powerful interpretations. Considering the need for flexibility and reusability however recommends to also provide more general concepts that allow for specialization.

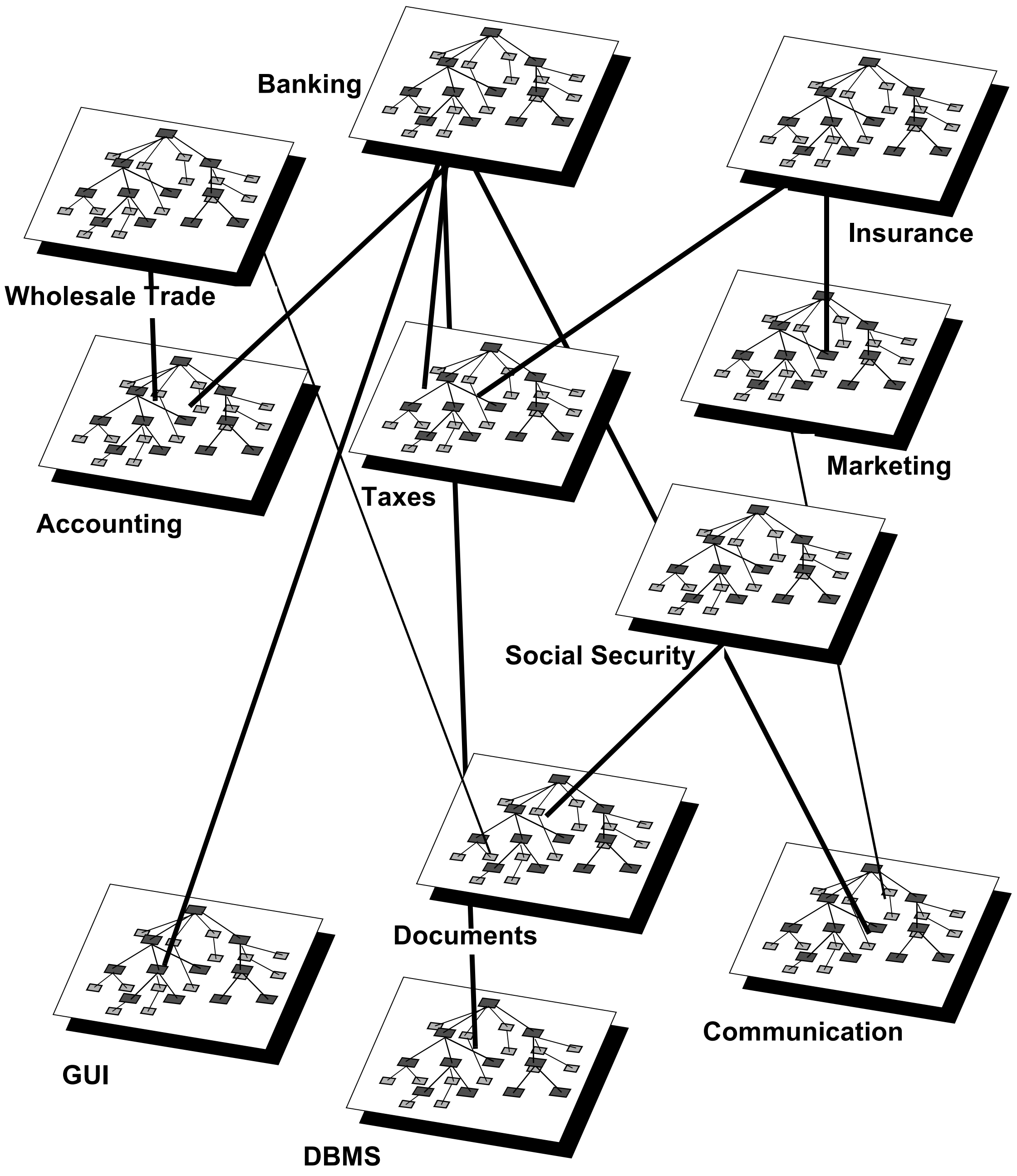


Figure 1. Visualizing the Vision: Generic Object Models as Promoters of both high Level Integration and Reusability

Common semantic reference systems are not only a prerequisite for technical integration. In order to overcome the frictions between the different phases of the software lifecycle it is desirable to use the same or at least similar concepts from analysis to implementation. Mediating between different human perceptions of reality also requires common reference systems or in other words: a common universe of discourse. Different from formal systems a certain amount of ambiguity is not only tolerable but sometimes even helpful to cope with complexity.

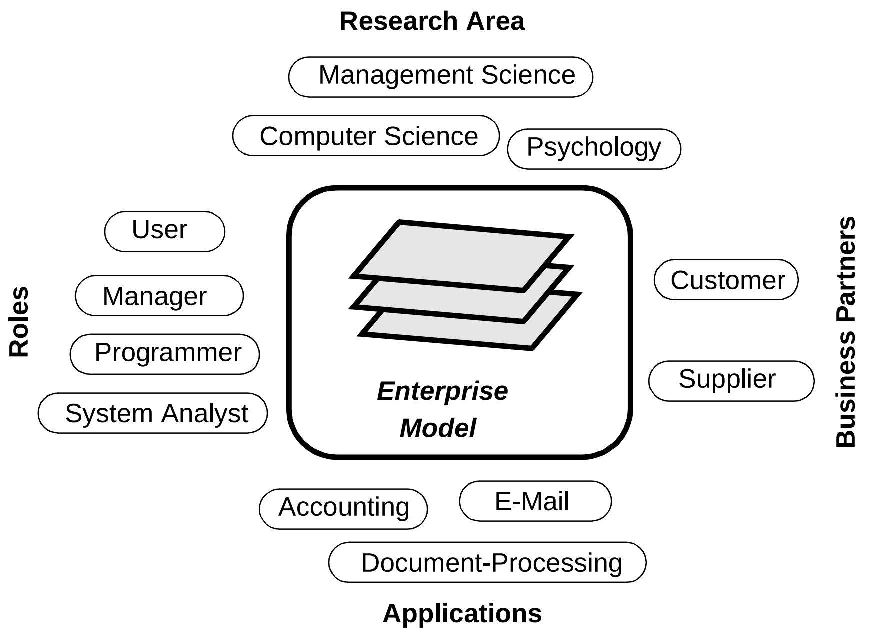


Figure 2. Dimensions of Integration fostered by Enterprise Models

2.2 Levels of Abstraction

What are the implications of these thoughts for the design of enterprise models? First: for enterprise models to serve as promoters of integration they need to be comprehensive. That means they have to provide a description of reality, "which correspond directly and naturally to our own conceptualizations" [Levesque/Myloupolos 1984]. Second: since there are different conzeptualisations as well as different requirements on modelling (for instance between business analysis and software development), an enterprise model should represent reality on different levels of abstraction. Considering the numerous views/conceptualizations (see for instance [ESPRIT 1991], [Zachman 1987]) one can think of it is necessary to make a suitable selection. We decided on three main levels of abstraction:

- a *strategic view*
- an *operational/organizational view*
- an *information system view*

Considering the complexity of the overall design process it is important to provide a tool that supports a systematic approach. Such a tool should enforce a certain methodology for object-oriented analysis and design. It should prevent the model from becoming inconsistent by checking for ambiguity and contradictions. Participation requires a substantial understanding of how the system will look like. Therefore the tool should allow for fast prototyping. On the strategic as well as on the organizational level there may be concepts which cannot be formalized although they can be comprehensively described. In order to link them to related concepts it is desirable that the tool includes some kind of hypertext-features.

It is often argued that an information system should be adapted to the organization, not the other way around. While such a request seems reasonable at first sight (specially when you consider how restrictive today´s software sometimes is) it is not completely convincing. This is for two reasons. First: a business firm´s actual organization does not have to be efficient. Adapting an information system to it means to put effort in reconstructing inefficient structures and procedures. Second: in order to exploit the potential of information systems it can be suitable to rearrange an organization that had been efficient on a lower level of automation. Like Savage [1990, p. xii] assumes: “Could it be that we are putting fifth generation technology in second generation organizations?” Taking these thoughts into account recommends mutual adaptation of organization and information technology. To reduce the complexity of this task it is desirable that a tool supports the evaluation of organizational alternatives.

The environment we developed is a first attempt to fulfil the requirements listed above. Currently it consists of three tools which are enhanced by a hypertext-system. Each of the tools covers at least one of the three main levels of abstraction (see fig. 2). All the tools have been written in Smalltalk-80 within the Objectworks® environment. Schultz [1990] has proposed an alternative concept for an object-oriented information systems architecture modeler (ZOOM) which is implemented in Smalltalk-V. ZOOM is based on Zachman's concept for an IS architecture [1987] and "... is a framework of loosely coupled cells organized to communicate an information systems architecture". In contrast to ZOOM, the environment we developed is not restricted to information system architecture but puts special emphasis on the design of multi-perspective enterprise models.

All the tools of the integrated design environment (Object Model Designer, Office Procedure Designer, Value Chain Designer) have been written in Smalltalk-80 (Vers. 4.0) within the Objectworks® environment. High productivity could be achieved by using additional class libraries:

- ObjectKit®, a Smalltalk class library that contains classes which allow to make objects persistent.
- Smalltalk Frame Kit (already mentioned above)
- Tigre®, an interactive interface-builder that also includes some database features.

- Analyst®, a desktop publishing system that supports hyper-documents.
- NEDT®, a class library that supports the development of customized graphical editors.

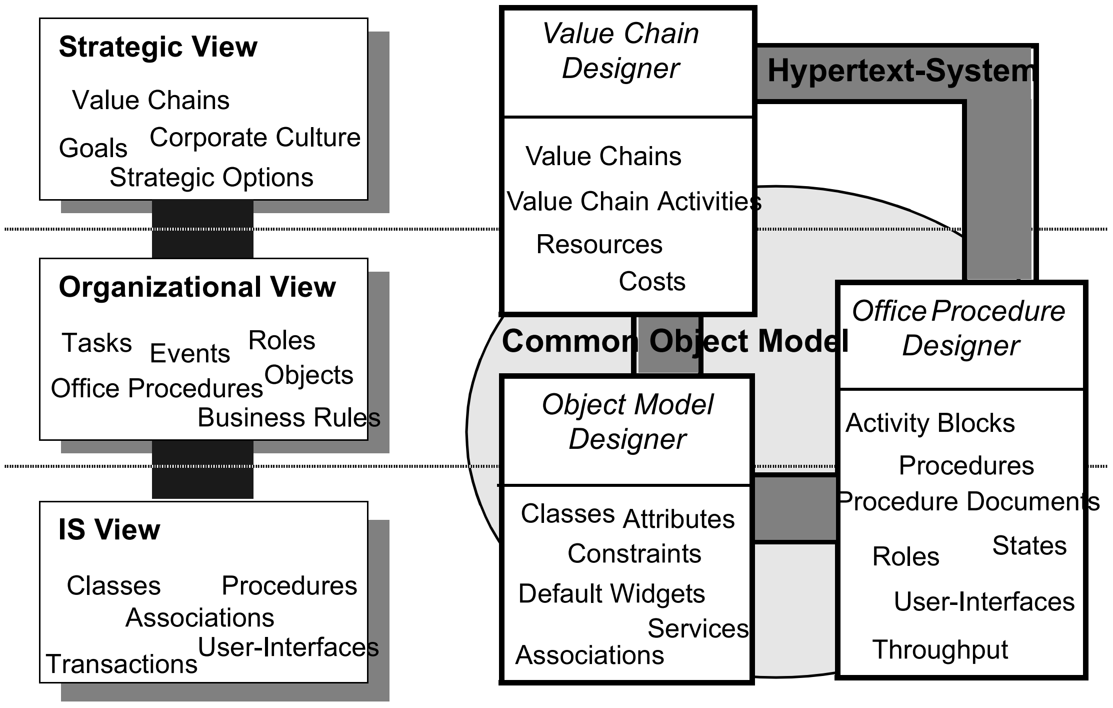


Figure 3. Tools of the design environment and their relation to different views of the enterprise

3 Developing a Static Representation of the Enterprise: The Object Model Designer

An object model is the core of an enterprise model. The concepts it describes can be referred to by other particular models. An object model consists of classes and relationships between them. While it is often argued that objects offer a natural way of describing reality it cannot be neglected that the notion of an object within a conceptual model has to be oriented towards a certain formal structure - no matter how people prefer to describe entities they perceive. The *Object Model Designer* is intended to provide analysts and users with a suitable and comprehensive concept of an object and guide the mapping of real world domains to object models.

3.1 Object Semantics

While from a (re-)using programmer´s point of view it is sufficient to describe an object solely by the services it provides analysis and design require a more detailed view. Our concept of an object model is inspired by Booch [1990] and Rumbaugh et.al. [1991]. According to them (and most other authors) an object is modelled by describing attributes and services. Additionally we use the category constraints. Furthermore it is possible to establish numerous relationships between objects/classes. Depending on the features of the implementation language it can be important to make a difference between class and instance level (like in Smalltalk). We found however that it is acceptable to neglect the class level. During analysis and design you primarily focus on the features of an instance-object (not to be confused with a particular instance, see 3.3). Furthermore class level specifications are hard to transform with languages that do not regard classes as objects. An object/class is modelled by describing attributes, services and a *default view*. Additionally we use the category *constraints*.

An attribute is regarded as an object that is encapsulated within the object. We do not allow attributes - like Coad/Yourdan [1991] - to only hold references to external objects that have an existence of there own in the object space. An attribute is described by the following aspects:

- *name*
- *class*
- *cardinality*
- *default value*
- *history*
- *authorization*

Specifying an attribute´s class is a prerequisite for typing. The OMD also allows to paste services from an attribute´s class into the classes interface. For instance: class Employee may contain an attribute of class BirthDate, the service age may now be generated for Employees, too. If name conflicts occur, the user will be notified.

Cardinality has to be defined in min., max.-notation. For instance: a costumer´s telephone number may have cardinality 0,*. Specifying a default value may allow for generating an appropriate initialization method. If history is set to true every update of the attribute has to be recorded somehow. The authorization to access an attribute can be separately described for get- and put-access where each access type can be assigned one of three authorization numbers: private (0), protected (1) or public (2). It is not possible to define write-permission to be greater than read-permission. Implementation of authorization levels certainly depends on the features of the implementation language. Deviating from a solution like it is featured by C++ other objects can access attributes only via services. The OMD generates put- and get- services for each attribute. Visibility of the services depends on the authorization level (see also below). The current version only generates Smalltalk-like interfaces. For instance: definition of the attribute “name” would result in the services name and name:.

Services are characterized by their interface, where each attribute is defined by its class, and a natural language description of the function they fulfil. Furthermore a precondition and a postcondition can be specified. If the service returns an object, this object´s class can be specified. While attribute and service descriptions already include constraints (like attribute-classes, pre- and postconditions) there may be other object-constraints that cannot be assigned to just one attribute or service. This is the case for integrity rules which interrelate different attributes or services. We differentiate between two types of constraints: *guards* and *triggers*. A guard is a constraint that prevents the object from merging into a certain state. For instance: the resale-price assigned to a product should never be less than the purchase-price. A trigger on the other hand prevents an information system from becoming inconsistent by not reacting if some condition is fulfilled. For instance: If a customer who holds a car insurance policy has been driving without an accident for more than three years and has not been assigned the highest claims bonus yet, his claim bonus has to be increased.

In order to allow for generating prototypical user-interfaces it is possible to assign a *default view* (a collection of widgets) to each class. One can also define labels that are to be presented with the widgets. Additionally the size of the widgets can be pre-specified. Since attributes are characterized by their class a default view of a newly defined class can be generated by using its attibutes' default views. This approach is a first attempt to deal with the complexity of modelling user interaction. It cannot be completely satisfactory: the way a value of a certain class is presented to the user often is not unique but varies with the context of interaction. For instance: you can display a name using a scrollable text view, a listbox etc.

3.2 Associations between Objects

Objects within an information system are interrelated in various ways: objects may use services from other objects, they may be composed of other objects, their existence may depend on other objects etc. Taking such associations/relationships into account is crucial for maintaining the integrity of an IS. Therefore they are commonly regarded as an essential part of an object model. There is however no consensus on how to describe them. Booch [1990] claims that it is sufficient to use only two sorts of relationships between objects: using and containing. While a containing relationship describes aggregation, a using relationship means that the related objects may interchange messages. Rumbaugh et.al. [1991] do not suggest a limited set of associations. Instead they allow the designer to define his own associations. We agree with Booch that aggregation and interaction relationships are probably sufficient to classify most relationships. Thinking of implementation it is also a good idea to limit the scope to a few well analyzed concepts. But in order to design illustrative as well as semantically rich domain level models we prefer associations which may include domain specific semantics and which are labeled with names that are known in the application domain. Having a wider range of different types of associations allows to define views on aspects of the object model. If somebody is interested in an organizational schema one could filter all classes which are associated via “is subordinated” or “is superior”. A relationship may have features that cannot solely be assigned to any of the connected objects. For instance: information on the relationship *attendsTo* between an insurance agent and an insured person like “when was the relationship established?" or "where was it established?". For this reason we adopt the approach Rumbaugh et al. suggest: associations may be modelled as classes. If you do not restrict the set of allowable associations picking an association is necessarily somewhat arbitrary (which is also the case for defining classes in general). This arbitrariness can be reduced by encouraging the analyst/designer to select from a collection of previously defined associations before defining a new one. Furthermore it is possible to take advantage of inheritance.

Last but not least: at the current state of art we regard the design of an object-oriented enterprise model as an evolutionary research process. That puts emphasis on cyclic refinement of the defined concepts. In the long range there is a chance to substantially reduce the number of classes (whether they are associations or "ordinary" classes) by inductive analysis.

The permissible cardinality range of an association has to be specified in min,max-notation. Each of the involved classes has to be assigned a tuple with the minimum number of instances that have to be part of the association and the maximum number that is permitted. While binary associations are preferable it is also possible to specify ternary associations. Associations should be named like predicates to make the model more descriptive. Since the appropriate predicate name often depends on the direction, it is possible to assign an inverse name to each association. For instance: "is controlled by" would be the inverse name to "controls".

One association class is thought to provide a substitute for multiple inheritance that even offers some advantages over the original. An object can import another objects´ features by establishing a "has role"-association (which is sometimes referred to as "dynamic" or "object-level" inheritance). The roles that are assigned to a class can be ordered to resolve possible naming conflicts. If you want to describe an employee who is a manager as well as a salesperson you do not define a class "managing salesperson" that inherits from manager and salesperson. Instead employee is assigned the roles manager and salesperson in a certain order.

In order to facilitate searching for already defined classes as well as to support a systematic approach to find new classes, the classes are grouped into categories. The definition of categories should be oriented towards domain level concepts. Some of the categories we have chosen: accounting, car insurance, marketing, people, documents, devices, associations. Different from the concept used in Smalltalk a class may be assigned to more than one category.

3 Prototypical Instantiation

The OMD allows for fast prototyping and evaluation on the instance level by generating executable code. Smalltalk does not directly support important aspects of the object model: there is no strong typing, in general constraints cannot be implemented in a convenient way. Therefore we use a frame-oriented object definition language that is part of the Smalltalk Framekit (SFK), which has been developed by two colleagues at GMD [Fischer/Rostek 1992]. The conceptual description of a class can be partially transformed in SFK´s object definition language (primarily attributes together with their associated access services). SFK allows to define classes and associations by providing a partially declarative definition language. It enhances Smalltalk with strong typing. Various types of constraints can be defined as well. Compiling a class goes along with generating code for implementing guards and triggers. Figure shows an example of a frame-class definition. The code that is printed in italic has been added manually.

```
initializeSlotDescriptions
  "CarInsurancePolicy allSlotDescriptions"

(self slot: #ClaimsBonus)
  range: Bonus;
  maxCardinality: 1;
  beforeAdd: [:policy :bonus :actBonus :transact|
  (actBonus < policy maxBonus).
(self slot: #dateOfSigning)
  range: ContractTime;
  minCardinality: 1;
  maxCardinality: 1.
(self slot: #methodOfPayment)
  range: PaymentMethod.
(self slot: #paidPremium)
    range: MoneyAmount;
    minCardinality: 1.
            •
            •
            •
```

Figure 4. Partial definition of class” CarInsurancePolicy” in SFK

4 Adding Dynamic Aspects: The Office Procedure Designer

Although the object model includes a few dynamic aspects (like method pre- and post-conditions) it does not allow to model business procedures in an illustrative and comprehensive way. Object oriented design methodologies (like those promoted by Booch or Rumbaugh et al.) suggest state transition diagrams. However, for our purpose these techniques have two shortcomings. They do not provide a representation that fits the average user´s perception of a business procedure. Since state transition diagrams describe the behaviour of objects of a certain class they can hardly be used to support the design of procedures from preexisting components. Dedicated methods/tools to support design and implementation of office procedures (for instance Croft 1987, Ellis/Bernal 1982, Hogg 1987, Kreifelts 1987) seem to be more suitable. But they usually do not emphasize the integration of the dynamic model with the static object model - if they are object-oriented at all.

4.1 Conceptualization of an Office Procedure

We regard a procedure as an ordered graph of activity blocks (which I will refer to as activity as well), which can be represented as a semantically enriched Petri net. Each activity block (for similar conceptualizations compare Hogg 1987 and Lochovsky et al. 1988) is an object associated with a certain role of an employee who is responsible for this particular task. An activity block can be modelled as a procedure itself. The subject and the state of a procedure are captured in an object of class “ProcedureDocument”. In the case of concurrent processing special constraints have to be fulfilled (see below). Each activity block requires a certain state of the document as a precondition. Processing the document within an activity results in one or more new states of the document. Unlike a physical document it can be worked on at different locations at the same time - provided there are constraints which prevent inconsistent states. An object of class “ProcedureDocument” does not only offer a collection of information that has been related to its different states. Furthermore it provides a prototypical user-interface for accessing this information. A document is an illustrative as well as a powerful metaphor for describing user-interfaces: it allows to present information in a way a user is familiar with and it is more versatile than a mere window/widget-metaphor since it may incorporate a numerous pages and various links between them.

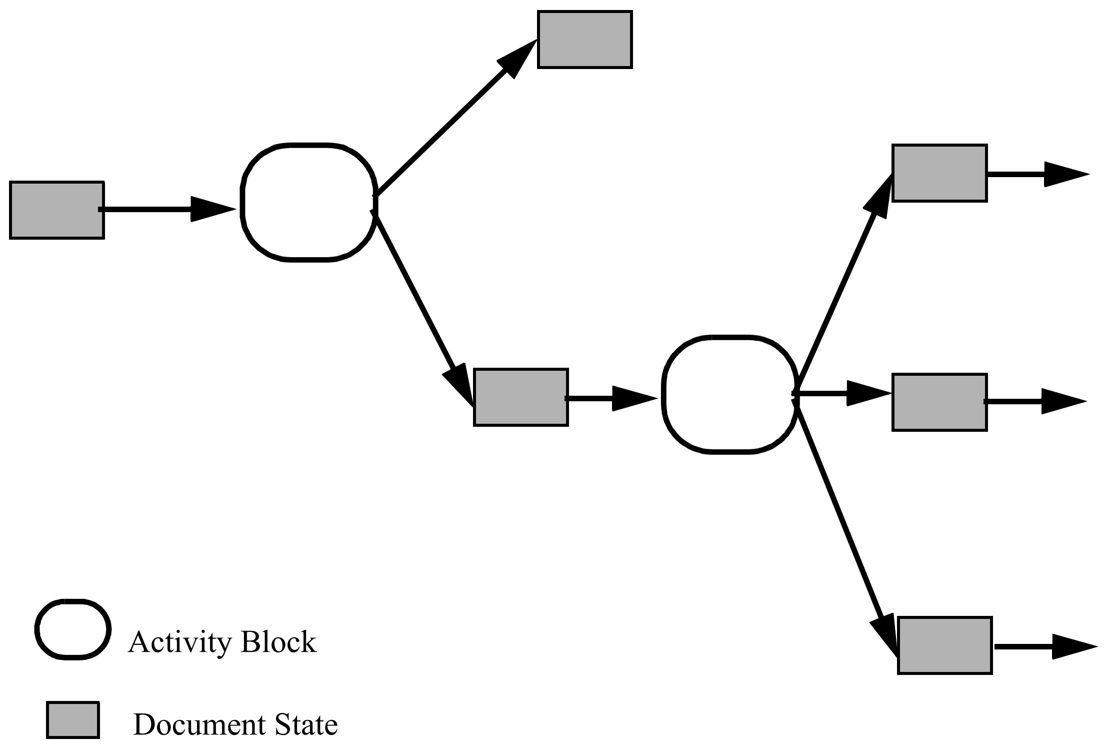


Figure 5. Conceptualization of Office Procedures

A procedure's semantics can be divided into the following categories:

General constraints

For instance: A procedure must not contain deadlocks. There must not be endless loops. There should be no task that cannot be reached by any chance.

Constraints on activities

For instance: An activity requires a certain state of a certain document type. It must produce one of a set of possible document states.

Constraints on documents

For instance: The variable parts of the document may be filled only with objects of a certain class. A part of the document that is processed within one activity may not be processed within another activity that works on the document concurrently.

Dispatching

For instance: After an activity block´s postcondition is fulfilled its successor has to be triggered, after an activity has been started, an employee who can take over the associated role has to be informed. It may be important to first check an employee's queue of activities before dispatching a new activity to him. Dispatching has to be done according to organizational rules, like: only one employee may be responsible for the whole procedure or for a collection of activities.

Exceptions
For instance: Within an activity block an inconsistent document state is detected that had been caused in a preceding activity. An employee becomes sick before completing the activity.

It is a crucial question for the design of a dynamic model to decide where to locate this knowledge. While general constraints should be checked already during the design process, all the other control knowledge can only be applied when the procedure is active. Each procedure is supervised by a procedure manager, which is an object that coordinates procedures of a certain domain. Whenever an event occurs that should trigger a procedure the procedure manager is notified. It then looks up its description of the particular type of procedure and instantiates the first activity block as well as the procedure document. Each activity block is responsible for transforming the document´s state to one of the states that are defined as postconditions. The procedure manager and the procedure document serve as "glue" to link the activity blocks. If an activity has terminated with one of its postconditional document states it notifies the procedure manager. The procedure manager looks up its list of available (human) operators and their queues of work to be done. Depending on its dispatch knowledge it will then instantiate an appropriate activity object and move it into the queue of the selected clerk. The procedure document fosters integration of the activity blocks by holding the collection of (at least partially) shared objects that need to be accessed within the procedure. When an activity is triggered it updates the procedure document by passing a collection of needed objects which have not been in the document yet. Only when the procedure has terminated regularly the procedure manager will release the involved objects (and thereby commit the final state of the procedure document).

When an activity runs into an exception (like a violated constraint or a user-generated interrupt) it will notify the procedure manager which will care for exception handling (for instance: roll back to the preceding activity block).

The Office Procedure Designer (OPD) is a tool to instruct analysis and design of office procedures according to the outlined architectural framework. For this purpose it provides the analyst/designer with an interactive template for systematically describing a procedure´s tasks. It also includes a graphical editor that allows to model office procedures in an illustrative way using a set of graphical icons (see fig. 4). The icons represent either document states or tasks:

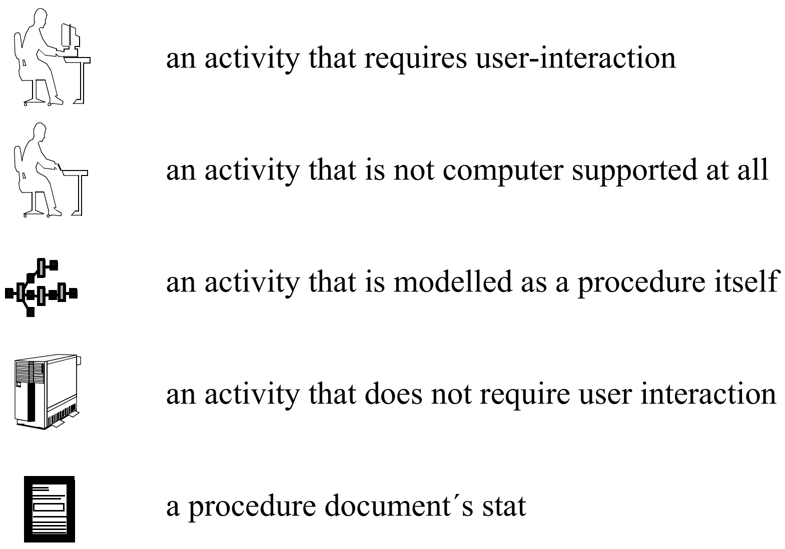

an activity that requires user-interaction
an activity that is not computer supported at all
an activity that is modelled as a procedure itself
an activity that does not require user interaction
a procedure document´s stat

Figure 6. Icons used for the graphical representation of office procedures

The OPD is not based on the waterfall-model. Instead we assume the different steps of system development to be interwoven by cyclic feedback-loops. In order to allow for a systematic description of the development approach that goes along with the OPD I will differentiate between requirements analysis, design, analysis of organizational effectiveness, and prototypical implementation.

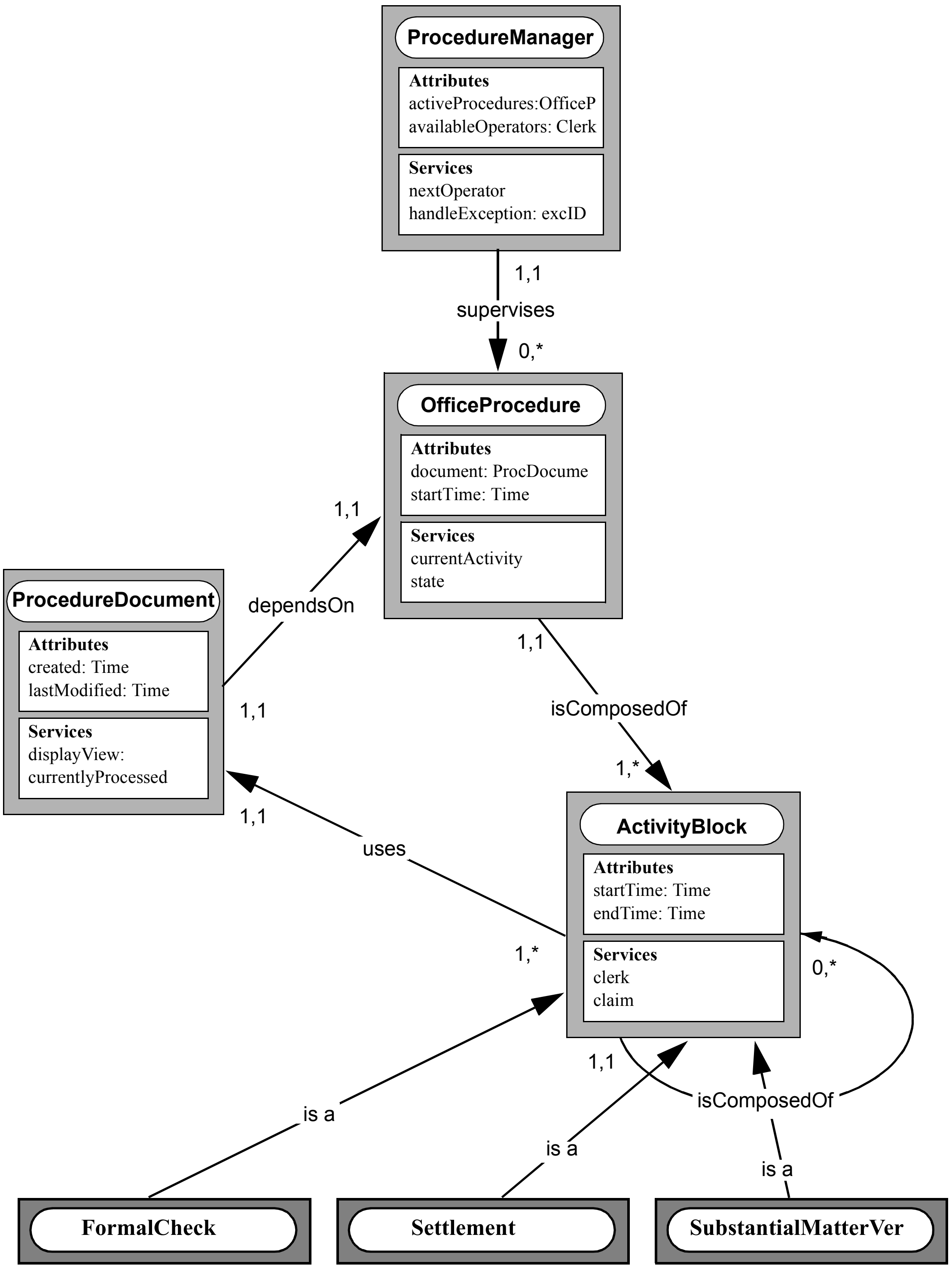


Figure 7. Partial object model of an office procedure

4.2 Requirements Analysis

Since the OPD is to support modelling of office procedures within a certain domain the first step is to collect a list of procedures which are currently established within the domain of interest. Thereby it should not matter whether the procedures are currently computer-supported at all. Modelling a particular procedure will then be done interactively by system analyst and domain expert. Starting with the event that triggers the procedure and that initializes the procedure document they describe the procedure using the available icons and connecting them by directed lines. Thereby we assume that it is intended to design a computer supported procedure - no matter how it has been organized in the past. However, there may be activities that cannot be supported by information technology. They can be characterized by an appropriate icon (see above). In this case the contents of the virtual procedure document would need to be (at least partially) hardcopied. Afterwards the changes that have been applied to it would have to be added somehow to the electronic document.

The first step of describing an office procedure as a net of activity blocks implicitly includes the definition of temporal semantics. Activity blocks can be ordered sequentially or concurrently which implies a notion of before, after and simultaneous. This allows the OPD to perform certain consistency checks. For instance: detecting deadlocks, or an activity block that produces a document state that had already been produced before (the last example is only a strong indicator of inconsistent design).

Within the next step the activities are characterized by the structured, semi-formalized description that is encouraged by the interactive-template. Thereby three main aspects are differentiated: *organizational, informational*, and *control*. Organizational aspects are expressed by assigning a responsible employee (represented by an appropriate role, like "Manager") and a department both to the whole procedure and to each activity block. Furthermore it is possible to define organizational constraints on the assignment of employees to activity blocks (like each activity has to be taken care of by only one person, or an activity block has to be supervised by the same person who supervised the preceding activity). Each activity block should also be assigned an estimated processing time. Gathering the information that is needed within an activity is crucial for capturing the essence of an activity. It is structured by offering three categories of information sources: *information system, people*, and *paper based documents*.

To specify the information that could be provided by the information system the system analyst has access to the object model. He can browse through the available classes and select the ones that are needed. If it is not a whole object of a certain class that is required services and objects of a particular class can be selected, too (see screenshot in figure 7). With each object/service a template is presented to encourage the description of additional characteristics. Among others it allows to specify the location (internal IS, external IS) of the object/service, the access permission (read, write) that is required, what exceptions could occur, and whether the information should be pasted into the procedure document. The tool will notify the system analyst if write permission is assigned to a particular attribute within two simultaneously active activity blocks. In case a service is selected that requires input, it can be specified where the input comes from (paper document, user ...). At this step it may turn out that information from the IS is needed which has not been defined in the object model yet. Looking at office procedures thereby supports finding, specifying as well as refining classes for the static part of the enterprise model.

To characterize information that is provided by people the particular person has to be specified by selecting a role from a given collection or adding it to the collection. Then you can pick one or more media that are used for communication (like phone, fax, face-to-face, letter ...). Finally there is another template that instructs what else could be specified (time estimated for delivering information, costs, exceptions, what is to be transferred to the procedure document, etc.).

There is also a collection of paper based documents (contracts, manuals, letters, memos ...) the system analyst can select from or enhance. A paper´s origin can be specified by picking from a collection of locations (departments, organizations ...). The paper document can then be further described by filling in a template that requests information on time, costs, exceptions, what is to be transferred to the procedure document, etc.

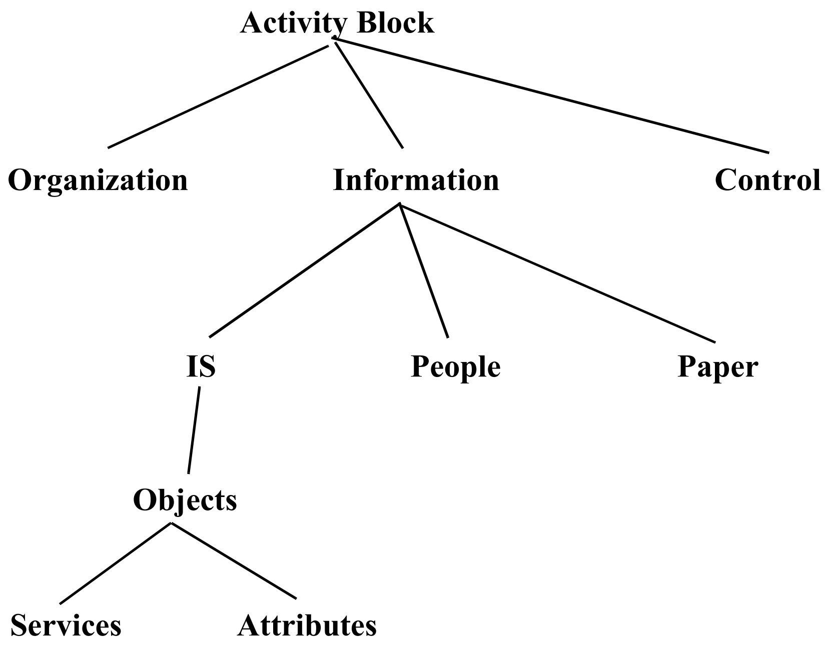


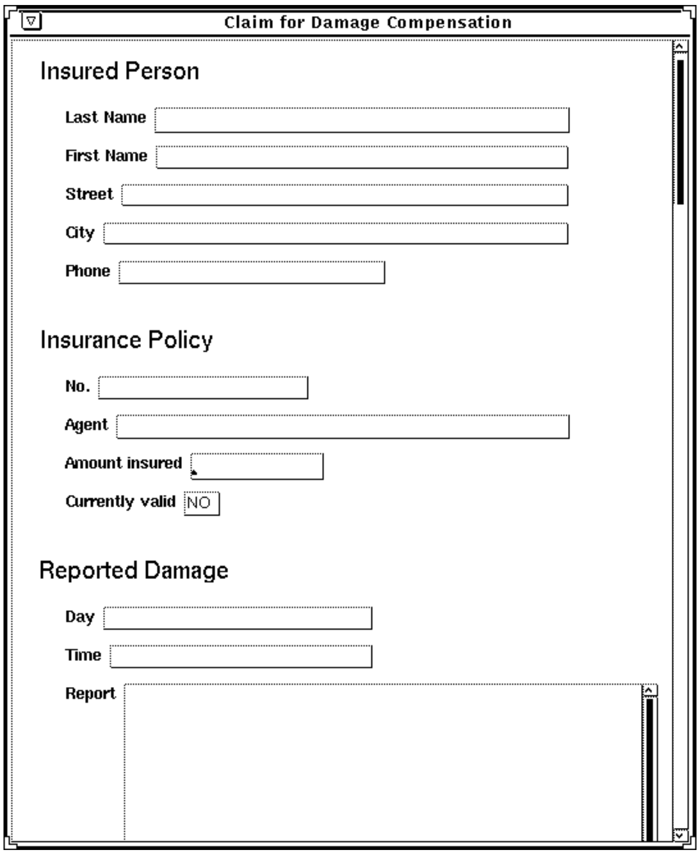
Figure 8. Partial Profile of an Activity

In order to instruct the description of the control flow within an activity block a template is presented that is generated depending on the document states that may result from the activity (see figure 7). It encourages a declarative description, which is however not required in this phase. To support system analyst and domain expert in filling the template a report that includes a description of all the required information is presented in another text view.

4.3 Design

What we called requirements analysis already results in a preliminary design of an office procedure. One activity that can be related to design is reviewing how the results of requirements analysis affected the object model. Do the proposed additional attributes or services recommend to redesign the object model? If this is the case it may be necessary to refine the procedure description as well.

The activities that constitute a procedure are preliminary named in a way domain experts are familiar with, for instance "Verification of Substantial Matter". These names have to be changed now to appropriate class names in order to allow for generating class templates.


Claim for Damage Compensation

Insured Person

Last Name
First Name
Street
City
Phone

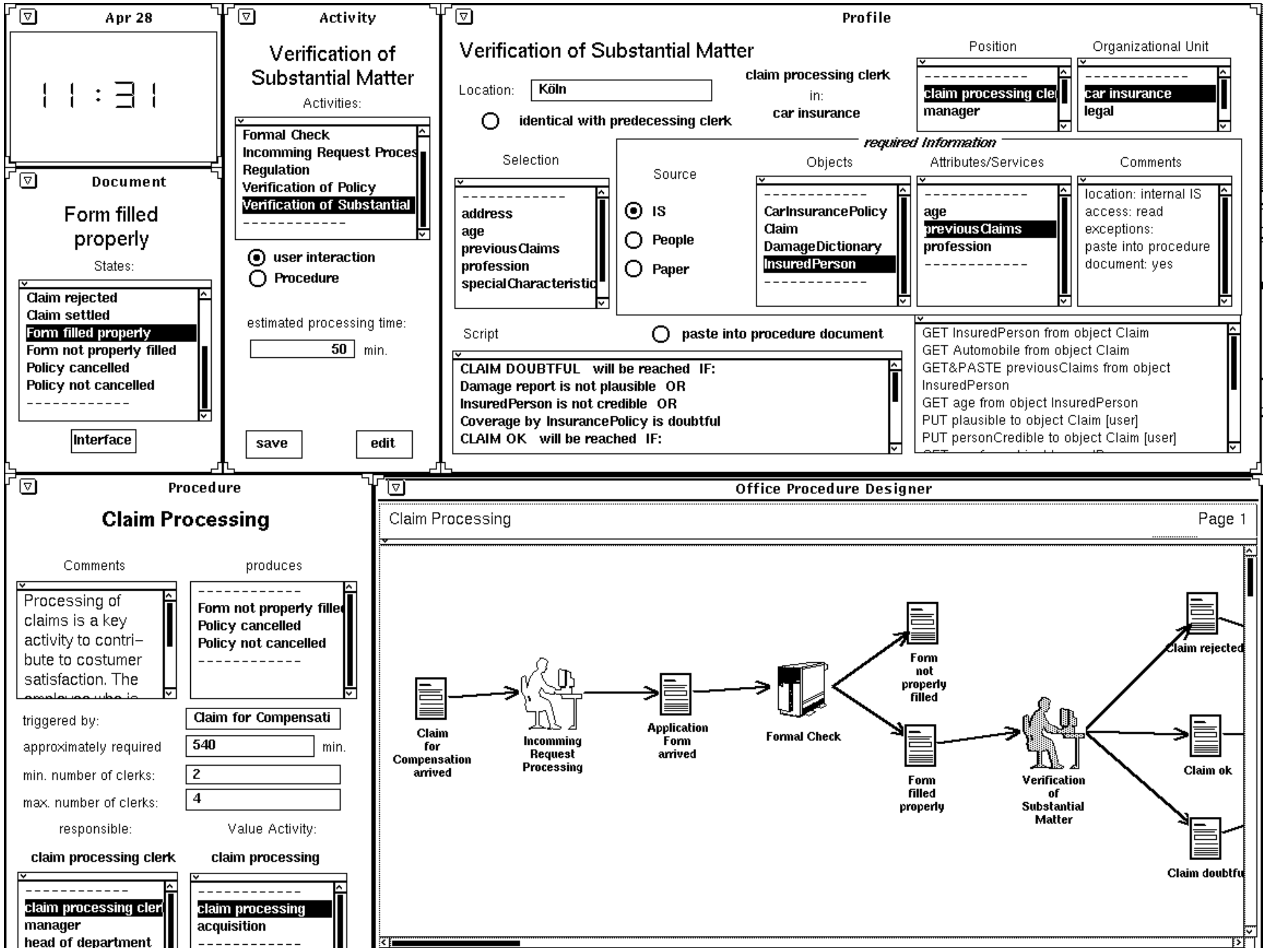
Insurance Policy

No.
Agent
Amount insured
Currently valid NO

Reported Damage

Day
Time
Report

Figure 9. User-interface generated by the Office Procedure Designer


Apr 28

11:31

Document

Form filled properly

States:

Claim rejected
Claim settled
Form filled properly
Form not properly filled
Policy cancelled
Policy not cancelled

Interface

Activity

Verification of Substantial Matter

Activities:

Formal Check
Incomming Request Proces
Regulation
Verification of Policy
Verification of Substantial

(●) user interaction
( ) Procedure

estimated processing time: 50 min.

save | edit

Profile

Verification of Substantial Matter

Location: Köln

( ) identical with predecessing clerk

claim processing clerk
in:
car insurance

Position: claim processing cle | manager

Organizational Unit: car insurance | legal

required Information

Selection: address | age | previousClaims | profession | specialCharacteristic

Source: (●) IS | ( ) People | ( ) Paper

Objects: CarInsurancePolicy | Claim | DamageDictionary | InsuredPerson

Attributes/Services: age | previousClaims | profession

Comments:
location: internal IS
access: read
exceptions:
paste into procedure document: yes

Script

( ) paste into procedure document

CLAIM DOUBTFUL will be reached IF:
Damage report is not plausible OR
InsuredPerson is not credible OR
Coverage by InsurancePolicy is doubtful
CLAIM OK will be reached IF:

GET InsuredPerson from object Claim
GET Automobile from object Claim
GET&PASTE previousClaims from object InsuredPerson
GET age from object InsuredPerson
PUT plausible to object Claim [user]
PUT personCredible to object Claim [user]

Procedure

Claim Processing

Comments:
Processing of claims is a key activity to contribute to costumer satisfaction. The

produces:
Form not properly fille
Policy cancelled
Policy not cancelled

triggered by: Claim for Compensati
approximately required: 540 min.
min. number of clerks: 2
max. number of clerks: 4

responsible: claim processing clerk
claim processing cler | manager | head of department

Value Activity: claim processing
claim processing | acquisition

Office Procedure Designer

Claim Processing | Page 1

Figure 10. User-interface of the Office Procedure Designer

During analysis the control knowledge template is filled in a narrative natural language style. This description is to be reviewed now in order to accomplish a more precise specification that refers to objects and attributes/services. In order to allow for automatic interpretation it is desirable to use a formal language. It is also necessary to analyze the exceptions that have been listed in order to specify how they should be handled. Furthermore the procedure manager´s dispatch knowledge may have to be modified.

The OPD can now generate a prototypical user-interface. To accomplish this it looks up what attributes/services as well as access types have been specified for each activity block. Within the object model a default widget should be associated with each attribute, service-parameter or returned object respectively. Taken these specifications together it is possible to preliminarily associate a set of widgets with each document state. These widgets are then placed within a window. The (sizeable) window comes up in a default size. The number of widgets however is not limited by the window size since the window´s content (that is all its widgets) is scrollable. The generated user-interface does not always provide a satisfactory layout (see figure 6 for an example of an acceptable result). Moving, resizing and even replacing widgets however can be done interactively. Each widget is linked to a service. When the objects that are needed within a procedure are instantiated it is possible to access them via the interface. Defining the order of input can be done interactively, too. A description of more specific interaction semantics can only be added as comment.

4.4 Analysis of organizational Effectiveness

After having preliminarily completed requirements analysis and design the available descriptions can be used to analyze the effectiveness of the procedure´s organization. For this purpose a communication diagram can be generated. It shows the different roles participating in the procedure as well as the media they use to communicate. For further evaluation this diagram has to be interpreted by a domain expert. A more substantial indicator for the need to reorganize the procedure is a report of detected media frictions (like they occur when paper-based information has to be transferred to the IS). Other indicators for further evaluation are the total time the involved employees have to work on the procedure as well as the costs that can be calculated from the different costs that have been specified.

Another question is more interesting but also more complicated to analyze since its scope is not restricted to the described type of procedure: what is the optimum number of employees needed to guarantee a satisfactory throughput? Or in other words: how can organizational slack be reduced to an optimum? For this kind of analysis the conceptual level is not sufficient. Instead it is the case for simulation. The current version of the OPD provides only limited simulation capabilities. Bottlenecks only occur in case more than one person works on a procedure (assumed that totally automated activities do not take considerable time). For this case it is possible to assign a number of people to each activity block that requires user interaction. Simulation then reveals bottlenecks and total throughput-numbers for different constellations. This however will only be sufficient in rare cases. Employees occupied within one procedure may also have to fulfill other tasks. It has also to be taken into account that employees have vacation days, that they may become sick (may be depending on the work load they face), that effectiveness of human work depends on a variety of aspects. Furthermore quality of work cannot be neglected, its relation to other variables however is hard to find out. Last but not least it does not make much sense to optimize the organization of a single type of office procedure. Since procedures may be interrelated you need to widen the scope (Porter´s value chain concept is one approach to get an enterprise wide view). Optimizing the organization of work has been a dream for long. We do not think that enterprise modelling along with simulating organizational alternatives will make this dream come true. It can help however to reduce complexity by providing an illustrative representation of important aspects and by detecting certain types of organizational misconception.

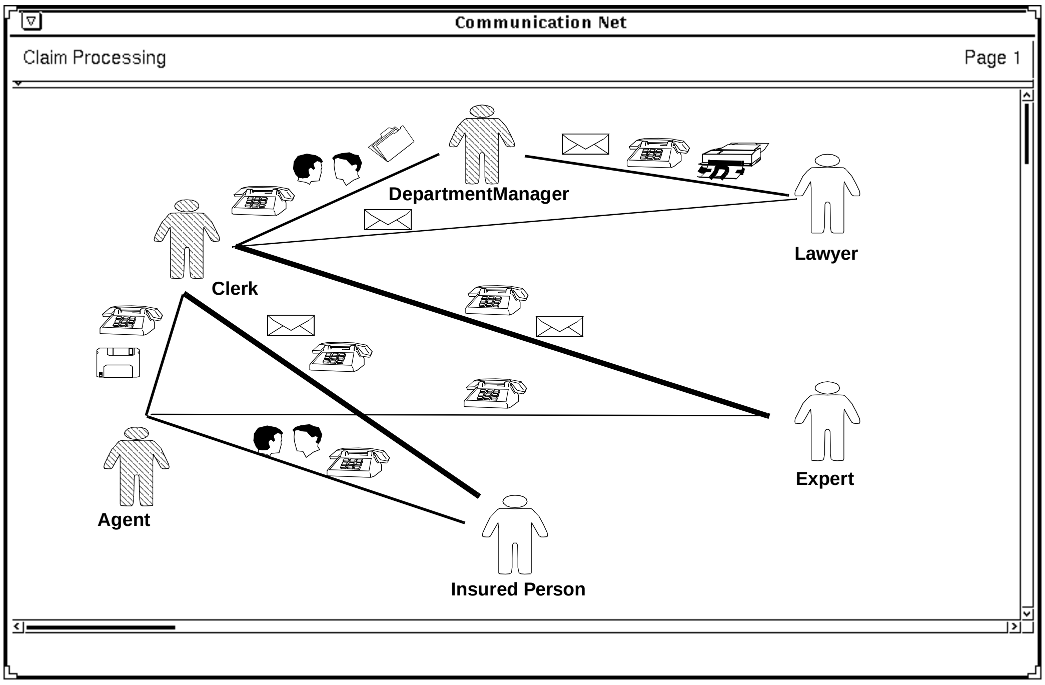


Figure 11. Communication Diagram

4.5 Prototypical Implementation

Analysis and design should deliver a comprehensive description of activity blocks. Prototypical implementation aims at completing/refining this description to an extent that allows for a test run of a procedure. Such a test run is not intended to offer simulation but to give the potential users a substantial impression of the system.

The framework needed for an office procedure has already been implemented. On the conceptual level it mainly consists of three classes: ProcedureManager, ActivityBlock, and ProcedureDocument. They have to be specialized now for the particular type of procedure. In the easiest case a class that had already been implemented in the past can be (re-)used. Otherwise specialization requires modification. This is particularly the case for new activities. The corresponding classes inherit from the abstract class "Activity-

Block” (see fig. 5). Their semantics is usually not completely formalized during analysis and design. Therefore the current version of OPD requires to write some additional code using an implementation language, which is Smalltalk in our case.

One part of the prototype´s user-interface is based on the OPD´s graphical representation of a procedure. Such a representation also provides an illustrative view of an active procedure. The current state of the procedure is indicated by a highlighted icon. To get a more detailed view (like: who is responsible for this activity, what is the name of the costumer involved in this procedure etc.) the user clicks on the icon.

5 Supporting a Strategic Perspective: The Value Chain Designer

Designing an enterprise wide object model as well as a dynamic model of office procedures does not only mean a big investment on itself. It also establishes a plan for future investments in information technology. In order to protect these investments it is desirable to apply a long term view of the enterprise. It is the core function of strategies to guide the coordination of internal activities according to the long range development of the firm, especially the adaptation to a changing environment. The strategic perspective refers to business goals and corporate culture of a firm. It provides a link between the organizational and the IS perspective. There is, for example, a strong interdependence between the development of technology and business strategies: current distributed and cooperative information systems reflect strategies and organizational concepts (e.g. concurrent engineering, outsourcing) and strategies like the development of information partnerships reflect technological options. Nevertheless, an integration of the strategic perspective into the development cycles of enterprise information systems is particularly difficult as the strategy encompasses long term, abstract and sometimes even elusive goals, which can not be easily translated into goals for systems engineers. This gap between the different perspectives held for example by system engineers or strategic managers, often leads to misunderstanding and badly coordinated activities.

In order to bridge this gap, we have developed the *Value Chain Designer* (VCD). It comprises different modules which support stepwise decomposition of strategies into value chain activities and finally business procedures. However, decomposition is not a deductive inference process. It rather serves as a heuristics to facilitate analysis and to develop a network of interrelated business activities. The analytic procedures are only partly formalized and require a considerable deal of interpretation in order to cope with complexity and ambiguity.

The core features of the VCD are:

- It offers a systematic representation and evaluation of business activities within the value chain (strength & weaknesses, potential for out-/insourcing, external links, e.g. connection to inter-organizational systems, necessity for further development and use of IS),
- It supports the analysis of input-output relations (including cost and quality aspects) and of interdependencies between activities.

Thereby it enforces a formal concept and encourages the interactive, discursive development of business strategies at the same time. Furthermore it facilitates the linking of strategic planning, organizational development and the design of information systems.

5.1 Conceptual Model: Value Chain

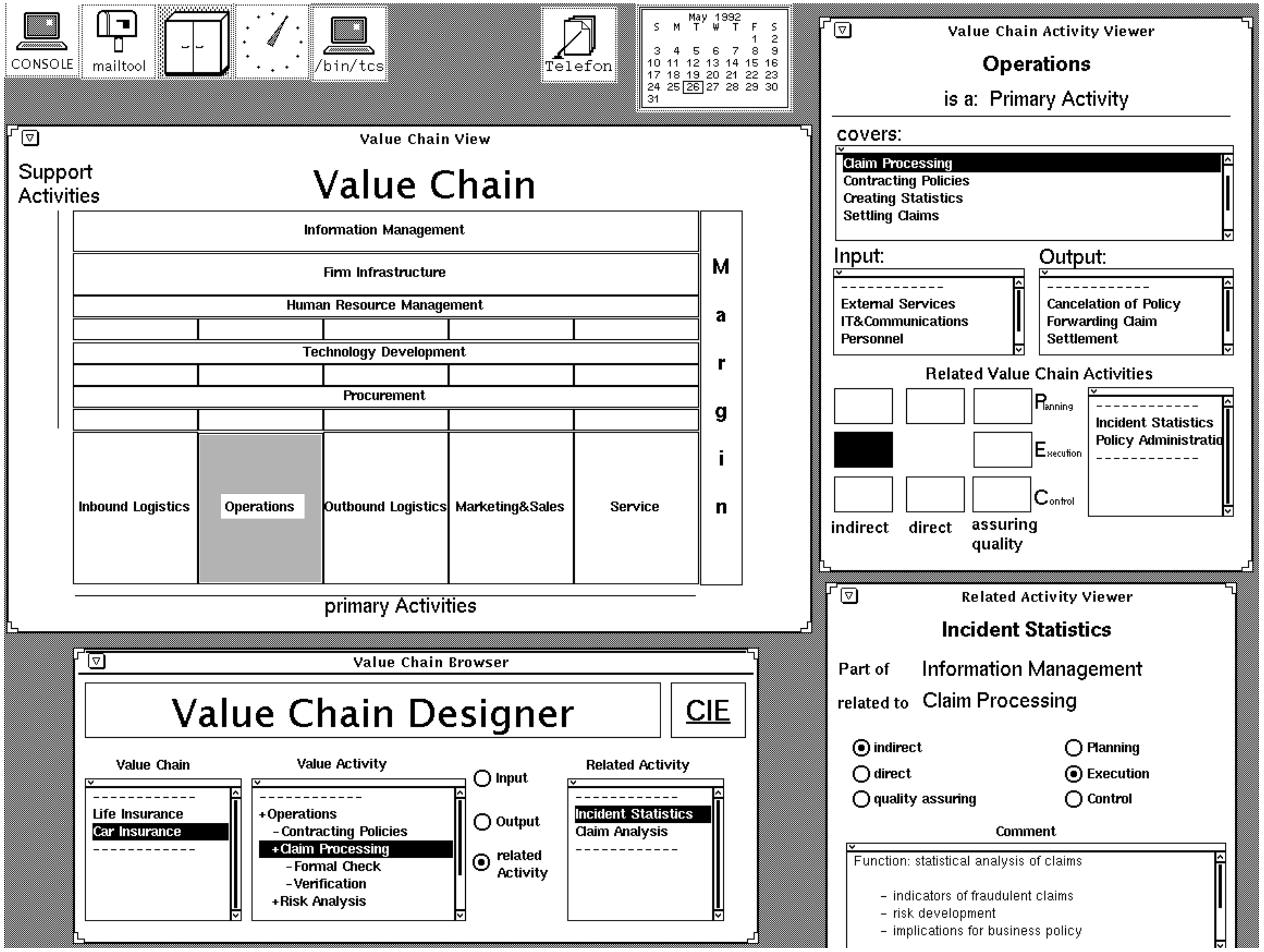
We have chosen and adapted Porter's value chain concept [Porter 1985] for its integrative potential, its dynamic perspective concerning the future developments as well as for its widespread acceptance and communicational power - and because "... the activities of the value chain constitute the foundation of the controllable factors to achieve competitive superiority." [Hax/Majluf 1991, p. 78]

The value chain is a comprehensive and generic model of business activities, which claims that all of the tasks performed by a business organization can be attributed to nine different broad categories (cf. [Hax/Majluf 1991, pp. 77]). It distinguishes between primary activities, which represent the classical managerial functions of the firm (inbound logistics, operations, outbound logistics, marketing & sales, service), and support activities, which represent the pervasive managerial infrastructure of the firm (human resource management, technology development, procurement) for the coordination and (long-term) development of resources. The value chain thus represents the internal operations of an organization in a dynamic perspective. These operations are embedded in a value system which reflects the environmental scope of business activities. Car insurance e.g. may be viewed as a part of the system of transport and logistics. The value chain aims at a holistic view of the movements of goods and services from the sources through to the consumer or "final" customer.

4.2 Strategic Scrutiny: Value Chain Modelling and Analysis

The VCD is mainly a tool for representing and documenting business activities and for strategic scrutiny. It provides a general framework as well as analytical categories to segment business units and to establish relations between different activities. It thus supports modeling of alternative future strategies. Strategic issues dealt with are - among others - *added-value, critical success factors, competitive advantage, horizontal strategy, vertical integration, business partnerships and coordination, analysis of interdependencies between different firms*.

In order to foster the evolution of a common universe of discourse (and to avoid redundancy) the user is encouraged to look up existing dictionaries (i.e. of business procedures, inputs, outputs). Only if he cannot find an appropriate item he should explicitly add a new one.



```
      May 1992
 S  M  T  W  T  F  S
                1  2
 3  4  5  6  7  8  9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30
31
```

Value Chain View

Support Activities

Value Chain

Information Management

Firm Infrastructure

Human Resource Management

Technology Development

Procurement

Inbound Logistics | Operations | Outbound Logistics | Marketing&Sales | Service

Margin

primary Activities

Value Chain Activity Viewer

Operations

is a: Primary Activity

covers:
Claim Processing
Contracting Policies
Creating Statistics
Settling Claims

Input:
External Services
IT&Communications
Personnel

Output:
Cancelation of Policy
Forwarding Claim
Settlement

Related Value Chain Activities

indirect	direct	assuring quality	
			Planning
■			Execution
			Control

Incident Statistics
Policy Administratio

Value Chain Browser

Value Chain Designer

CIE

Value Chain:
Life Insurance
Car Insurance

Value Activity:
+Operations
-Contracting Policies
+Claim Processing
-Formal Check
-Verification
+Risk Analysis

Input [ ] Output [ ] [✓ related Activity]

Related Activity:
Incident Statistics
Claim Analysis

Related Activity Viewer

Incident Statistics

Part of Information Management

related to Claim Processing

[✓ indirect] direct quality assuring

Planning [✓ Execution] Control

Comment

Function: statistical analysis of claims

- indicators of fraudulent claims
- risk development
- implications for business policy

Figure 12. User-interface of the Value Chain Designer

The VCD comprises four views, which are linked to each other. It thereby supports the inquiry into the relations and sources of value chain activities:

(1) Value Chain View

The value chain view provides a framework for business segmentation: business activities can be allocated and classified in the value chain, clusters of strategic business units can be formed and their strategic potential annotated. In particular the pervasive character of communication and information systems as part of the business infrastructure (support activities) can be illustrated. Assumptions concerning future developments, the integration of system functions and the contribution for business strategies can be made transparent and can be made subject of a common evaluation. The value chain analysis comprises an external perspective, linking activities to activities of suppliers and customers.

The activities are presented as buttons which are ordered in a way similar to Porter´s representation. Each button provides a link to a more detailed description of an activity.

(2) Value Chain Browser

The value chain consists of value chain activities (like inbound logistics, marketing, operations etc.). These activities encompass further activities which are made up of business procedures.

The Value Chain Browser presents the different value chains of a firm (in our example: life insurance and car insurance) and the hierarchical structure of the value chain activities. This tool combines the functionality of a hierarchical browser and shows for every chosen activity a list of either inputs, outputs or related activities.

(3) Value Chain Activity Viewer

A more detailed analysis of inputs, outputs and related activities is supported by the Value Chain Activity Viewer:

For every selected activity lists of inputs (resources and goods) and outputs (settlements etc.) as well as related value chain activities are represented simultaneously. Inputs and outputs can be with costs and quality measures: "At the essence of value analysis are the trade-offs to be made among price, quality, design, manufacturability, standardization, and cost." (Hax; Majluf (1991), 305)

In a matrix of interdependencies eight types of relations are distinguished: The rows refer to the quality of relation (indirect, direct, quality assurance), the columns refer to different phases of the management process (planning, execution and control). Each type is implemented as a button. When it is pressed, a list of activities which are related to the value chain activity in the specified way is presented within the Related Activity Viewer.

(4) Related Activity Viewer

For every selected activity in the Value Chain Activity Viewer the Related Activity Viewer shows which primary or support activity it belongs to and which function it fulfils. Our example this is incident statistics: it is part of information management and is related to claim processing (which in turn is linked to one or more detailed descriptions of a claim processing procedure within the Office Procedure Designer). The relation is qualified as indirect and the statistics refer to execution, rather than to planning or control, of claim processing. A text editor allows for a detailed description of the functions the chosen activity performs.

Summarized the VCD supports documentation and management of business strategies by the following conceptualizations:

- Activities are aggregated and classified into primary or support activities of the value chain.
- Primary and support activities, which represent business units, are disaggregated into several partial activities for a detailed analysis of interdependencies, costs, quality.
- The hypertext mechanisms (which are not shown in the figure) support the documentation of assumptions, background knowledge and evaluations which are linked to value chain activities.

The VCD is integrated with the OMD and the OPD. On the technical level integration is accomplished by storing all tools in one Smalltalk image. On the conceptual level integration is fostered by using common objects and by establishing links. For instance: a value chain activity is defined as a class within the OMD (providing services like *listOfOutput, listOfInput, relatedActivities* etc.). The resources used to describe a value chain activity may refer to concepts defined as classes (like organizational roles). Value chain activities can be linked to a set of office procedures (represented in the OPD), which may include a more detailed description of resources.

6 Conclusions

Our experience with modelling an office domain within an insurance company indicates that the proposed representations offer illustrative abstractions of an enterprise. This is especially the case for the representation of office procedures. The graphical notation was intuitively understood by both system analysts and domain experts. Thereby it is a valuable medium for starting knowledge acquisition or object modelling respectively. Users seem to prefer procedures as guidance in conceptualizing the domain they work in. Therefore asking for a detailed description of office procedures does not only serve the purpose of adding dynamic or temporal semantics to the model it also provides a heuristics to shape the static object model.

Our main focus was to develop general concepts and instantiate them solely for prototyping issues. However the tools can also be used on an operational level. This is specially true for the OPD and the VCD since they are also thought to guide an economic analysis of a particular firm. For this purpose it would be important not only to instantiate them with a complete description of the enterprise but also to record the evolution of business data. That would not only allow to analyze (or detect) interdependencies. It would also enrich modelling of office procedures and strategical analysis with enterprise specific data. Enterprise specific knowledge that cannot be formalized could be added using the already available hypertext features.

We plan to integrate activity based accounting (activity based costing) methods in order to find a more precise judgement of cost effects related to proposed changes and innovations. Thereby the mutual refinement of office procedure modelling and strategic analysis will be improved. At the same time it promises to facilitate the evaluation of business processes.

Although office procedures are an illustrative metaphor it is not sufficient to describe all kinds of work in the office. Ill-structured cooperative work requires other concepts as well as another graphical representation. It would be interesting to complement the Office Procedure Designer by a tool that allows for illustratively modelling CSCW-applications (for an example on the instance level see [Ellis 1987]).

Currently the tools are reimplemented using a different user-interface management system, that allows for generating Smalltalk's code and makes use of the model/view/controller-concept (Objectforms® by Heeg). The tools will then allow to dynamically change the Look&Feel of their user-interfaces (covering the following styles: Open-Look, Macintosh, Motif, Smalltalk, Windows, Presentation Manager). Furthermore we are modifying the concepts used for modelling associations and triggers.

References

Araya, A.A.; Stefik, M.J., "Generic Knowledge in Office Activities", in: Lochovsky, F. (Ed.), *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, pp. 54-59

Booch, G., *Object-oriented design with applications*. Redwood City 1990

Brodie, M.L., "On the Development of Data Models", in: Brodie, M.L.; Mylopoulos, J.; Schmidt, J. (Ed.), *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming*. Berlin, Heidelberg etc. 1984, pp. 19-47

Bruni, G.; Cardigo, C.; Damiani, M.; Seminati, G., *Final Report on Insurance Domain Requirements Analysis*. ITHACA.Datamont.89.D.7, 1990

Coad, P.; Yourdon, E., *Object Oriented Design*. New York 1991

Croft, W.B., "Representing Office Work with Goals and Constraints", in: Lochovsky, F. (Ed.), *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, pp. 13-18

Dur, R.C.J., "Dynamic Modelling for Analysis and Design of Office Systems", in: Sol, H.G.; Van Hee, K.M. (Ed.), *Dynamic Modelling of Information Systems*. Amsterdam, New York etc. 1991, pp. 303-321

ESPRIT Consortium AMICE, *CIM-OSA AD 1.0 Architecture Description*. Brussels 1991

Ellis, C.A.; Bernal, M., "OFFICETALK-D: An experimental office information system", in: *SIGOA Newsletter* 3, No. 1, 1982, pp. 131-140

Ellis, C.A., “NICK: Intelligent Computer Supported Cooperative Work”, in: Lochovsky, F. (Ed.), *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, pp. 95-102

Fischer, D.H.; Rostek, L., *SFK: A Smalltalk Frame Kit. Concepts and Use*. Darmstadt 1992 (in print)

Frank, U.; Klein, S., *Unternehmensmodelle als Basis und Bestandteil integrierter betrieblicher Informationssysteme*. GMD research paper, No. 629, Sankt Augustin 1992

Frank, U. , "Designing Procedures within an Object-Oriented Enterprise Model". In: Sol, H.G. (Ed.), *Dynamic Modelling of Information Systems*. Delft 1992

Frank, U.: "A Tool-Supported Methodology for Designing Multi-Perspective Enterprise Model". To appear soon in: Proceedings of the Third Australian Conference on Information Systems. Wollongong 1992

Graham, I., *Object oriented methods*. Redwood City 1991

Hax, A. C.; Majluf, N. S., *The Strategy Concept and Process - A Pragmatic Approach*. Englewood Cliffs 1991

Hogg, J.: OTM, “A Language for Representing Concurrent Office Tasks”, in: Lochovsky, F. (Ed.), *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, pp. 10-12

Hogg, J.; Nierstrasz,O.M.; Tsichritzis,D., “Office Procedures”, in: Tsichritzis, D. (Ed.), O*ffice Automation*. Berlin, Heidleberg etc. 1985, pp. 137-165

IBM, *IBM Enterprise Business Process Reference Model*. 1990

Katz, R.L.: Business/enterprise modelling. In: *IBM Systems Journal*, Vol. 29, No. 4, 1990, S. 509-525

Kreifelts, T.; Woetzel, G., “Distribution and Error Handling in an Office Procedure System”, in: Bracchi, G.; Tsichritzis, D. (Ed.), *Office Systems: Methods and Tools*. Proceedings of the IFIP WG 8.4 1986. Amsterdam, New York etc. 1987, pp. 197-208

Levesque, H.J.; Mylopoulos, J., “An Overview of Knowledge Representation”, in: Brodie, M.L.; Mylopoulos, J.; Schmidt, J. (Ed.), *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming*. Ed. by , Berlin, Heidelberg etc. 1984, pp. 3-17

Lochovsky, F.H.; Hogg, J.S.; Weiser, S.P.; Mendelzon, A.O., “OTM: Specifying office tasks”, in: Allen, R.B. (Ed.): *Conference on Office Information System*s. New York 1988, pp. 46-54

Macdonald, H.K., "The Value Process Model", in: Scott Morton, M.S. (Ed.), *The Corporation of the 1990s: Information Technology and Organizational Transformation*. New York, Oxford 1991, pp. 299-309.

Meyer, B., *Object-Oriented Software Construction*. New York 1989

Porter, M.E., C*ompetitive Advantage. Creating and Sustaining Superior Performance.* London 1985

Pröfrock, A.-K.; Tsichritzis, D.; Müller, G.; Ader, M., "ITHACA: An Integrated Toolkit for Highly Advanced Computer Applications", in: Tsichritzis, D. (Ed.), *Object Oriented Development*. Genf 1989, pp. 321-344

Rothenberg, J., "Prototyping as Modelling: What is Being Modeled?", in: Sol, H.G.; Van Hee, K.M. (Ed.), *Dynamic Modelling of Information Systems*. Amsterdam, New York 1991, pp. 335-357

Rumbaugh et.al., *Object-oriented modelling and design*. Prentice Hall 1991

Schank, R.C., *The Cognitive Computer. On Language, Learning and Artificial Intelligence*. Reading/Mass. 1985

Schultz, R., *ZOOM - The Object-oriented Information Systems Architecture Modeler*, Version 1.0 (Software Documentation), Dublin/Ohio 1990

Tsichritzis, D., "Form Management", in: *Communications of the ACM*, Vol.25, No.7, July, 1982

Zachman, J.A., "A framework for information systems architecture", in: *IBM Systems Journa*l, Vol. 26, No. 3, 1987, pp. 277-293