Published in: McGregor, R. (Hg.): Whitley, E.A. (Hg.): Proceedings of the First European Conference on Information Systems. London 1993, pp. 131-140

An Integrated Environment for Designing Object-Oriented Enterprise Models

Ulrich Frank
German National Research Center for Computer Science
Schloß Birlinghoven, 5205 Sankt Augustin, Germany

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 static view that focuses on the representation of structural aspects, a dynamic view that serves to represent office procedures and a strategic view. The environment that is introduced is intended to encourage the design of enterprise models on these three levels as well as interconnecting them. The paper however will only focus on two levels - the strategic view will be neglected. Two tools of the environment will be described in detail. The Object Model Designer guides conceptualisation 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 conceptualised as ordered graphs of activity blocks. It provides means to analyse the effectiveness of business procedures and generates prototypical user-interface as a representation of a virtual procedure document.

Beside describing the tools listed above the paper presents a conceptual framework for the design of multi-perspective enterprise models. 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.

1 Introduction

Designing, implementing and using corporate information systems face numerous challenges, e.g. frictions between the different stages of system lifecycles 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 a) 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 analyse 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 analyse, design and use an information system
- contributing to strategic planning of the business and the 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.

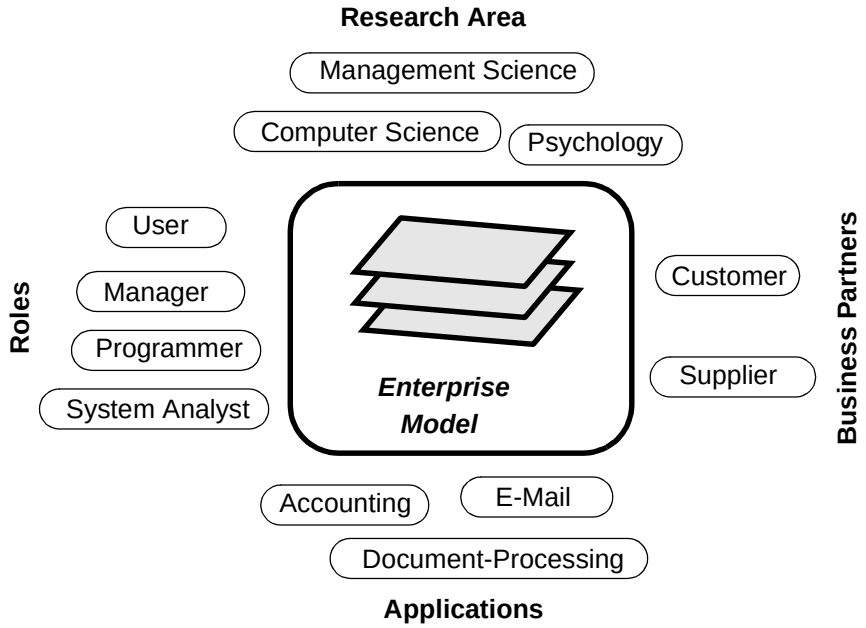

Figure 1. Dimensions of Integration fostered by Enterprise Models

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.

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 life-cycle 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.

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 conceptualisations" (Levesque & Myloupolos 1984). Second: since there are different conceptualisations 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/conceptualisations (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:

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

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. The strategic level - which is not subject of this paper - is represented using concepts like goals, value chains (Porter), portfolios and corporate culture. 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.

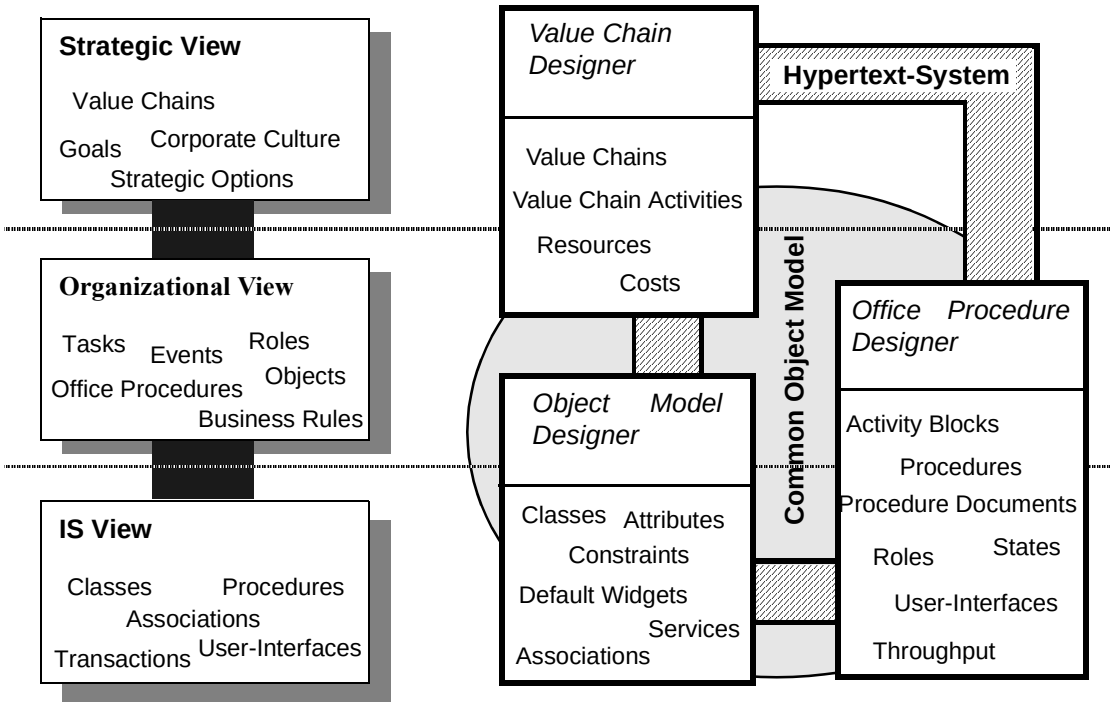

Figure 2. Tools of the design environment and their relation to different views of the enterprise

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 (for a description of the *Value Chain Designer* see Frank & Klein 1992 b) which are enhanced by a hypertext-system. For an illustration of the tools correspondence to the three main levels of abstraction see figure 2. All the tools have been written in Smalltalk-80 within the Objectworks® environment, so they are highly integrated by residing in a single Smalltalk-Image.

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

An object/class is modelled by describing attributes, services, associations and triggers. Additionally it can be assigned a default view.

An attribute is regarded as an object that is encapsulated within the object. Among others it is described by *class*, *cardinality* and *history*. Specifying an attribute´s class is a prerequisite for typing. Cardinality has to be defined in min., max.-notation. For instance: a customer´s telephone number may have cardinality 0,*. If history is set to true every update of the attribute has to be recorded somehow. 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. This approach is a first attempt to deal with the complexity of 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. From a software engineering point of view it is desirable to limit the number of association types that are used.

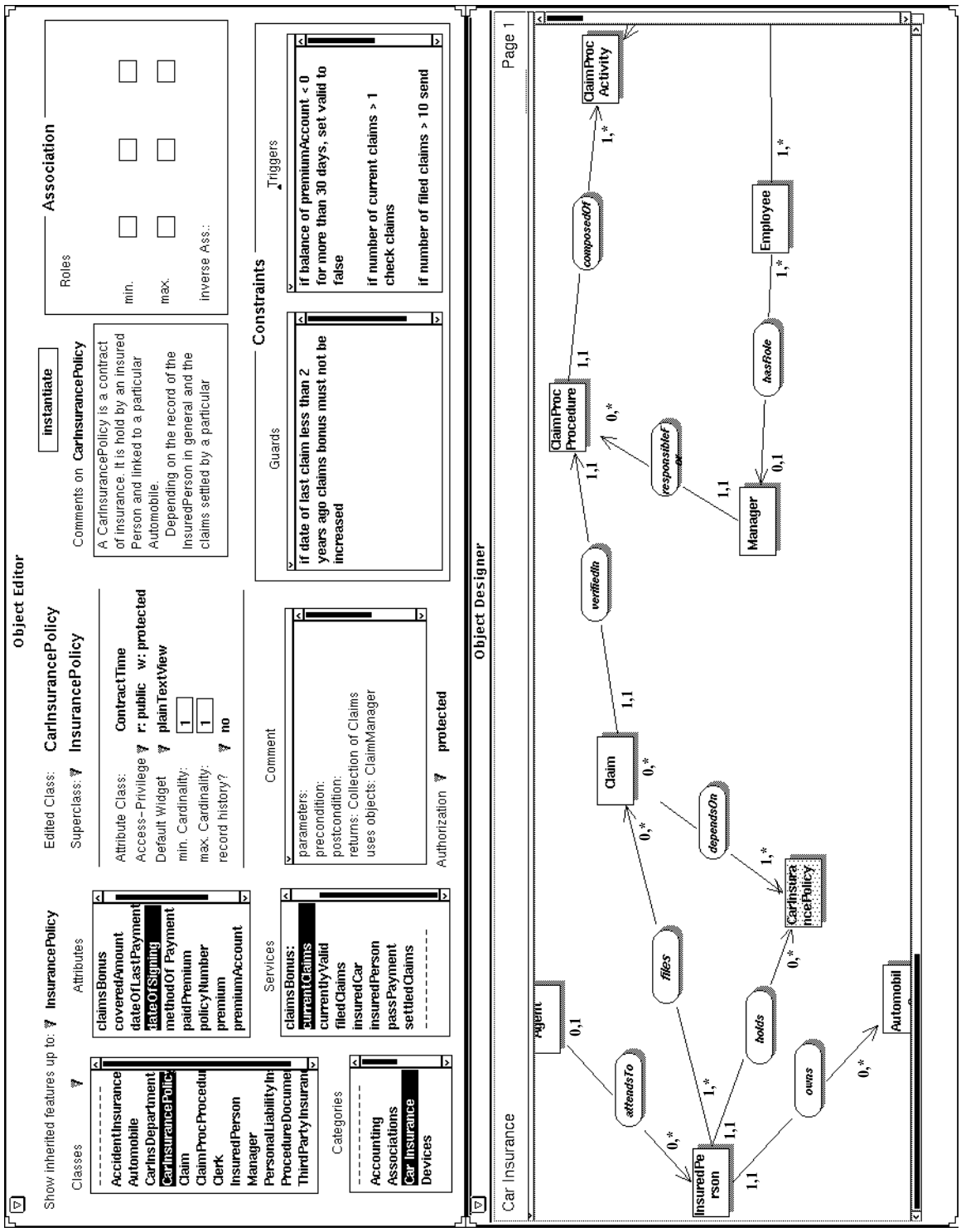

Figure 3. User-interface of the Object Model Designer

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 labelled 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. 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. 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.

The Office Model Designer allows for a graphical representation of an object model (see screenshot in fig. 3). Furthermore it provides dictionaries of already defined classes and checks for name conflicts. 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. 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 into 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 (for an example of the transformation into SFK see Frank 1992).

4 Adding Dynamic Aspects: The Office Procedure Designer

While an object model can be sufficient to capture all the semantics you need for implementation it is definitely not sufficient to cover all important aspects of analysis and design. It hardly allows for comprehensively expressing temporal and functional semantics of an information system. Object-oriented analysis and design methodologies (like Booch 1990, Rumbaugh et al. 1991) usually suggest to complement the static object model with a dynamic model (represented by state transition diagrams), and a functional model (represented by data flow diagrams). However, for our purpose these techniques have two shortcomings. They do not provide a representation that is comprehensive for non IS-people. Since state transition diagrams describe the behavior of objects of a certain class they can hardly be used to support the design of procedures from preexisting components. The Office Procedure Designer is intended to provide a more illustrative representation that is based on an elaborated software architecture at the same time.

4.1 Conceptualisation of an Office Procedure

We regard an office procedure as an ordered graph of activity blocks (which we will refer to as activity as well), which can be represented as a semantically enriched Petri net. Each activity block (for a similar conceptualisation compare 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 information that is processed within a procedure is collected 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.

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.

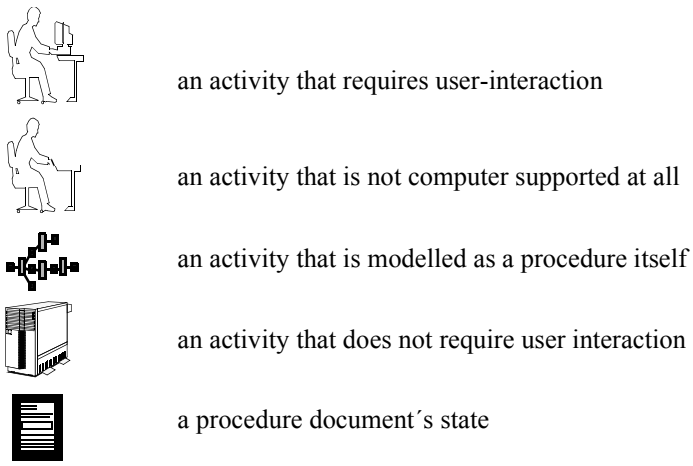

Figure 4. Icons used for the graphical representation of office procedures

4.2 The Design Process

The Office Procedure Designer provides the analyst/designer with an interactive template for systematically describing a procedure´s activity blocks. 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.

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 tool 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).

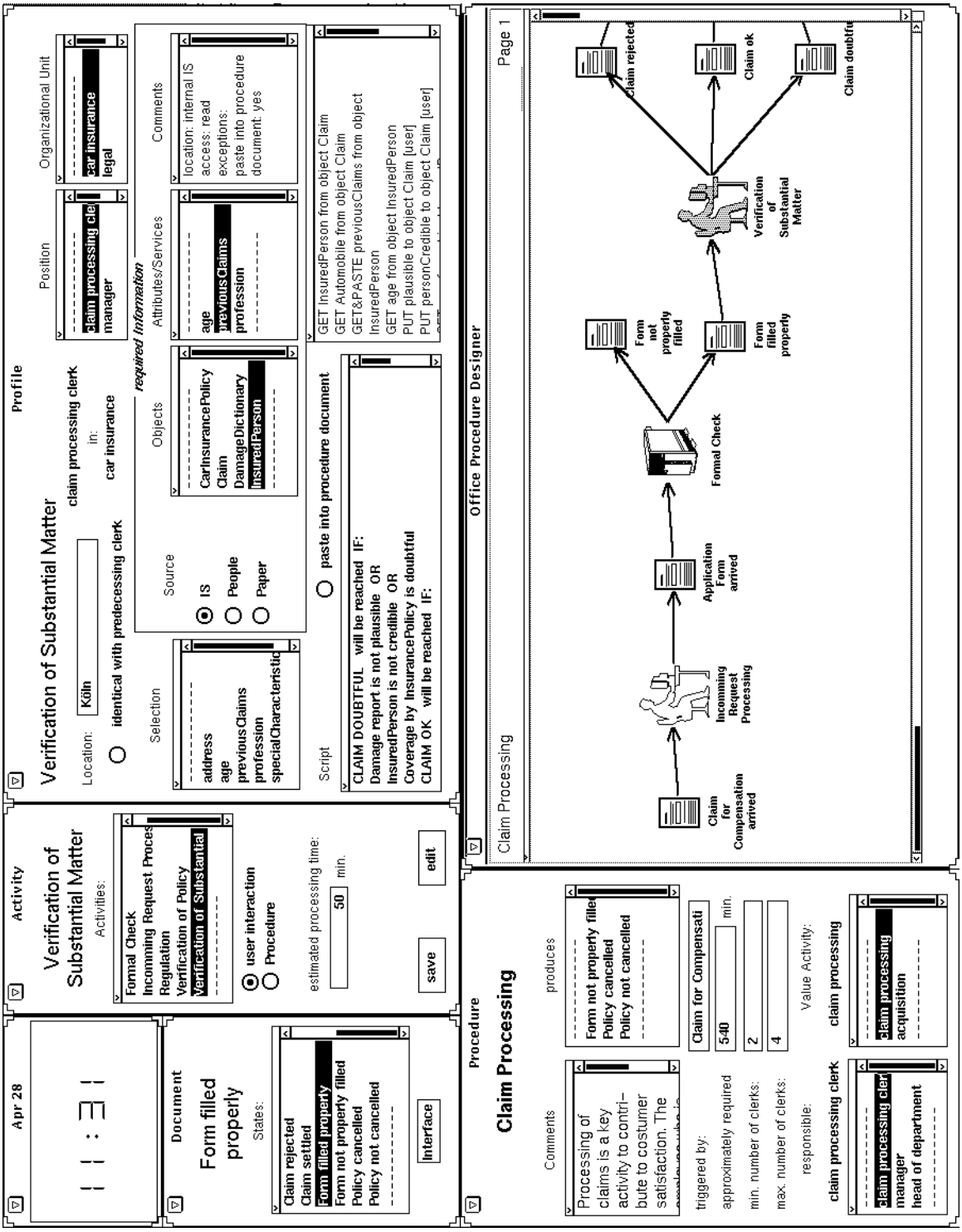

Figure 5. User-interface of the Office Procedure Designer

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*.

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. It encourages a declarative description, which may be more or less formal. 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 (see fig. 5).

The framework needed for an office procedure is already implemented. On the conceptual level it mainly consists of three classes: procedure manager, activity block, and procedure document. For the purpose of prototyping they have to be specialized 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 "ActivityBlock". 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.

4.3 Generating User Interfaces and Analyzing Organizational Effectiveness

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 (for an example of an acceptable result see Frank 1992). Moving, resizing and even replacing widgets however can be done interactively.

The OPD also allows to analyse the effectiveness of a 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.

5 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 conceptualising 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 analysis of a particular firm´s competitive position. 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 analyse (or detect) interdependencies. It would also enrich modelling of office procedures and strategical analysis with enterprise specific data. Enterprise specific knowledge that cannot be formalised could be added using the already available hypertext features.

Although office procedures are an illustrative metaphor it is not sufficient to describe all kinds of work in the office. Ill-structured cooperative work however 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).

References

Booch G (1990) *Object-oriented design with applications*. Benjamin/Cummings, Redwood City

Coad P and Yourdon E (1990) *Object-Oriented Design*. Prentice Hall, Englewood-Cliffs

Fischer D H and Rostek L (1992) *SFK: A Smalltalk Frame Kit. Concepts and Use*. GMD research paper, Darmstadt

Frank U and Klein S (1992 a) *Unternehmensmodelle als Basis und Bestandteil integrierter betrieblicher Informationssysteme*. GMD research report, No. 629. Sankt Augustin

Frank U and Klein S (1992 b) *Three integrated Tools for designing and prototyping Object-Oriented Enterprise Models*. GMD research report, No. 689. Sankt Augustin

Frank U (1992) Designing Procedures within an Object-Oriented Enterprise Model. In *Dynamic Modelling of Information Systems* (Sol H G and Crosslin R L, Eds.) pp. 385-388. Delft

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

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

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

Katz R L (1990) Business/Enterprise Modelling. *IBM Systems Journal*, Vol. 29, No. 4, 509-525

Levesque H J and Mylopoulos J (1984) An Overview of Knowledge Representation. In *On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming* (Brodie M L, Mylopoulos J and Schmidt J, eds.) pp. 3-17. Springer, Berlin, Heidelberg etc.

Lochovsky F H, Hogg J S, Weiser S P and Mendelzon A O (1988) OTM: Specifying office tasks. In *Proceedings of the ACM SIGOIS Conference on Office Information Systems*. Palo Alto

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

Rumbaugh J et.al. (1991) *Object-oriented modelling and design*. Prentice Hall, Englewood Cliffs

Savage C M (1990) *Fifth generation management - integrating enterprises through human networking*. Digital Press

Zachman J A (1987) A framework for information systems architecture. *IBM Systems Journa*l, Vol. 26, No. 3, 277-293