Published in: Proceedings of the Third International Working Conference
on Dynamic Modelling of Information Systems. Delft 1992, S. 365-388

DESIGNING PROCEDURES WITHIN AN OBJECT-ORIENTED ENTERPRISE MODEL

Ulrich Frank
GMD
Postfach 13 16
5205 Sankt Augustin 1, Germany
Phone: xx49 2241 142705
E-Mail: fr@gmdzi.gmd.de

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. Two views are of outstanding importance: the static object model and a dynamic model. First it is demonstrated how conceptualizing an object model is guided by the *Object Model Designer*. Object models are represented using a graphical notation that is enhanced by a structured description of classes and associations which are modelled as classes as well. Then a methodology and a tool related to it - the *Office Procedure Designer* - to develop dynamic models of office domains are presented in detail. The tool allows for a graphical representation of office procedures that is illustrative for business people and takes into account the requirements of object-oriented analysis and design at the same time. In interaction with the Object Model Designer it supports identification, specification and refinement of objects (classes). It also provides means to analyze the effectiveness of business procedures. The environment supports the integration of different steps within the development process from analysis to prototypical implementation. Is also allows to specify features of user interaction which are used to generate prototypical user-interfaces.

1 INTRODUCTION

Design, implementation and use of corporate information systems are still afflicted with severe economic problems. To name a few:

- There is not enough emphasis on a thorough and detailed requirements analysis that would also include possible future demands.
- The participants in requirements analysis - like designers, users and managers - do not share the same view of the problem. In other words: when they talk about the enterprise or features of information systems they do not necessarily use the same language.
- The software *architecture* does not sufficiently reflect needs for maintenance and adaptability.

- Documentation is poor. There are no guidelines for style and content of documentation.
- The transition from design to implementation is still costly. It is often accompanied by loss of information.
- Software development is too expensive. The results however are often poor.

Analyzing these and related problems reveals a lack both of *integration* and *reusability*. Enterprise-wide conceptual models are commonly regarded as a proper orientation to overcome these problems. For a number of reasons an object-oriented approach seems to be very promising. The project "Computer Integrated Enterprise" [12] that had been started in 1990 at the German National Research for Computer Science is dedicated to this subject. The general project goals are to:

- develop a methodology as well as a set of tools to support the design and maintenance of enterprise models.
- demonstrate the benefits of object-oriented analysis and design in a real-world application by comprehensively modelling a specific enterprise.

We regard an object model as the core of an enterprise model. 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. An object model hardly allows for expressing dynamic aspects of an information system. Object oriented design methodologies (like [2, 21]) suggest state transition diagrams. However, we found these techniques not to be appropriate for our idea of enterprise modelling (see also chapter 4). It has also to be taken into account that objects are not always the preferred conceptualization for domain experts to describe their perception of reality. We found that they rather choose a procedural perspective that is expressed in terms like functions, activities and business procedures. Furthermore it is desirable to model the different users´ views of objects - which can be best evaluated by providing a prototypical user-interface. Views and user-interfaces however can hardly be part of a static object model since they vary with the user´s working context.

The methodology and design-environment that is introduced in this paper is intended to overcome these deficiencies. Particularly it is to

- provide a representation of office procedures that is illustrative for business people and takes into account the requirements of object-oriented analysis and design at the same time.
- support identification, specification and refinement of objects (classes).
- provide means to analyze and refine the effectiveness of business procedures.
- demonstrate the construction of procedures by (re-) using configurable components.
- contribute to fast generating of prototypical user-interfaces.
- demonstrate a smooth integration with the object model.

The term “enterprise model” is far away from being used in a unique way. Therefore I will first outline a framework that illustrates the purposes and design requirements of enterprise models from our point of view.

2 A PRELIMINARY FRAMEWORK FOR DESIGNING MULTI-PERSPECTIVE ENTERPRISE MODELS

Enterprise wide conceptual models are widely regarded as the most important prerequisite to build integrated corporate information systems. Integration however should not be restricted to a mere technical point of view. On our opinion enterprise models should be designed to foster other dimensions of integration as well. At least the following aspects should be taken into account:

- Integrating the components (software, hardware) of a computerized information system.
- Integrating the different stages of system life-cycles.
- Integrating the views of those who participate in analyzing, designing, implementing and using an information system - as well as the views of those who make decisions about corporate strategies.
- Mutual integration of organization and information system.

Although these dimensions certainly require individual approaches, all those approaches need to have one feature in common: in order to accomplish integration you have to establish conceptualizations of the domain that is of interest, which are shared by a collection/community of technical or human interpreters. Integration then happens by referring to common concepts. We all know such semantic reference systems: sets of data types, functions of an operating system, the terminology of a certain community of software-developers or of sales-personal in a particular firm. Looking at these examples reveals that enterprise models should represent different views of the enterprise.

2.1 Perspectives on the Enterprise

Considering the numerous views/conceptualizations (see for instance [8], [24]) 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*

Designing and implementing a corporate information system requires to have a solid idea of how the operations of the firm are organized. Since an information system should be effective for a long time it is desirable to consider strategic options in time. On the other hand the information system is such an important part of the organization that managers should be provided with an illustrative representation of the information system. The strategic view, which is not subject of this paper, is described by concepts like business goals, value chains (Porter), portfolio analysis, corporate culture etc. Within the organizational view we differentiate between three perspectives. The macro-view describes the main organizational units or functional areas of an enterprise, like marketing, accounting, controlling, personal etc. On a more detailed level concepts like roles, functions, objects, business rules, and scenes (for instance: sales negotiations) are described. Finally there is a dynamic level to represent office procedures in a way that is illustrative for managers. The informations system view is focussed on what traditionally is called conceptual model. It should cover the following aspects ([3], p. 20):

“

- Static properties such as objects, object properties (sometimes called attributes), and relationships amongst objects (i.e., a particular class of object properties)
- Dynamic properties such as operations on objects, operation properties, and relationships amongst operations (e.g., to form transactions)
- Integrity rules over objects (i.e., database states) and operations (i.e. state transitions)”

Thereby it is important to emphasize that models should be illustrative for people involved in the development process, that is they should use concepts people in the domain of interest are familiar with (for a more detailed discussion on this matter see [20]). While it is often argued that the user-interface should not be part of a conceptual model because it depends on varying user requirements and technical changes we think it is desirable to include a model of user interaction. A user-interface can be essential for the effective use of a system. For this reason it should be taken into account in time. Implementing a convenient user interface takes a great amount of the overall effort. Thinking of a conceptual model as prerequisite for reusable software components it is not satisfactory to exclude user-interface issues from the model. It is important however to abstract as much as possible from specific features that are supposed to be subject of technical change.

For specifying a conceptual model of the IS as well as for reconstructing the other views different levels of abstraction could be used. Is there any indication for the appropriate level of abstraction or - in other words: what is the right amount of semantics that is specified for the terms that are used? In general we may state that the concepts that are referred to in order to accomplish integration should contain enough semantics not to bother any of the involved components/humans with the need to reconstruct meaning for further processing. On the other hand they should also support transparency by avoiding details that are not relevant for the given purpose of integration. These thoughts recommend to use elements within the IS-model that represent domain level concepts rather than more general technical concepts. For instance: define a class “account” rather than only providing classes 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 (rather than displaying some sort of graphical primitives) - which would improve the chances for powerful interpretations.

While domain level semantics certainly promote integration, evaluating the amount of semantics is more complicated when it comes to reusability. We can hardly agree with Graham ([13], p. 239) who claims: “The fact is that all semantics compromise reuse”. It seems to be more reasonable to put it this way: comfortable reuse requires a high amount of semantics, however, the more semantics you put into a concept the higher is the chance that it does not satisfy all specific requirements of the intended domain anymore. Our consequence from this logical conclusion is not to incorporate less application level semantics. Whenever it does not seem to be possible to find general concepts we would try another - less elegant - approach to accomplish reusability: by providing a set of more special concepts that is offered to be selected from in the case of a particular enterprise.

For enterprise models to be a medium for integration of the (in our case: three) different views it is crucial to translate or mediate between them. Since the concepts that are used on the strategic and on the organizational level cannot always be formalized, translation to the

concept of the IS-level in the sense of a unique mapping is sometimes impossible. In these cases one could use some sort of links that indicate that the concepts are related.

The idea of multi-perspective enterprise models recommends to aim towards authentic abstractions of reality. We think however that it is not sufficient to take the way an enterprise is actually organized for granted. This is specially important when you think of effectively integrating an information system with the organization; like Schank ([22], pp. 23) states for a broader context:

> "The first users of cars and computers had to struggle to make these completely new machines operate within the limits of the systems that were designed for an earlier world. ... Computers are severely limited by the world views and ideas that have preceded them."

Merely concentrating on representing given structures may result in sophisticatedly reconstructing a mess. In order to avoid such a misconception it is necessary to provide means to analyze the business, particularly the effectiveness of organizational structure and business processes. Therefore a methodology/environment that is to support the design of enterprise models should allow for analysis and simulation on the instance level (see [7] for a similar approach).

Although generic enterprise models that fulfil the requirements of a wide range of firms are an attractive research vision it is not possible to develop such models from scratch. You have to start with one enterprise of a particular domain. Applying the same approach to a set of similar enterprises allows for comparatively analyzing invariances and differences. Then there is a chance to condense the specific model to a configurable generic model of a certain scope. The domain we started with is car insurance within an insurance company.

2.2 Positioning the Tools within the Framework

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. 1). The Value Chain Designer (which is not subject of this paper) allows to design and analyze strategic options by applying Porter´s concept of value chains. Value activities, the core elements of value chains, are in part described by resources and business processes they use - which are described on the organizational level. The primary scope of the Object Model Designer is the IS-view. It is however possible to model links to related concepts on other levels. For instance: class "InsurancePolicy" refers to a real world object that may include legal aspects which cannot be formalized. Such aspects could be described on the organizational level using appropriate representations and then serve as an additional comment/explanation for the IS-level as well. The Office Procedure Designer mainly covers (and connects) the organizational and the IS-view.

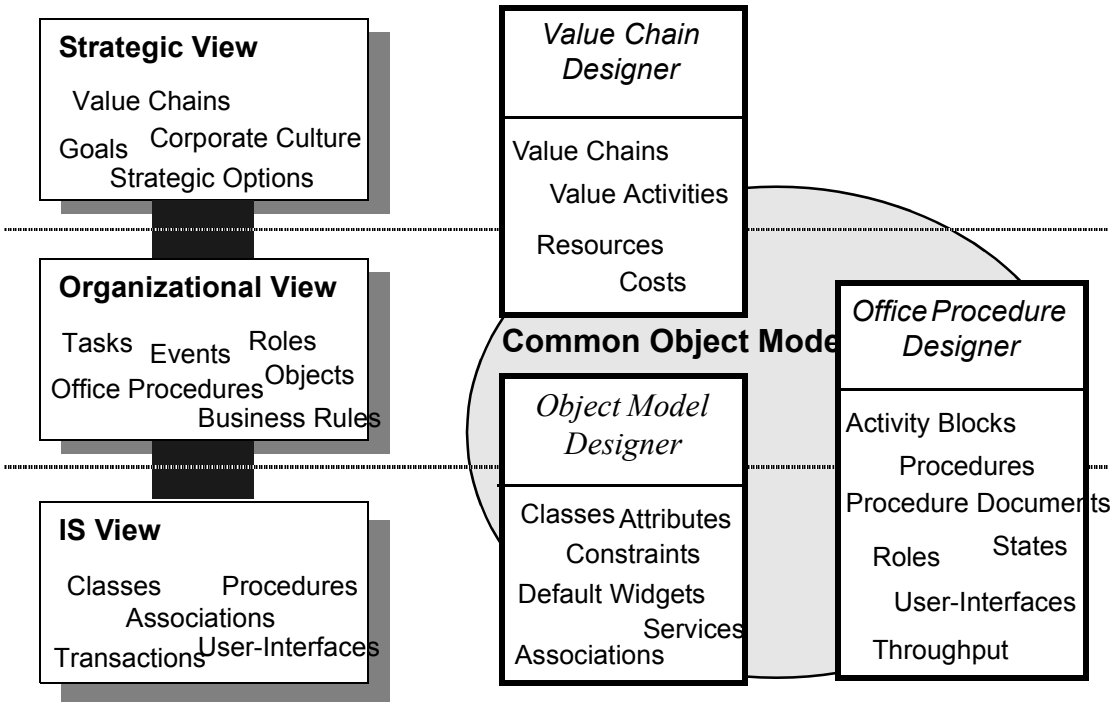

Figure 1. Tools of the design environment and their relation to different views of the enterprise

3 THE OBJECT MODEL DESIGNER

The objects/classes and their relationships constitute the IS-model of an enterprise. The Object Model Designer (OMD) is to support a systematic approach to object-oriented analysis and design. It fosters fast-prototyping by partially generating executable code from design-specifications. In order to allow for a distinctive as well as illustrative description a structured object-editor is combined with a graphical representation of the model. The object model can be accessed by other design-tools, too.

3.1 Conceptualizing Objects

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 [2] and Rumbaugh et.al. [21]. 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 attribute is regarded as an object that is encapsulated within the object. We do not allow attributes - like Coad/Yourdan [5] - 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*
- *default widget*

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:.

In order to allow for generating prototypical user-interfaces it is possible to assign a default-widget to each attribute. One can also define a label that is to be presented with the widget. Additionally the size of the widget can be specified. 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. To improve the chance to assign an appropriate user-interface it is possible to define widget groups, for instance to display an address.

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. Like an attribute a service can be defined public, protected or private, which means to propagate it into the classes public protocol, to allow it to be used only with some sort of authorization (like a password) or to keep it secret. 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 costumer 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 general it is desirable to provide multiple-inheritance which however should be used very carefully. The current version of the OMD only allows to define single-inheritance relationships between classes. One reason for this decision was not to hinder implementation by languages that do not support multiple-inheritance which is particularly true for the implementation language we use for generating executable objects (Smalltalk). There is however a possibility to define semantics that are multiple-inheritance alike using the concept of roles (see below).

3.2 Associations

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 [2] 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. [21] 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.

Figure 2. User-interface of the Object Model Designer

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.

3.3 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 [11]. The conceptual description of a class can partially by 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. The following example shows part 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 3. Partial definition of class" CarInsurancePolicy" in SFK

4 THE OFFICE PROCEDURE DESIGNER

Although the object model includes a few dynamic aspects (like method pre- and postconditions) it does not allow to model business procedures in an illustrative and comprehensive way. Object oriented design methodologies (like [2, 21]) 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 [6, 9, 14, 15, 16, 23]) 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 [14, 17]) 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 precon-

dition. 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.

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 state

Figure 4. 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.

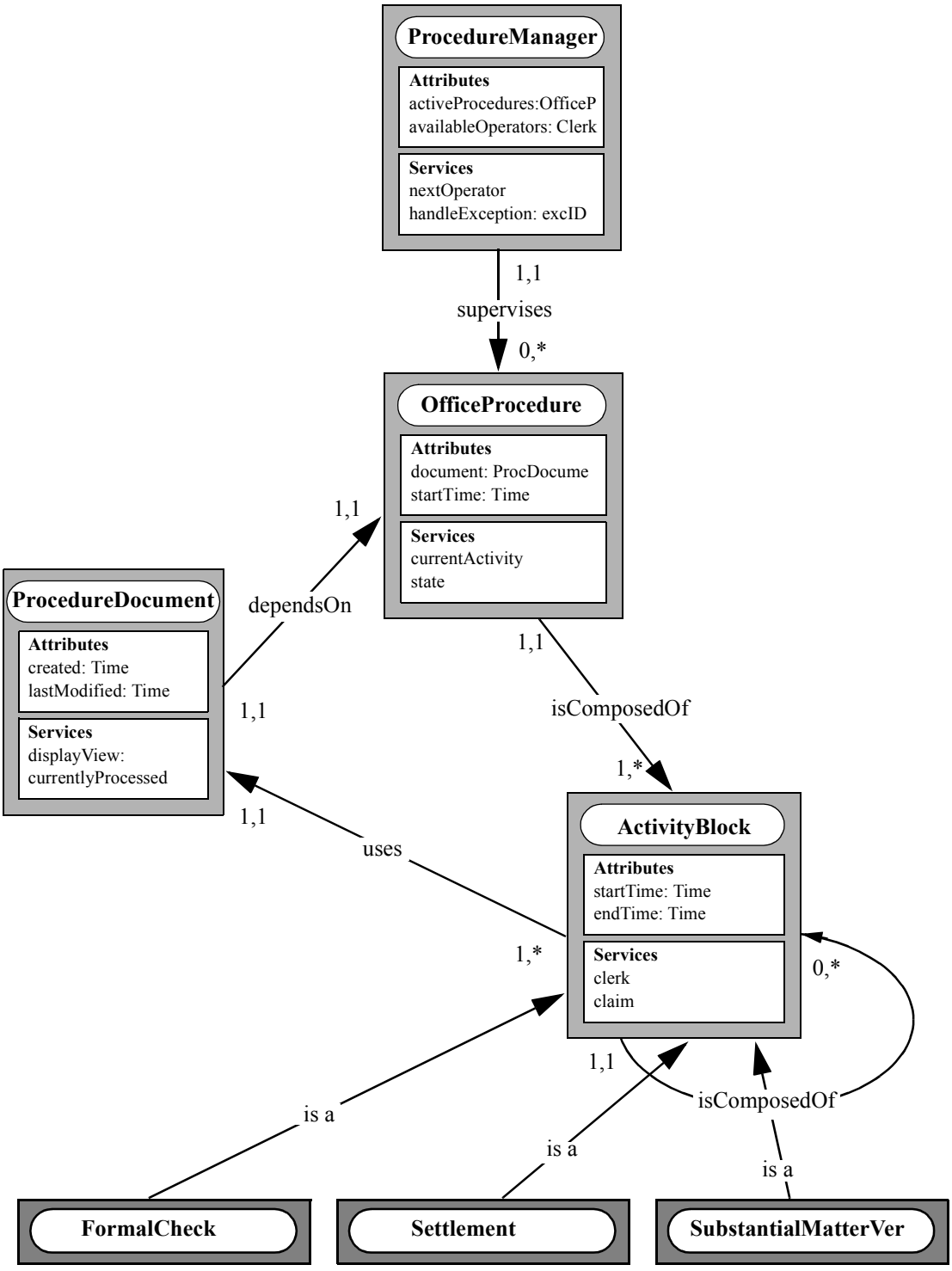

Figure 5. Partial object model of an office procedure

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.

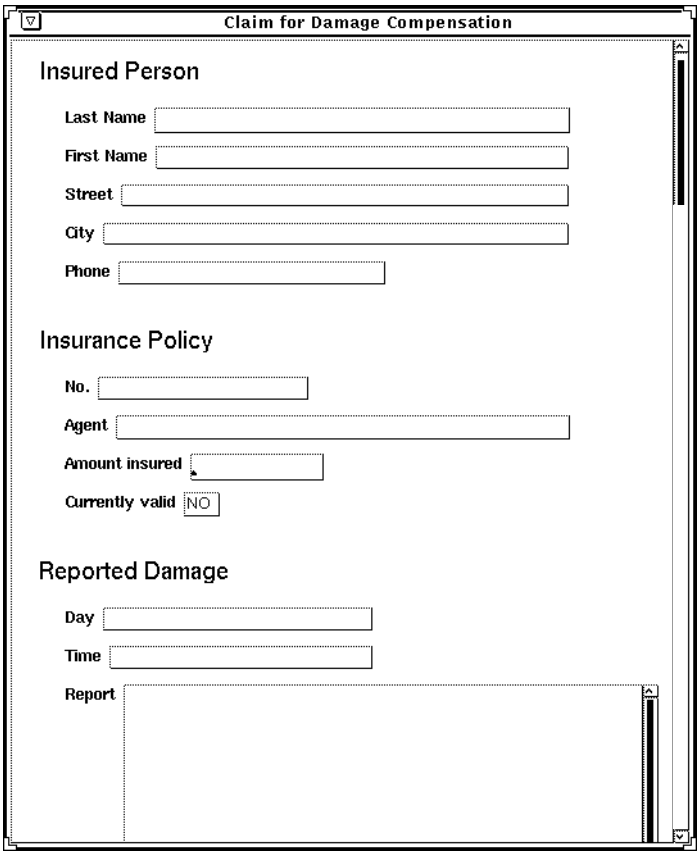

Figure 6. User-interface generated by the Office Procedure Designer

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.

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.

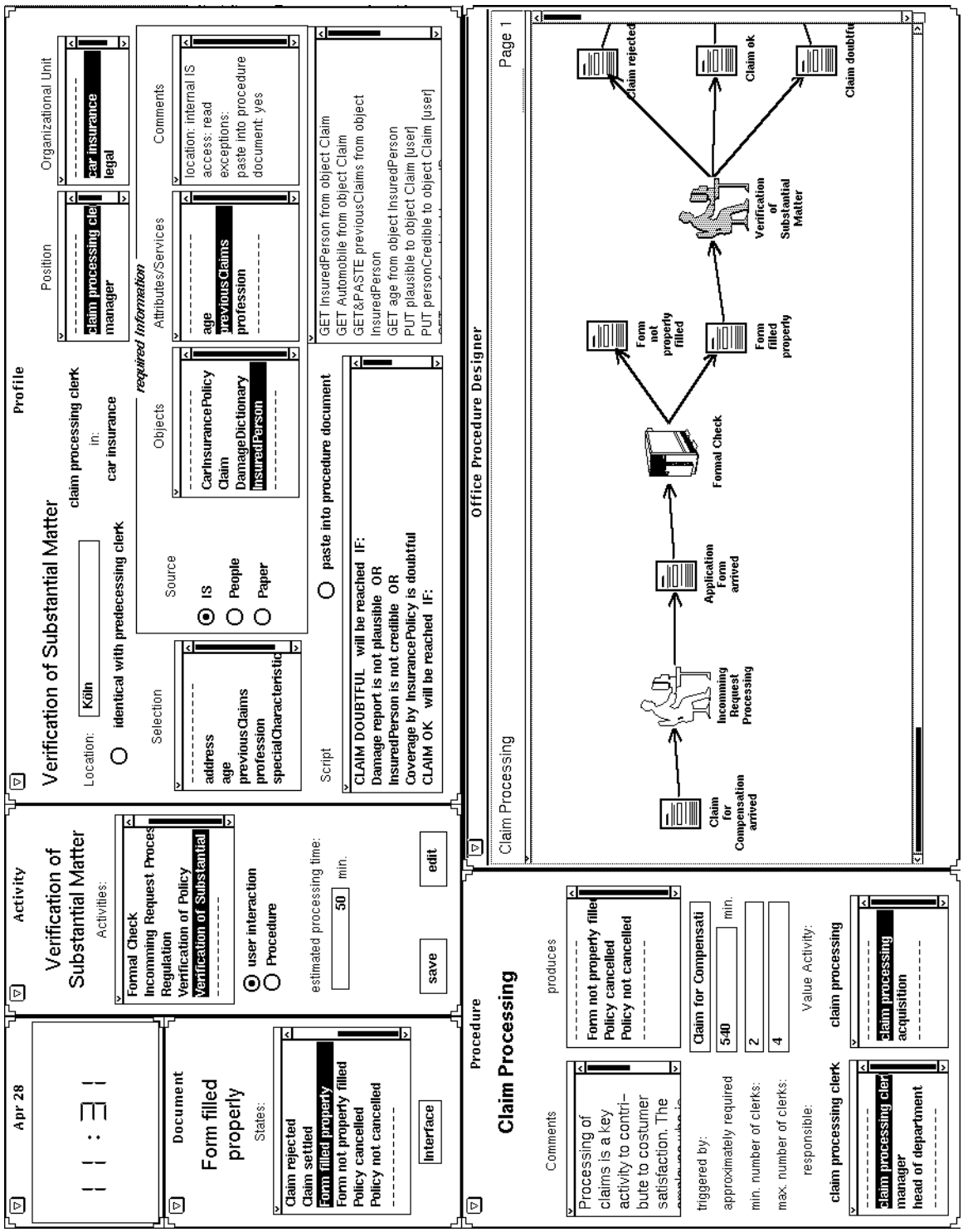

Figure 7. User-interface of the Office Procedure Designer

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.

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.

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 is already implemented. On the conceptual level it mainly consists of three classes: procedure manager, activity block, and procedure document. 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 “ActivityBlock” (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 IMPLEMENTATION OF THE DESIGN ENVIRONMENT

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

- 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.
- Objectforms®, an interactive interface-builder that allows to generate an interface from a formal description. Different from Tigre® it uses Smalltalk´s model/view/controller concept.
- Analyst®, a desktop publishing system that supports hyper-documents.
- NEDT®, a class library that supports the development of customized graphical editors.

All these tools reside in one Smalltalk image which allows for a high level of integration. Implementing the current version which was accompanied by modelling the car insurance department of an insurance company took less than two man years. Development was done on Sun workstations. To port the environment to another platform (like Macintosh or MS/DOS-Windows) it is sufficient to transfer the image file and have it interpreted by the target-machine´s Smalltalk interpreter.

6 CONCLUSIONS

Our experience with modelling an office domain within an insurance company has shown that the proposed representation of office procedures offers a suitable level of abstraction for enterprise modelling. Not only that it allows to add dynamic (respectively temporal) semantics to the model. 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. While Meyer´s optimistic claim “the objects are just there for the picking” ([18], p. 51] may be true for those of us who are object-enthusiastic software engineers it does definitely not reflect the way the average user perceives his domain. Instead 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 designing a dynamic model it also provides a heuristic to shape the static object model.

There is still a lot of research to be done. In order to refine the domain model we have built so far it is necessary to apply the approach to other domains, preferable car insurance departments within other insurance companies. Our work has been primarily concentrated on object models and office procedures. Other levels of abstraction proposed in the conceptual framework (within the organizational and the strategic view) recommend a less formal representation. Knowledge that is described in textbook-style could be added using the already available hypertext features. We also plan to enhance the system´s capabilities to simulate and thereby evaluate alternative ways to organize procedures. A graphical representation of an object model is intended to be more illustrative than a mere textual representation. Large models however become complex and are not easy to survey anymore - specially when you allow for an unlimited number of associations. We plan two measures that contribute to more clarity: a “pretty-print” algorithm to rearrange the icons and filters that allow to restrict the classes of associations that are shown.

Although office procedures are an illustrative metaphor it is not sufficient to describe all kinds of work in the office. Certain tasks (like comparing the records of certain customers) can be regarded as short procedures (consisting only of one activity block). Unstructured

cooperative work however requires other concepts as well as another graphical representation. It would be interesting to complement the OPD by a tool that allows for illustratively modelling CSCW-applications (for an example on the instance level see [9]).

Reusability as well as integration cannot be accomplished by isolated research. There is not one best design that has to be discovered. Instead there may be numerous good designs. Wide range reusability therefore heavily depends on commitment to one or at least a few designs. The commitment we think of implies participation. A preliminary model of a certain enterprise type could serve as input for discussing and refining the model within a larger community. Although comparable projects [8, 19] recommend not to be too optimistic we think this vision is too attractive to give it up.

References

[1] 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

[2] Booch, G., *Object-oriented design with applications*. Benjamin/Cummings 1990

[3] 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*. Springer 1984, pp. 19-47

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

[5] Coad, P.; Yourdon, E., *Object Oriented Design*. Prentice Hall 1991

[6] 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

[7] 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*. North-Holland 1991, pp. 303-321

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

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

[10] 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

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

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

[13] Graham, I., *Object oriented methods*. Addison-Wesley 1991

[14] 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

[15] Hogg, J.; Nierstrasz,O.M.; Tsichritzis,D., "Office Procedures", in: Tsichritzis, D. (Ed.), O*ffice Automation*. Springer 1985, pp. 137-165

[16] 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. North-Holland 1987, pp. 197-208

[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. ACM Press 1988, pp. 46-54

[18] Meyer, B., *Object-Oriented Software Construction*. Prentice Hall 1989

[19] 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

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

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

[22] Schank, R.C., *The Cognitive Computer. On Language, Learning and Artificial Intelligence*. Addison-Wesley 1985

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

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

Acknowledgements

I wish to thank all the colleagues who have been participating in the project “Computer Integrated Enterprise”, specially Matthias von Bechtolsheim and Stefan Klein.