Published in: Nunamaker, J.F.; Sprague, R.H. (Ed.): Proceedings of the 27th Annual Hawaii International Conference on System Sciences. IEEE Computer Society Press 1994, p. 663-672

An Object-Oriented Methodology for Analyzing, Designing, and Prototyping Office Procedures

Ulrich Frank

German National Research Center for Computer Science (GMD)
D-53757 Sankt Augustin, Germany. E-mail: ulrich.frank@gmd.de

Abstract

The paper presents a methodology for designing office procedure models in interaction with object models. The proposed models of office procedures are conceptualized in a way that helps to analyze the organizational effectiveness of procedures. Furthermore the methodology supports identification, specification, and refinement of classes in the object model. Integration is accomplished by establishing references from the office procedure models to the object model and by specifying the management of a procedure within classes of the object model. The specification can be partially transformed into a frame-based object definition language which in turn can be compiled into executable code. The models are thought to be located within an enterprise-wide repository. The methodology is supported by a design environment that provides various levels of abstraction - not only for system designers but also for domain experts.

1: Introduction

In 1990 the project 'Computer Integrated Enterprise' was launched at the German National Research Center for Computer Science (GMD) in order to develop a framework along with a tool environment to support the design of enterprise models. Such models should not only foster integration and reusability of software components that capture a high degree of domain level semantics (as it is proposed for instance in [21]) but also provide a medium that promises to reduce the well known communication barriers between the various perspectives on the enterprise and thereby support business planning and organizational design (like it is demanded in [5, 14, 23]).

An object-oriented approach was found to be best suited for this purpose - for widely accepted reasons: Usually objects offer a more direct and natural correspondence to real world entities than data structures. Inheritance and encapsulation promote reusability of concepts and components. The various parts of a corporate-wide information system can be integrated on a higher level: In order to communicate they may refer to domain level classes instead of (structured) data types. It is however not sufficient to model an enterprise by only focussing on objects or classes. From a software engineering point of view it is desirable to enhance an object model with dynamic constraints in order to foster system integrity. Furthermore models of business processes in general, of office procedures in particular provide a view on the enterprise that is essential for understanding the business and (re) designing organizational structures. Finally we found that objects are not always the preferred conceptualization for business professionals to describe their perception of reality. Often they rather choose a procedural view for understanding and explaining a domain within the enterprise. For this reason it is more promising to interview them about procedures - and use such an interview as a vehicle to find relevant objects (and their features respectively).

Although special approaches to design office procedure systems (like [4, 15, 24]) were very helpful for our work, none of them was completely satisfactory. While they aim at providing convenient (that is on a high level of abstraction) means to define the formal constraints of office procedures in order to support a reliable implementation or allow for fast prototyping respectively (which is important for enterprise modelling as well) they usually do not include the integration with an object model - if they are object-oriented at all (for one exception see [10]). Furthermore they typically do not include means for organizational analysis or design. Approaches on the other hand which suggest office procedure models in order to support organizational analysis and design (like [12]) usually neglect the requirements of system development. So it seemed to be most promising to adapt one of the existing methodologies for object-oriented design that include means for defining dynamic models by introducing state transition diagrams (like [1, 22]). However, for our purpose these techniques have two shortcomings. They only describe the permissible behavior ('life cycle') of objects of *one* class. Within an office procedure however one typically deals with a variety of different objects which interact in a particular context.

Furthermore state transition diagrams do not provide a representation that fits the average user´s perception of a business procedure.

The methodology and design-environment that is described in this paper is intended to integrate the various aspects of modelling office procedures. In particular 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. It supports identification, specification and refinement of objects (classes) and provides means to analyze and refine the effectiveness of office procedures. Last but not least it demonstrates how office procedure models and an object model can be smoothly integrated within an enterprise-wide repository. First I will give an overview of the underlying object model. After that the proposed conceptualization of office procedures is described in detail. In the remaining section of the paper it is demonstrated how to apply the methodology and a design environment that is based on it.

2: The Object Model

While the benefits of object-oriented software development are getting widely accepted there is no common understanding of how to define objects - neither on the specification or implementation level nor on the conceptual level, which is of outstanding importance for our point of view. Among the still increasing number of object-oriented design methodologies (in a survey we did last year we found more than forty approaches) we felt most inspired by the ones proposed by Booch [1] and Rumbaugh et al. [22]. Nevertheless none of the two was satisfactory for our purpose. While Rumbaugh et al.' methodology suffers from being somewhat superficial and not consequently object-oriented (for instance: attributes are defined by data types and not by classes) Booch's approach seems to be overloaded by details of various programming languages - which, on our opinion, should not be part of a general methodology for the design of *conceptual* models (for a comprehensive comparison of both methodologies see [9], for other comparisons see [11, 17]). When we decided to develop yet another meta model for the design of object models we wanted to take advantage of the benefits provided by all three roots of object-oriented software development, namely programming languages, data modelling, and Artificial Intelligence. In particular we put special emphasis on requirements like integrity, event-handling, business-rules, and user-interface modelling.

2.1: 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. Within an object model one defines classes and associations between classes or between objects respectively. Like in most other methodologies the outstanding features of a class are attributes and services. An attribute is regarded as an object that is encapsulated within an object. We do not allow attributes - like [3] - to only hold references to external objects that have an existence of there own in the object space. An attributes semantics is primarily defined by its class. Furthermore a cardinality (using min-max notation) may be assigned: A costumer may have no or many telephone numbers, but he has exactly one date of birth. Assigning a default value allows for generating appropriate initialization operations. 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 allowed to define write-permission to be greater than read-permission. Each attribute is also characterized by a history-flag. Setting it to true means that every update should be recorded somehow.

In order to allow for generating prototypical user-interfaces it is possible to assign a default-view to each class. A default view is either a widget or a collection of widgets. One can also define a label that is to be presented with the default view. Additionally features like size and font may 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.

In order to specify a service the designer may describe a list of input-parameters (which can be empty) where each parameter is characterized by its class and its name. If a service returns a result it has to be exactly one object. So it is sufficient to define the class of this object. It is important to note that such an object may be a composed object (like an array, a container etc.) that contains many other objects. A precondition in general specifies conditions "under which a routine will function properly" ([16], p. 114). In our model it can be defined by referring to object or parameter states. For instance: For a service that requires an object of class 'Person' as a parameter it may be necessary that the attribute 'sex' is in state 'male'. A postcondition has to be fulfilled after the service has terminated. Similar to a precondition it can be defined by referring to an object state or to a state of the object that is returned by the service. Access permission is specified by one of three authorization levels ('public', 'protected', 'private'). Each service can be assigned a set of exceptions (like media errors) which should be named in a unified way for a whole object model. Thereby exception handling can be defined for all involved

objects in the same way.

During the life time of an object there may be certain events and rules which go beyond the scope of a single service - otherwise they could be defined as pre- or postconditions. For this purpose we introduce triggers and guards. A trigger can be generally defined as a tupel consisting of an event and an action. The event is specified by a condition that in turn is defined by referring to attribute states or states of objects which are returned by a service. The action is defined by the service that has to be executed when the event occurs. To give an example for a trigger: Whenever an object of class 'CostumerAccount' has a balance less than x, the object should execute a service that is suited to notify somebody who is managing the account. A guard is a condition that has to be fulfilled during the lifetime of any object of a particular class (similar to what Meyer ([16], p. 124) calls "class invariant"). For instance: The value of attribute 'retailPrice' within objects of class 'Product' should never be lower than the value of attribute 'wholesalePrice'.

Every class may have exactly one superclass. Although there is a number of arguments in favor of multiple inheritance we restricted our model to single inheritance. We found that in most cases single inheritance is satisfactory while multiple inheritance increases the complexity of an object model and thereby makes it more difficult to maintain it in a consistent way. 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. Every class as well as any of its features can be annotated by a comment which is of class 'Hypermedia'. This class is used to express that a comment can consist of anything like text, sound, pictures, video etc. Furthermore it may include links to other objects.

2.2: Associations

While generalization describes a relationship between classes there are relationships or associations on the instance level as well. Objects within an information system are interrelated in various ways: they 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 information system. Therefore they are commonly regarded as an essential part of an object model. There is however no consensus on how to describe them. Modelling associations as objects (as it is suggested by [22]) allows to assign certain characteristics that do not belong directly to one of the interrelated objects to the association. For instance: If two objects of class 'Person' are connected by an association 'is married to' then an attribute like 'dateOfMarriage' could be located in the association object. After we first had adopted this approach (see [8]) it became evident that the 'everything is an object'-idea has its shortcomings. We found that one particular advantage of object-oriented design is to establish a correspondence between real world entities and objects and then describe associations between them. Giving up this differentiation leads to conceptual confusion sometimes.

So we finally decided for modelling associations according to Booch's methodology. Booch ([1], pp. 88) distinguishes between only two types of associations: interaction ("using") and aggregation ("is part of" or "contains"). Furthermore our model only allows for binary assocations. Each class (which actually represents a number of its instances) involved in an association has to be assigned a cardinality using min-max notation.

From a software engineering point of view aggregation and interaction are sufficient. For a conceptual object model to be illustrative however it is desirable to allow for a more detailed differentiation. For this reason each association has to be assigned a domain level identifier. Such identifiers do not include any semantics they only improve readability of the model and allow for enhanced retrieval capabilities. For instance: If one is interested in a part of an object model that represents organizational concepts, identifiers like 'supervises' or 'is in charge of' could be used to filter the associations which are relevant for this purpose. In order to express an associations direction its type and each identifier can be supplemented by the inverse type (i.e. 'is part of' for 'contains' et vice versa) or an inverse identifier respectively. Fig. 1 gives an overview of the concepts proposed for designing object models.

3: Modelling Office Procedures

A methodology for modelling office procedures should allow for conveniently expressing temporal and control (in other words: dynamic) aspects of structured tasks in the office. It should also help to avoid inconsistencies, like non terminating cycles, subtasks which cannot be reached by any chance, deadlocks, etc. As already mentioned above a procedural view is of special relevance for analysis - for refining the object model as well as for evaluating organizational effectiveness. Therefore we will first look at concepts which have been introduced in order to support analysis.

3.1: Analysis and preliminary Design

In order to enhance object models with dynamic semantics some object-oriented design methodologies (like [1, 13, 22]) include state transition diagrams. At first sight

such diagrams seem to be appropriate for modelling office procedures, since they allow to describe events and corresponding state changes. They are however restricted to events and state transitions which may occur during the lifetime of objects of one class. Therefore they are not satisfactory for our purpose: Within an office procedure one usually needs objects of more than one class. Peters and Schultz [20] propose a modelling technique that allows to include objects of more than one class. They use Petri nets where each transition represents a state transition of an object of a particular class. Different transitions within a net may represent operations of objects of different classes. Furthermore they allow - different from traditional concepts (for an overview see [2]) - transitions to have an execution time larger than zero. Mapping transitions to operations of an object of a certain class is attractive from a software-engineering point of view since it supports the idea of building procedures by 'glueing' objects together (as it is proposed in [18]). Nevertheless such an approach has its deficiencies, too. It is important for a model to allow for direct correspondence to familiar conceptualizations of the relevant domain. Office procedures are not necessarily structured in a way that there is always only one object operated on within a subtask. In addition to the operation of an object Peters and Schultz suggest to assign resources to a transition, like 'outgoing telephone line' - which is to indicate that these resources are needed for accomplishing the task associated with a transition. Adding such a category is certainly helpful for the purpose of organizational analysis.

The approach we have chosen is similar to the one suggested by Peters and Schultz in that we also use semantically enriched Petri nets and allow transitions to have an execution time larger than zero. Our methodology however allows to explicitly assign objects of many different classes to one transition and it provides categories to structure resources. An office procedure is modelled as a directed graph of activities (which are represented as transitions in a Petri net). Each activity is triggered by a certain state of all the relevant information and it produces one or more new states. In order to describe IS-objects and other information sources at the same time we use a special object that we call a 'virtual procedure document'. It contains references to all the objects needed within a procedure. These objects are either IS-objects or external objects. An IS-object is an instance of a class specified in the object model. Its state resides in the corresponding computer system (for instance in an object-oriented database management system). External objects reside outside the computer system. They are typically - but not necessarily - paper based. In order to include them in the model we use reference objects. They provide a description of relevant features of external objects - like attributes/fields or certain states. For instance: A form may contain fields like 'name', 'dateOfBirth' etc. and it may have states like 'complete', 'incomplete', 'consistent', etc. Furthermore a reference object can hold information about the physical location of an external object, its costs, availability etc. Whenever the objects a procedure document contains reach a relevant state (which is either monitored by the procedure document itself or - in the case of external objects - indicated by the user) it triggers an activity.

An activity is modelled by assigning the needed and produced information, its organizational context, the re-

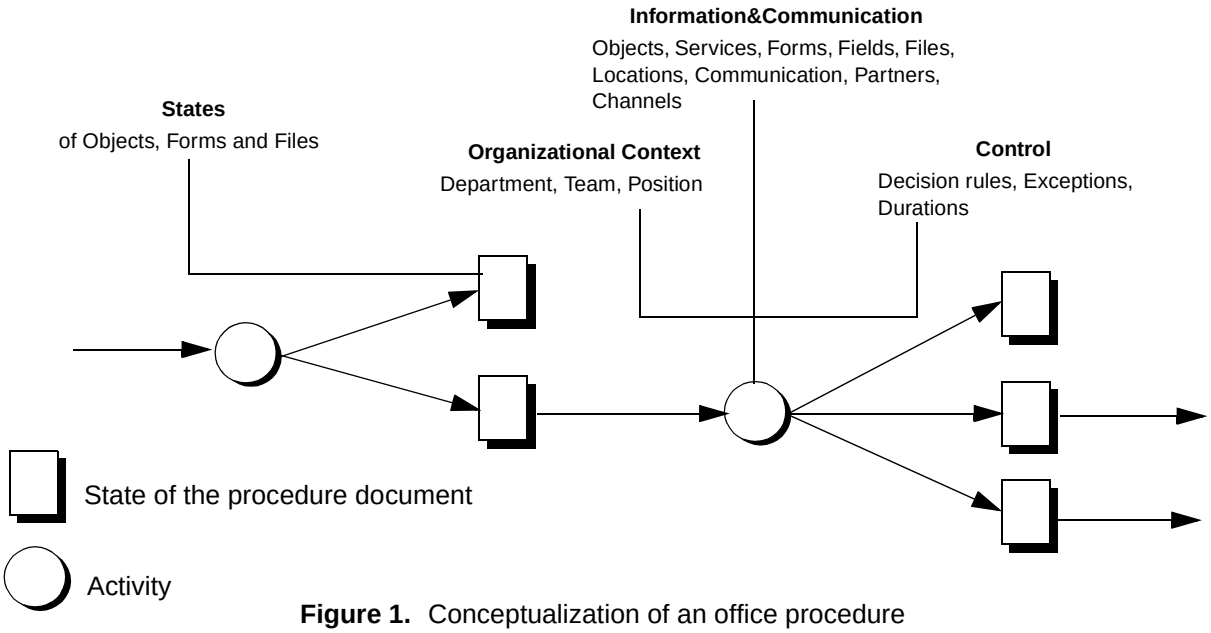

Figure 1. Conceptualization of an office procedure

quired (human) communication, control, exceptions, and execution time. Information that is located in IS-objects is referred to by the services which are needed. For each service that requires input parameters it can be specified where the input stems from (other objects, the user himself, or the user via communication with others). External objects are divided into two categories: forms and files. Forms have a formal structure, that is they contain fields, have well defined states, and a set of constraints defines the permissible operations. Their content may be changed within an activity. The term file is used to summarize documented information that is read-only - like office files, letters or journals. The organizational context of an activity is defined by assigning an organizational unit (like a department) and a position (like a clerk) which should both be represented in the object model. Communication is described by sources, communication channels, condition and purpose. Source is either a person, represented by a position, or an organizational unit. Communication channel is specified by picking items from a list that contains 'face to face', 'telephone', 'telefax', etc.

If the communication does not have to occur a condition can be defined in a semi-formal way. The purpose of the communication can either be characterized as input for an information object (internal or external) or for a decision. Control should be expressed in a semi-formal way by using declarative rules, referring to objects, object states and services. Exceptions should be named in a unified way. Exception handling can be modelled by describing the action that should be taken - again in a semi-formal way. Execution time can be assigned in three different extensions: estimated time, minimum time and maximum time.

In order to allow for different levels of abstraction each activity can be modelled as a procedure itself. The information represented by the model supports analysis in various ways. Asking for the required objects and their services respectively helps to complete and refine the object model. By relating required information to sources and communication channels it is possible to analyze the model for media frictions (like a person's age has to be transferred from an object within the information system to a paper-form), which may give hints for improving the procedure's organization. The communication with roles and organizations allows to generate a communication net - for a particular activity as well as for the whole procedure. Such a net can then be used to find communication patterns and thereby deliver hints for improving organizational effectiveness. The times assigned to each activity allow to calculate worst case, best case and average throughput - which could be implemented as a simulation. Furthermore the execution time of an activity could be varied by changing its capacity, which could be expressed either by assigning other execution times or adding parallel activities. Usually there is more than one type of office procedure within an enterprise. Procedures of different types may be interdependent by requiring each others results directly or competing for resources. Models of all procedure types together with figures about the expected workload and the available resources offer the chance for an enterprise-wide optimization of office procedures or at least allow to reveal chances for a more economic allocation of resources.

3.2: Prototyping and Implementation

Analysis and organizational design do not only result in an elaborate description of requirements but in a preliminary design. The dynamic structure of the relevant office procedures is described by Petri nets. Furthermore it should have been decided to what extent the information used in a procedure is to be represented in the object model. So far we have however abstracted from issues directly related to the implementation of an office procedure. For this purpose it is crucial to decide where to locate the knowledge necessary to manage a procedure. One important question in this respect: How should activities be represented/implemented within an object-oriented system? Additionally there is more generic knowledge to be taken into account. It 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 activity that cannot be reached by any chance.

- 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.

While general constraints should be checked already during the design process (preferable by an appropriate tool, see 4.), dispatching and exception handling can only be applied when the procedure is active. For this purpose we use a object that we call a procedure manager for each procedure type. It is initialized with the Petri net that contains the pre- and postconditions for each activity. Since more than one instance of a procedure type may be active the procedure manager allocates and distributes the resources needed within the activities. It also provides an in-

terface to an active procedure. For instance: It contains services that list all its active procedures and that allow to investigate those procedures' states. A procedure manager registers with the procedure supervisor, an object that is responsible for allocating and distributing resources requested by different procedure managers. It provides services that allow to browse through all active procedures of an enterprise (or department) and to investigate them by requesting the required services from the appropriate procedure manager. An activity itself is represented by a service of the procedure document that is defined for each procedure type. Its precondition has to correspond to one of its predecessor's postconditions. Its postconditions are combined with a trigger that is to notify the procedure manager. Any exception that occurs during the operation that implements the service is active causes a notification of the procedure manager as well. The operation itself uses other services of the procedure document and of other objects that have been assigned to the activity.

While there is no doubt that the proposed architecture can be implemented somehow it is desirable to take advantage of reusable artifacts as much as possible. At the beginning of the project we were committed to the vision already mentioned above: composing an office procedure solely from pre-existing classes. We had to give up this vision soon - which is not surprising after all: It is an essential feature of an office procedure (in other words: a long transaction) that a single activity or transition cannot be treated independently from other activities. Nevertheless an object-oriented approach offers starting points for reusability through specialization and variety. Designing a new type of office procedure goes along with specifying corresponding classes for the procedure document and the procedure manager. Three general classes are pre-specified and implemented: 'ProcedureSupervisor', 'ProcedureManager', and 'ProcedureDocument'. They are specified according to the meta-model described above. Among others they provide services to check a procedure's state, to support exception handling and to arrange for dispatching. Corresponding objects for a specific domain or procedure type can be defined by specializing from those general classes. With an increasing number of existing classes it becomes more likely to find a class that is very similar to the one actually needed. Reusability is additionally fostered by the fact that the operations required within an activity intensively use services provided by classes defined in the object model.

In order to provide the model with prototyping capabilities it is necessary to enhance it with information about a suitable user-interface. This information can be deducted from those services of objects assigned to an activity which are needed for user-interaction. The widgets needed to interact with the services can be looked up in the object model: input and output parameter of any service should be specified by a class. Since every suitable class in the object model should have a default view assigned to it, a prototypical user-interface for an activity can be generated. It may look somewhat awkward if there is no additional information on how to arrange the widgets. But by providing an interactive interface-designer the user of the prototype could rearrange the widgets (see 4:).

As already outlined above the procedure models are integrated with the object model in two ways. First they refer to the objects they use, second the management of an office procedure is done by objects that are specified within the object model themselves. Fig. 2 shows how an object model and office procedure models could be represented within

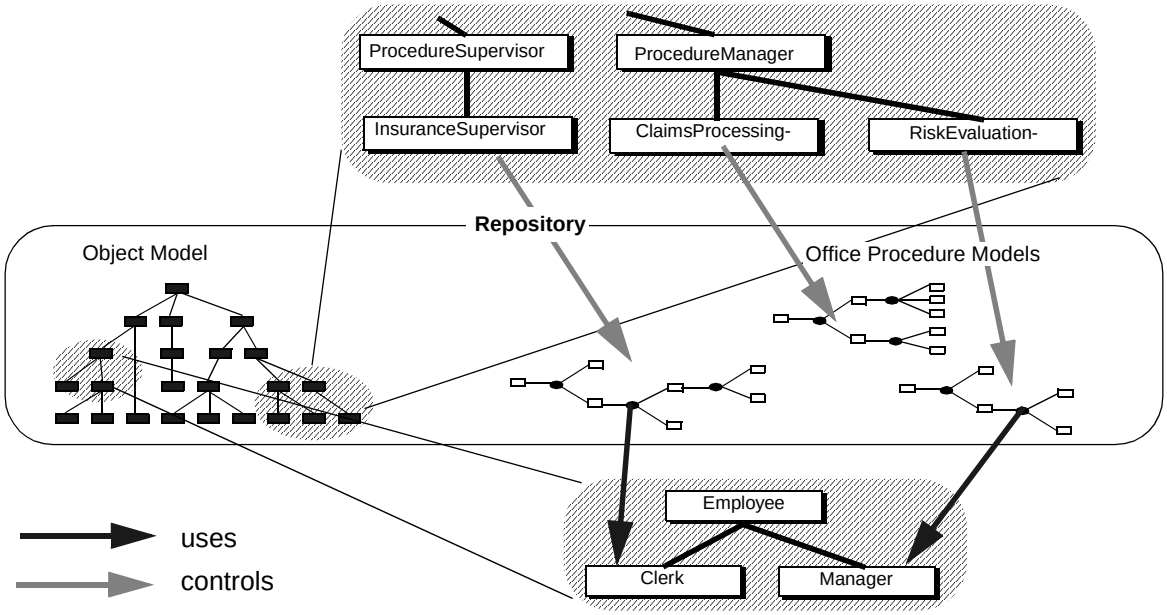

Figure 2. Integration of object model and office procedure models within an enterprise-wiede repository

an enterprise-wide repository.

In order to allow for a smooth transition from the object model and the procedure models to an implementation an appropriate runtime system is required. Furthermore it is necessary to provide a suitable language that can be used to code those parts of the specification which are not covered by the conceptual model. The most preferable solution would be to use an object-oriented database management system (OODBMS) - or, more appropriate on my opinion: an object management system - where the repository would serve as the schema for the OODBMS. However, working with Smalltalk (see below) we were not satisfied with most OODBMS we evaluated two years ago. The languages they use for schema definition did not provide the constructs which would have allowed to conveniently map from our object model to the schema. Smalltalk itself 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 [6]. The conceptual description of a class can partially be transformed in SFK´s object definition language (primarily attributes together with their associated access services). SFK enhances Smalltalk with strong typing. Various types of constraints can be defined as well. Compiling a class goes along with generating code for implementing demons which act as guards and triggers. An example of SFK-code generated from the object model is given in [7].

4: A Design Environment to support the Methodology

Designing enterprise models according to the proposed methodology can hardly be accomplished without appropriate tools: A complex model requires support for browsing and searching. Because of multiple integrity constraints maintenance that is solely done manually jeopardizes a model's consistency to an unacceptable extent. Finally it is impossible to do without tools when prototyping is to be accomplished. For these reasons we developed an environment that supports analysis, design and maintenance. It was implemented using Smalltalk-80 within the Objectworks® 4.0 environment on Sun4-workstations. High productivity could be achieved by using additional class libraries (see [7] for more details).

The environment consists of two main tools: The Object Model Designer (OMD) supports the specification of classes. It provides various features to search for certain elements of an object model and to browse through the model. The Office Procedure Designer (OPD) allows for conveniently designing office procedures. It provides functions for simulation, fast prototyping and organizational analysis. Both tools are tightly integrated. The integration on the conceptual level has already been characterized. System integration is accomplished by locating both tools within one Smalltalk image. It is important to note that both object model and office procedure models are analyzed and designed concurrently: Specifying an office procedure requires to establish references to the object model and may also give hints to enhance or refine the object model.

To demonstrate how to use the environment in a more illustrative way we look at a little example: a claim processing procedure within an insurance company. We begin with designing the object model using the OMD. The OMD provides different levels of abstraction. On the highest level one starts with inserting class names which have to be grouped into categories. For instance: 'Employee', 'Manager', 'Lawyer', 'Person', 'InsuredPerson', etc. could be assigned to the category 'People'. One superclass can be assigned to each class. According to the meta model a class can be characterized by features like attributes, services, triggers, and guards. First only their names are added to listboxes. The top left window in fig. 4 shows how a class is described on this level of abstraction. In order to specify a single feature in a more detailed way, the user has to select it within the listbox. This will cause the corresponding window to pop to the front. In fig. 4 this is the window that contains the template for specifying an attribute. The example shows the attribute 'dateOfBirth'. Its class is 'Date'. In order to foster reusability the services which are provided by this class are shown in a listbox. If one of these services is needed for the class that is currently selected (which is 'InsuredPerson' in our example) it can be pasted to the services-listbox of this class. The OMD will then establish a reference to this service.

After a few classes have been specified the OMD allows to generate graphical representations of the object model. The generalization hierarchy can be generated for the whole model. It turned out however that such a hierarchy of models with more than only a few classes does not fit into a window anymore - even if it is expanded to cover the whole screen. Therefore it is possible to alternatively generate a partial hierarchy. Each node of this hierarchy may be expanded (its subclasses may be added) dynamically. Furthermore the associations of a class on the object level can be presented in another window (see bottom right window in fig. 3).

In order to foster system integrity it is not possible to type in a class name directly to characterize an attribute, a superclass, or a parameter. Instead it is required that the dictionary that contains all class names is updated first. Then the name can be pasted to the corresponding field. The OMD controls a number of integrity constrains. It prevents the user from deleting elements which are referenced by other elements, from assigning superclasses or classes

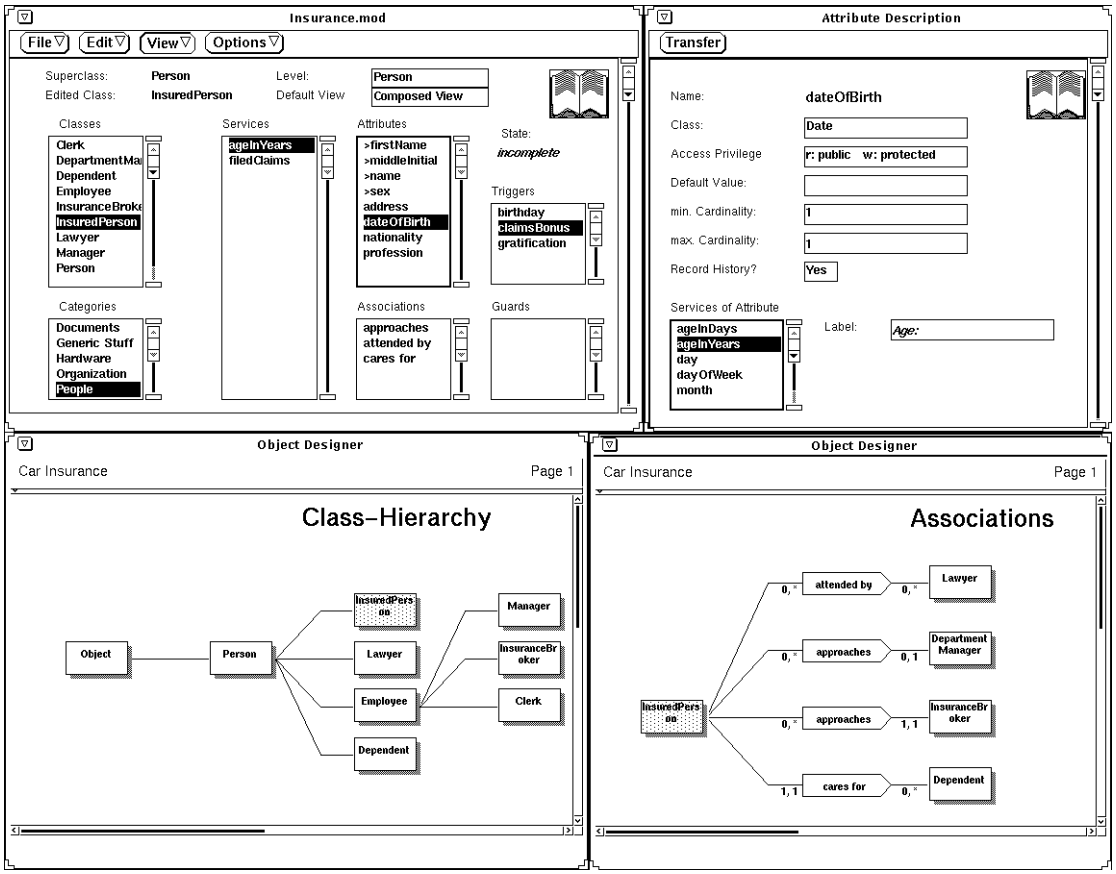

Figure 3. User-interface of the Object Model Designer

of attributes in an inconsistent way (for instance: it is not allowed to assign a superclass that is already a subclass), etc.

The state of a class description in the OMD can be characterized by one of three different values: 'incomplete', 'complete' and 'implemented'. As soon as the state is defined as complete by the user the specification is compiled - as far as possible - into SFK-code. Since the services' semantics is not completely formalized in the model only the code for type checking of parameters and for checking pre- and postconditions can be generated. After the user has completed the implementation of all the services of a class the state 'implemented' can be assigned. That means that objects can be instantiated from this class.

In parallel to - and in interaction with - developing the object model the model of the claim processing procedure is designed using the OPD. At the beginning - on the highest level of abstraction - the whole procedure is drawn using a specialized graphical editor. For this purpose the user is provided with illustrative icons which represent states of the procedure document and different types of activities. Before an operation to modify the net is committed the OPD checks if it is consistent. When the procedure is described on this level (see fig. 4), the user can 'zoom' into it by selecting an icon - either an activity or a document state. Within the example shown in fig. 4 the activity 'Verification of substantial matter' is selected. Within the window titled 'Activity' it can be characterized by assigning execution times, an organizational unit, an organizational position, and possible exceptions. Furthermore the classes (like 'InsuredPerson', 'Policy', etc.), forms, and files which are needed as well as the involved persons can be listed in this window.

In order to be more specific about an item it can be selected which will then cause a window with an appropriate template to pop to the front. Within the example shown in fig. 4 this is the window in top right position. It allows to pick the required services from the selected class. For each service - or the object it delivers respectively - it can be specified what it is needed for: either for a form or for communicating with an involved person. The example shows that the service 'profession' is needed for the 'ApplicationForm' as well as for communicating with the 'InsuredPerson' and the 'Expert'.

Other templates exist to specify the relevant information within forms (fields and what they are needed for) or files (where are they physically located, what is the subject of interest within the file) and as well as the communication with involved persons (channel, subject). These specifications are used to generate a protocol which is shown in the right bottom corner of the 'Activity' window in fig. 4. Another text-widget within this window presents a template that can be filled to describe decision rules relevant for the

focussed activity. A selected document state is specified in a similar way. First the classes, forms, and files which are involved in this state are assigned to it. After that these items have to be characterized in a more detailed way. The state an object (which is represented by the name of its class) has to be in, is defined by naming a boolean service that checks this state. That requires to enhance the class with this service by switching to the OMD. For instance: An object of class 'Policy' is required to be in state 'valid'. That requires a service like 'isValid' to be specified for this class. For every form it has to be defined which fields have to be filled in. A file is characterized by its physical location and optionally the means of transportation to get it to the clerk. In order to support the user and to avoid inconsistencies the classes, forms and files assigned to a document state are pasted to the activity that is triggered by this state.

The OPD can generate a prototypical user-interface for every activity - provided the default views have been assigned to the corresponding classes in the object model (see above). The widgets placed within such an interface may be rearranged interactively. In order to use the simulation features built into the OPD it is necessary to first assign probabilities (using percentage values) to the states produced by every activity. Furthermore it is required to type in the workload as number of procedures in a time period. After that the simulation can be started. It can be done in a various speeds. Animation is accomplished by dynamically marking the activities which are active during a certain time period. If the simulation reveals any chances to improve the organization of the procedure the net can be rearranged interactively.

The evaluation of a procedure's organization is further supported by a function that generates a report on detected media frictions as well as by another function that draws the specified communication relationships as a star where the clerk who is in charge of the activity is positioned in the center.

Both the OMD and the OPD allow to annotate most of their models' features with hypertext-comments. Furthermore they provide various retrieval capabilities. For instance: Searching for all the classes which are referenced within a certain class, searching for all the classes which are referenced within a certain office procedure, etc. The hypertext-annotations are completely inverted so that annotations which contain logical combinations of (truncated) strings can be retrieved within an negligible amount of time.

5: Conclusions

Different from general methodologies to design object-oriented information system the proposed methodology does not only allow to define object models on a high level

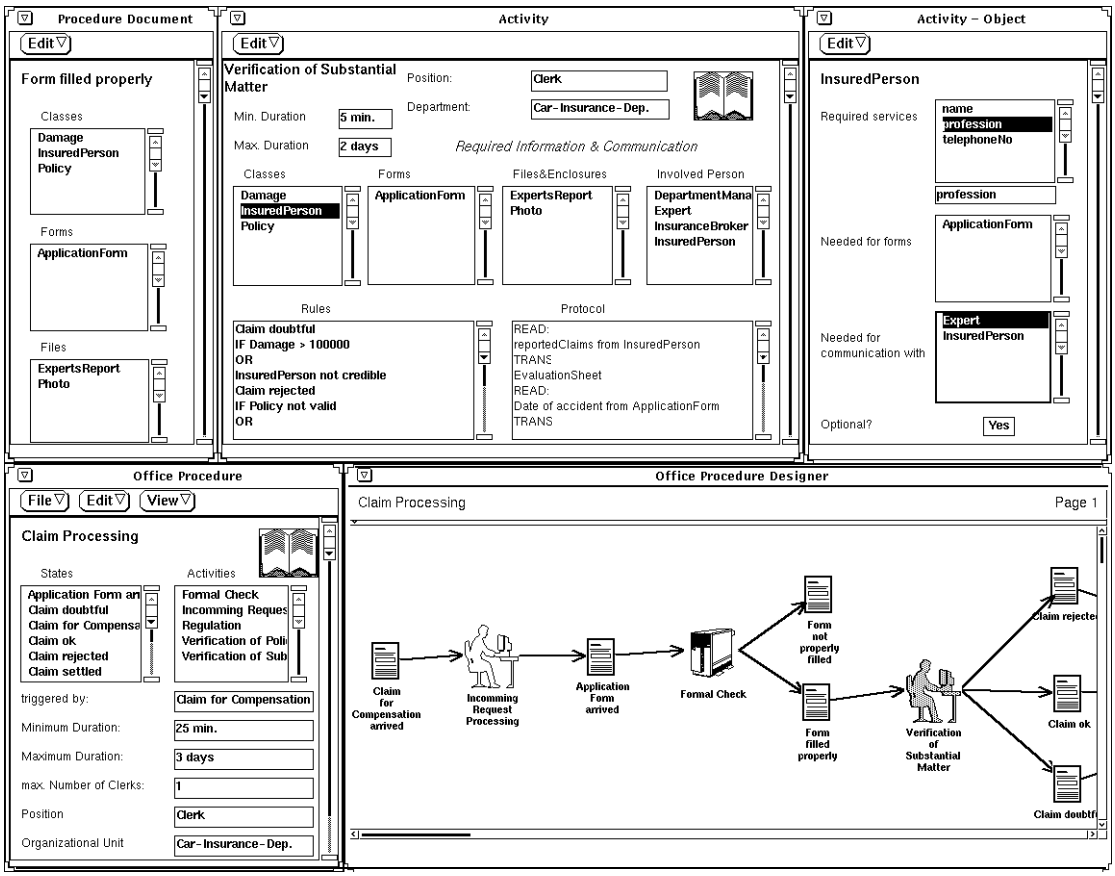

Figure 4. User-interface of the Office Procedure Designer

of semantics. Furthermore it supports the analysis and design of office procedure systems which are closely integrated with an object model. Both object model and office procedure models represent static as well as dynamic constraints of a firm's information system. They can be located in a central repository in order to allow for a high degree of integration an reusability. The environment is thought to be a tool for system designers. The experience we gathered so far indicates however that it also provides levels of abstraction which are suited to facilitate discussions with domain experts who are computer illiterate.

The frame-based object definition language we use on the implementation level provides powerful concepts. It is however not satisfactory in the long run: Distribution is an essential feature of office procedure systems. Although we have successfully experimented with a server/client-architecture using Smalltalk's C-interface and Sun's RPC-library it is definitely more desirable to generate a schema (or part of it) for a distributed OODBMS. We currently work on such an interface for ONTOS, an OODBMS we have been using within other projects for a few years. In the long run however it will not be sufficient to connect to one particular system. Therefore we are watching the few emerging standards for object models. So far we have specified how to map our object model to the one suggested by the Object Management Group [19] - which unfortunately is accompanied by a loss of semantics.

References

[1] Booch, G.: *Object-oriented Design with Applications*. Benjamin Cummings, Redwood 1991

[2] Brauer, W.; Reisig, W.; Rozenberg, G. (Eds.): Petri Nets: *Central Models and Their Properties*. Springer, Berlin, Heidelberg etc. 1987

[3] Coad, P.; Yourdon, E.: *Object Oriented Design*. Prentice Hall, Englewood Cliffs, NJ 1991

[4] Croft, W.B.: Representing Office Work with Goals and Constraints. In: Lochovsky, F. (Hg.): *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, p. 13-18

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

[6] Fischer, D.H.; Rostek, L.: Consistency Rules and Triggers for Thesauri. In: *International Classification*, Vol. 18, No. 4, 1991, p. 212-225

[7] Frank, U.; Klein, S.: *Three integrated tools for designing and prototyping object-oriented enterprise models*. GMD-Research Report, no. 689, Sankt Augustin 1992

[8] Frank, U.: Designing Procedures within an object-oriented Enterprise Model. In: Sol. H. (Ed.): *Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems*. Delft 1992, p. 365-387

[9] Frank, U.: *A Comparison of two outstanding Methodologies for object-oriented Design*. GMD-Research Report, no. 779,, Sankt Augustin 1993

[10] Hogg, J.: OTM: A Language for Representing Concurrent Office Tasks. In: Lochovsky, F. (Hg.): *Proceedings of the IFIP WG 8.4 Workshop on Office Knowledge: Representation, Management and Utilization*. Toronto 1987, p. 10-12

[11] Hong, S.; Goor, G.: A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In: Nunamaker, J.F.; Sprague, R.H. (Hg.): *Information Systems: Collaboration Technology, Organizational Systems, and Technolgoy. Proceedings of the 26th International Hawaii International Conferenc on System Sciences*. IEEE Computer Society Press, Los Alamitos 1993, p. 689-698

[12] Humphreys, P.; Berkeley, D.; Queck, F.: Dynamic Process Modelling for Organizational Systems supported by SA-SOS. In: Sol, H. (Hg.): *Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems*. Delft 1992, p. 1-36

[13] Kappel, G.; Schrefl, M.: Using an Object-Oriented Diagram Technique for the Design of Information Systems. In: Sol, H.G.; Van Hee, K.M. (Hg.): *Dynamic Modelling of Information Systems*. North-Holland, Amsterdam, New York etc. 1991, p. 121-164

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

[15] Kreifelts, T.; Woetzel, G.: Distribution and Error Handling in an Office Procedure System. In: Bracchi, G.; Tsichritzis, D. (Hg.): *Office Systems: Methods and Tools. Proceedings of the IFIP TC 8 WG 8.4 1986*. North-Holland, Amsterdam, New York etc. 1987, p. 197-208

[16] Meyer, B.: *Object-Oriented Software Construction*. Prentice Hall, Englewood Cliffs/NJ 1988

[17] Monarchi, D.E.; Puhr, G.: A Research Typology for Object-Oriented Analysis and Design. In: *Communications of the ACM*, Vol. 35, No. 9, 1992, p. 35-47

[18] Nierstrasz, O.; Dami, L.; De Mey, V.; Stadelmann, M.; Tsichritzis, D.; Vitek, J.: Visual Scripting: Towards Interactive Construction of Object-Oriented Applications. In: Tsichritzis, D. C. (Ed.): *Object Management*. Geneva 1990, p. 315-331

[19] Object Management Group/Object Model Task Force: OMG Architecture Guide 4. The OMG Object Model. Draft July, Framingham/Mass. 1992

[20] Peters, L.; Schultz, R.: The Application of Petri-Nets in Object-Oriented Enterprise Simulations. In: Nunamaker, J.F.; Sprague, R.H. (Eds.): *Information Systems: Collaboration Technology, Organizational Systems, and Technolgoy. Proceedings of the 26th International Hawaii International Conferenc on System Sciences*. IEEE Computer Society Press, Los Alamitos 1993, p. 390-398

[21] Pröfrock, A.-K.; Tsichritzis, D.; Müller, G.; Ader, M.: ITHACA: an Integrated Toolkit for Highly Advanced Computer Applications. In: D. C. Tsichritzis (Hg.): *Object Oriented Development*. Genf 1989, p. 321-344

[22] Rumbaugh et.al.: *Object-oriented Modelling and Design*. Prentice Hall, Englewood Cliffs/NJ 1991

[23] Sowa, J.F.; Zachman, J.A.: Extending and formalizing the framework for information systems architecture. In: *IBM Systems Journal*, Vol. 31, No. 3, 1992, p. 590-616

[24] Tsichritzis, D.: Form Management. In: *Communications of the ACM*, Vol.25, No.7, July, 1982, p. 453-478