Working with objects
A three-model architecture for the analysis of information systems

Professor Trygve Reenskaug, Taskon

The presentation discusses the merging of object oriented technology with the well-established data base technology in a third generation client-server architecture. The architecture is based on three kinds of models. *The first is an information model* describing the information used in the organization. It will usually describe a relational database, but object-oriented databases can be effective under certain circumstances. *The second is an organization model* describing the users and their work processes. This object-oriented model acts as a formal requirements specification; the required information can be deduced from the description of the users and their work processes. *The third is a tool model* describing the computer-based tools applied by the users to perform the tasks described in the second model. The tools will typically access the databases; the tool model is thus a specification of the operations that must be supported by the information model.

The architecture is based on the OOram method that is described in a new book: *Reenskaug, Wold, Lehne: Working with objects. Manning/Prentice Hall 1995.*

1 Introduction

Modern database technology has all the hallmarks of the successful technology. The relational model has sufficient flexibility for the expression of a large variety of interesting business phenomena; yet it has sufficient rigidity to offer great leverage. The result is that we can describe our business information in a very high level language such as the graphical NIAM schema language; and leave the programming of the application details to automatic code generators. [Elmasri]

If the relational model gets its strength from its sharp focus on information structures, it also gets its main weakness from the same source. Focusing on information structures leaves functional aspects out of focus. The automatic code generators assume a certain functionality, e.g., *get, put, select,* and *join*. If more sophisticated functionality is needed, the functionality has to be defined in a separate application program that is outside the scope of the information model.

Another limitation of the relational model is that it assumes the existence of a *global schema.* Different clients may see different *external schemas*, but these schemas are created from the one and only global schema by filtering operations. The same limitation applies to traditional, procedure oriented application programs. There is a *main program* that controls all functionality and also defines the *global variables* that are available everywhere. The idea is illustrated in figure 1.

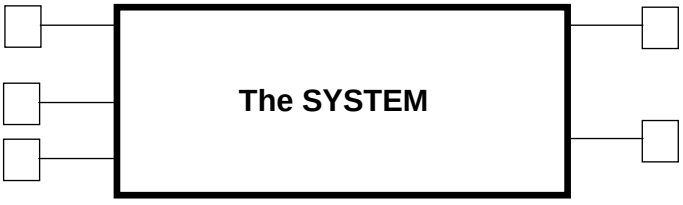

Figure 1: A centralized, procedure-oriented view of the world

The basic, usually unstated, assumption is that the system creator sees everything, knows everything, understands everything, and controls everything. These assumptions are valid in a large number of cases. The data centered approach and the 'global' paradigm are serving us well for developing many useful and interesting systems. But there are cases when the problem demands that we extend our scope.

Functional aspects can be too important to be treated as an afterthought; and we may want to see our different "global" application systems within a common context. We then have to change our fundamental way of thinking about systems. We need to find a way to combine data aspects and functional aspects without overloading our brain. We need to find a way that permits us to accept that the total system is too large to be comprehended as a whole; a way that permits us to build different models as the need arises. We need to find a way to formalize and generalize our insights and experience, so that we can build a solid library of components that we can reuse by composition and specialization to meet new and changing requirements.

The last point is an important one. We frequently hear the vulgar statement that "*change is the only stable characteristic of the post-industrial enterprise*". Taken as the whole truth, it leads to chaos. Taken as a half-truth, it is both interesting and challenging. The post-industrial enterprise has to be alert and continuously adapt to a rapidly changing environment. But the only way to do so is to identify a large, solid, and stable foundation that supports its rapidly changing surface. The foundation will consist of reusable models, change is supported by changing specializations of these models.

The object paradigm is the only known foundation that can support all the above requirements. The object combines data and function in a common, encapsulated entity. It supports generalization and specialization. It supports separation of concern and composition. But the most important characteristic is the distributed nature of the object system. This is illustrated in figure 2. The all-seeing, all-knowing system creator has disappeared. The object has responsibility, knows its collaborators, and is robust. Nobody needs to know everything, but we may study and understand any part of the total system that we care to consider. The object paradigm permits us to break the rigid boundaries of the centralized paradigms; it permits us to consider huge, truly global systems.

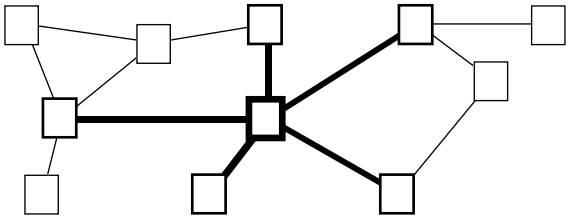

Figure 2: A decentralized, object-oriented view of the world

There is very little new here. As humans, we have always observed the world around us, we have always created models that help us understand and master it, and we have created different models for different purposes. A model cannot be good or bad in itself. If a model serves its purpose, it is a good model. If it doesn't, it is a bad model. We *choose* an applicable modeling paradigm, we *choose* the subject of our model, we *choose* what to include, we *choose* what to exclude.

We shall see how object technology enables us to work with many models with different foci while retaining overall control, and we shall see how the OOram method enables the practical application of such models. OOram (*Object Oriented Role Analaysis and Modeling*) is a mature method that has been applied to a variety of problems through more than ten years. It was formerly known as OORASS [Ree 92], the complete method has been published in [Ree 95].

2 A three-model system architecture

In this paper, we will *choose* a three-model architecture because it is interesting and helps us understand the new generation of distributed client-server systems. We have chosen to take the information model as our starting point, and extend our interest in two directions as illustrated in figure 3:

1. An *information model* describes information related to an interesting domain. There will be many information models, because it is impractical to cover everything of interest within one and the same model. Examples are financial models, budgeting and cost control models, materials management models, project planning and control models, computer-aided design models.

2. A *system user model* is a formal description of the human organization and its procedures. (We call it formal, because we do not model soft aspects such as inter-personal relationships.) There will be many system user models, because it is impractical to cover everything people do within one and the same model.
3. A *tool model* describes the physical interface between the user performing his or her tasks and the information models.

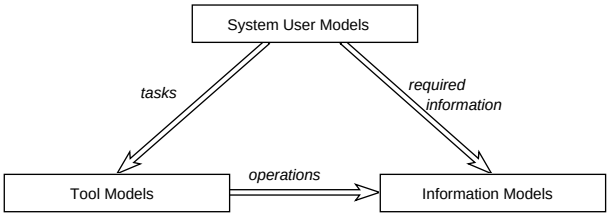

Figure 3: Three model architecture

The relationships between these three models is particularly interesting. Information models describe the subject of the system user model activities; they describe the users' universe of discourse. Given the System User Models, we can therefore specify the information to be represented in the Information models.

The people described in the System User Models need some kind of user interfaces to operate on the information. The Tool Models describe these interfaces. We use the term *tool* to denote the artifacts employed by the users to perform their tasks. The tasks are described in the System User Models, the tool specifications can be derived from these models.

The Tool Models describe how the users operate on the information. The operations that need to be supported by the Information Models are therefore specified by the Tool Models.

We will illustrate the three model system architecture by a simple example. We will first consider a system user model for the management of business travel. The focus will be on organization, division of authority and responsibility, and the dynamic work procedure. We clearly need an object oriented model to describe it all, and use the example as a vehicle for introducing the concepts and notation of OOram role modeling. We then quickly create an additional model for purchasing airline tickets. In section 6, we describe the idea of role model synthesis enabling us to derive composite models from any number of simpler base models.

We then move on to an information model covering our example requirements. We see that this model is basically a data model with almost no functionality. The information model can therefore be implemented by a relational database service. We finally consider the users' tasks, and give an example of a tool for the person who authorizes business travels. We see that a suitable tool needs access to a number of information services in addition to the obvious travel management service. A client-server implementation of the tools and information services is therefore appropriate even for this very simple example.

3 *TravelExpense:* A System User Model

We will now develop a System User Model, that is a model describing the business organization and its activities. It is clear that we cannot hope to create a model of *all* aspects of an organization; we have to focus on some aspect and ignore everything else. We *choose* to focus on the management of business travel, and we *choose* to focus on the people involved and their formal relationships.

The element of choice is important in all systems thinking. Regarding a system as an artifact of the mind, rather than as a solid fact in reality, prevents religious wars and facilitates the choice of useful models and modeling paradigms. The following definition of the object model is derived from the Delta project [Holbæk-Hanssen]:

> A *system* is a part of the real world which we *choose* to regard as as a whole, separated from the rest of the world during some period of consideration; a whole that we *choose* to consider as containing a collection of *objects*, each object characterized by a selected set of associated *attributes* and by *actions* which may involve itself and other objects.

3.1 The OOram object metaphor

One hundred years ago, Max Weber proposed a model for the rational organization of enterprises [Etzioni]. In this rule based, *bureaucratic* organization, each member has a clearly defined job with attendant authority, responsibility, and competence. The organization as a whole is structured for maximum efficiency, its members perform their duties strictly according to the rules.

Our organization-based object metaphor is illustrated in figure 4. We think of the objects as mechanical clerks. Each clerk has an *in basket*, and *out basket*, a book of *rules*, and a set of data *files*. An object (clerk) picks up a message from the in box and performs the steps specified in the applicable rule in the rule book. These steps can include reading and updating data in the data files as well as sending messages to other objects.

This simple metaphor covers some commonly stressed object characteristics:

1. *Identity*. An object has identity. An object retains its identity all through its life time. There has never been, and will never be, another object with the same identity.
2. *Encapsulation*. An object is encapsulated. Other objects can only access it by sending messages to it; they cannot observe its internal object construction.
3. *Polymorphism*. The object has its private book of rules. Different objects can handle messages differently according to their own, specific characteristics.

Humans are not automatons, and Max Weber did not live to see a perfect bureaucracy. But if he had been alive today, he would have seen it in the form of a structure of collaborating objects residing in a network of communicating computers.

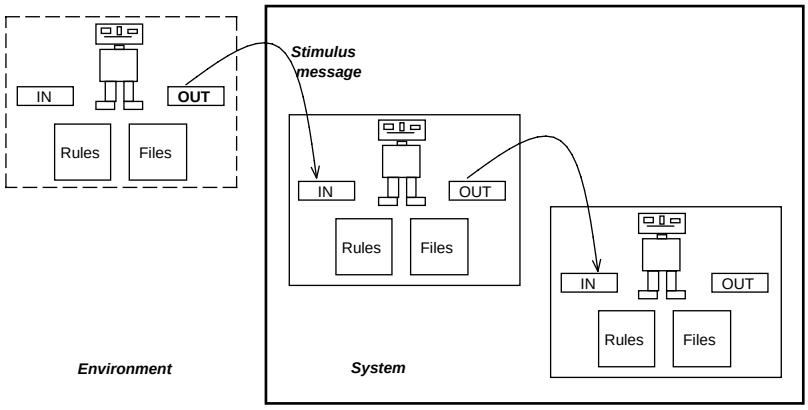

Figure 4: The OOram object metaphor: Clerks play roles in organization

One of the main differences between the old, "global" approach to system modeling and the new, distributed approach is that we regard our systems as *open systems* [HallFagan]:

> For a given system, the *environment* is the set of all objects outside the system whose actions affect the system and also those objects outside the system that are affected by the actions of the system.

A *stimulus* message is an initial message that is sent from an environment object to the system. The stiumulus message triggers an *activity* in the system, it consists of the actions performed by the objects and the messages sent between them. The final result of the activity is called the *response*.

3.2 TravelExpense object model

Applying this metaphor to our problem, we study the business organization and simply regard each person as an object. This is illustrated in figure 5, where we somewhat arbitrarily also show the *worksFor - reportsTo* relationships.

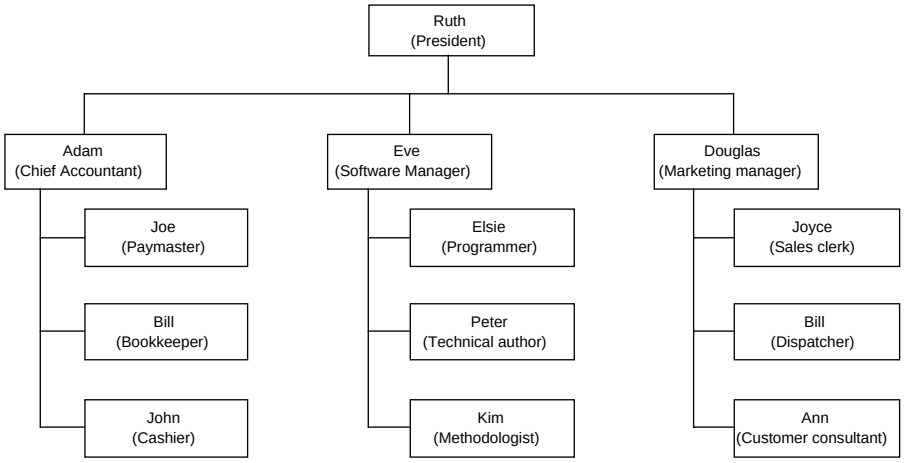

Figure 5: Company organization as an object structure

This figure can be interpreted in many different ways. The traditional way is to regard it as an organization chart; but we *choose* to let it denote that each member of the organization is modeled as an object in the sense of the figure 4 metaphor. This choice implies that we only study formal functions; we ignore all human aspects of the business organization. But even this is too wide; there are too many things going on in an enterprise to be represented in a single model. We elect to focus on the following *area of concern:*

> The area of concern is travel expense management. We focus on managing the trip, and do not model details about why the journey was made, nor how the traveler is reimbursed for his expenses.

We could create one model of each potential traveler: Ruth, Adam, Joe, Bill, John, Elsie, ... This is clearly unnecessary, we select a typical traveler and lets him represent all possible travelers. Let us focus on Peter and see what happens when he wants to make a trip. One possible process is illustrated in figure 6.

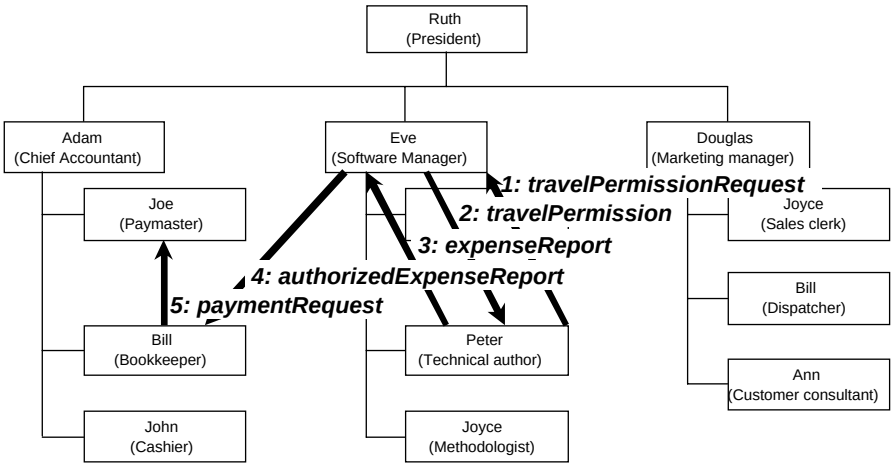

Figure 6: Typical expense report process

3.3 The role model abstraction

The model of figure 6 is clearly too specific. We need to make an abstraction; to describe the *phenomenon* of travel expense management independent of the person who happens to travel. The solution is well known to anybody who has written or followed a business procedure: we consider the *roles* people play in the context of the procedure. Mapping roles to people can be done in a very flexible manner at a later stage: a person can play many roles and a role can be played by many people.

We apply this idea to our study of complex object structures. In figure 7, we have identified the pattern and named the roles for our specified area of concern.

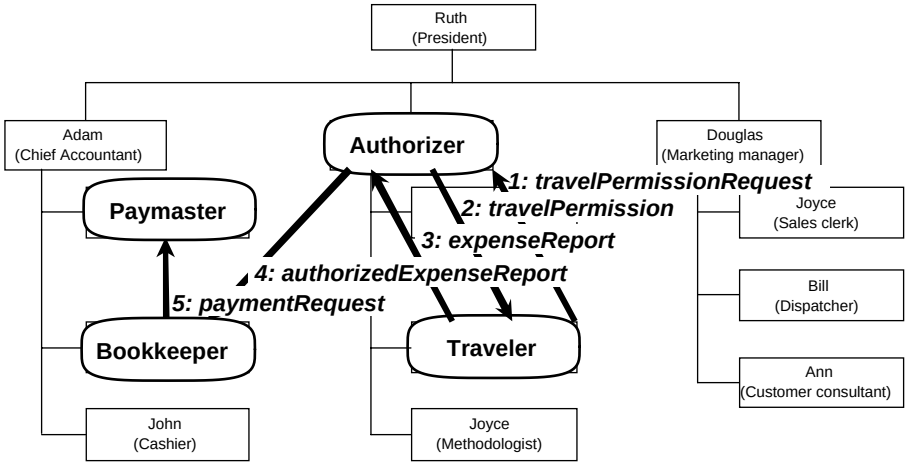

Figure 7: System user role model

Figure 7 is a somewhat untidy picture that illustrates the very powerful *role model abstraction.* We model a complex system as a structure of interacting objects. Our current example is a business organization, but we could just as well have considered a complex structure of interacting objects existing in a computer or even in a system of communicating computers. The total system is too complex to be comprehended as a whole, so we create simplified models by considering an *area of concern* and describe a pattern of *roles* that adequately represent it.

We select a notation and redraw as the *role model collaboration view* of figure 8. It shows the collaborating roles and their message communication structure.

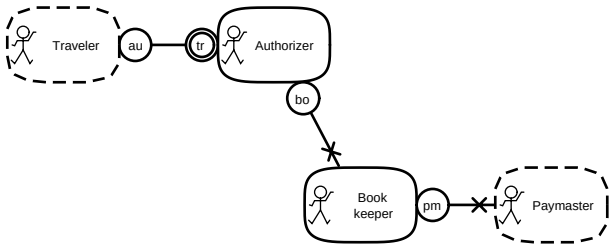

Figure 8: Collaboration view of System user model

The large, rounded symbols denote *roles*, the ones with dashed circumference denote *environment roles.* The lines denote message communication paths. The small circles denote *ports.* Associated with each port is the set of messages that the near role may send to its collaborator. A uni-directional path consists of a circle and a line to the collaborator role. To simplify the diagram, the path in one direction is superimposed on the path in the opposite direction. A single, small circle denotes cardinality of exactly one: the near role knows of exactly one collaborator role. A double circle denotes a cardinality of *any.* A cross denotes a uni-directional path, the near role does not send messages along the path.

The reason why the Traveler wants to travel is outside the area of concern. So the first, *travelPermission-Request,* message appears "out of the blue". It is a stimulus message and the Traveler is an environment

role. Intuitively, this feels wrong. Surely, the Traveler is an essential part of the model and cannot be a mere environment role? But the notion of environment objects is a technical one, and the fact that there is clearly more to the Traveler *object* than is represented in this Traveler *role* makes it an environment role.

The Paymaster role is likewise denoted as an environment role because it receives the final, *paymentRequest*, and the actions triggered by that message are outside the context of the current area of concern.

Objects describe static, data-centered properties as well as dynamic, process-centered properties. We cannot show all aspects of an object model in one diagram, and use different *views* to show different aspects of one and the same model. Some dynamic properties of our role model is shown in figure 9. This is a *scenario*, it describes one possible sequence of messages resulting from a stimulus message.

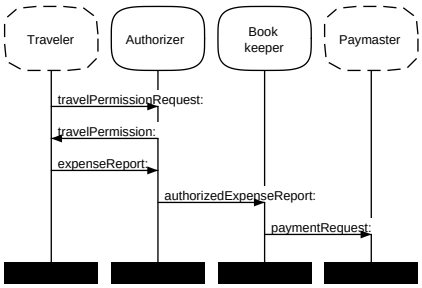

Figure 9: Interaction scenario of System User Model

The OOram way of modeling dynamic properties is related to the *Use Case* described by Ivar Jacobson [Jacobson]. A Use Case is triggered by a user action, an OOram activity is triggered by a stimulus message from an environment role. Both methodologies describe the actions by describing one or more examples in the scenario diagrams. Where we may differ, is that the OOram methodology defines a good model as a model where *both* the static, data centered properties and the dynamic, process centered properties are taken care of in a satisfactory manner.

The role model abstraction is useful for the study of large and complex object structures. In figure 10, we first select some real world phenomenon and represent it as a structure of objects. In a second step, we focus on a particular area of concern, and represent the relevant pattern of objects as a corresponding pattern of roles. The roles have all the properties of objects: they have identity, they represent data, they have behavior. We shall later see that role model inheritance not only reuses the properties of individual objects and classes, but it also reuses the dynamic properties of interacting objects.

1. We *choose* an area of concern, and disregard all objects that are not relevant in this context.
2. We further disregard all aspects of the considered objects that are irrelevant in the context of the chosen area of concern.
3. We generalize object identity, and represent similar patterns of interacting objects by a similar, archetypical pattern if interacting *roles.*
4. We use the object encapsulation property to hide roles that we consider to be of little interest within the boundaries of other roles.

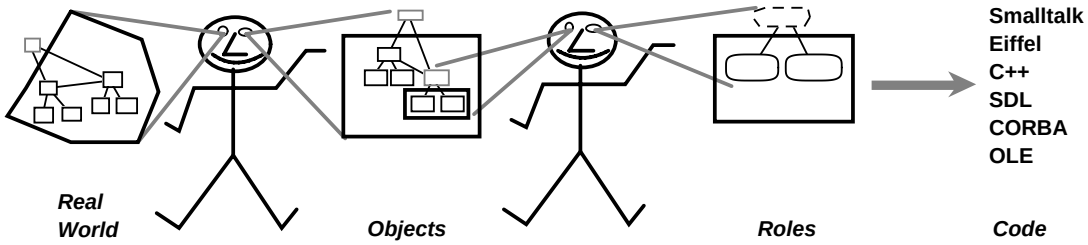

Figure 10: The OOram role model abstraction

3.4 Views and perspectives

The way we have defined that objects are encapsulated and that a system has an environment. This invites us to observe the system from different observation points as indicated in figure 11. The *perspective* depends on our observation point. The *view* depends on what we want to see.

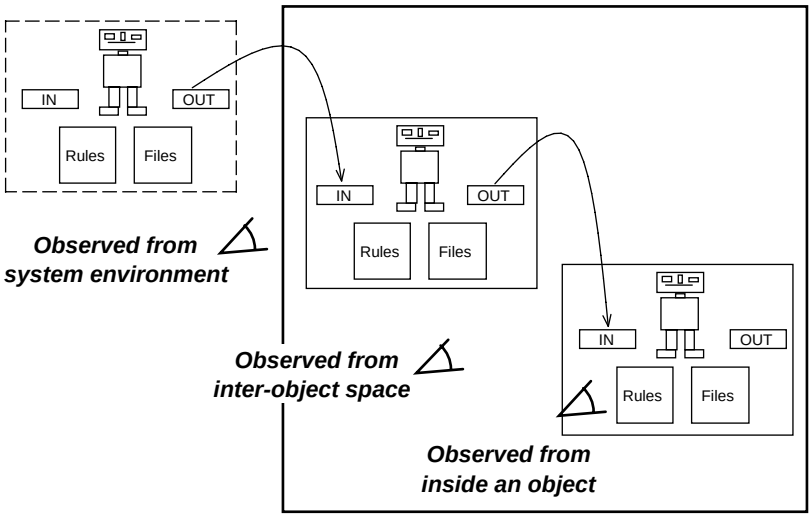

Figure 11: The importance of observation point

If we observe the system from inter-object space, we get views similar to figures 8 and 9. If we observe the system from its environment, we see the environment objects. The system itself appears as a single, *virtual role* as shown in the collaboration view of figure 12. We readily see that we could also make scenario views, they will look like figure 9 with all internal details suppressed.

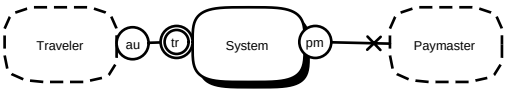

Figure 12: System User Model collaboration view as seen from the environment

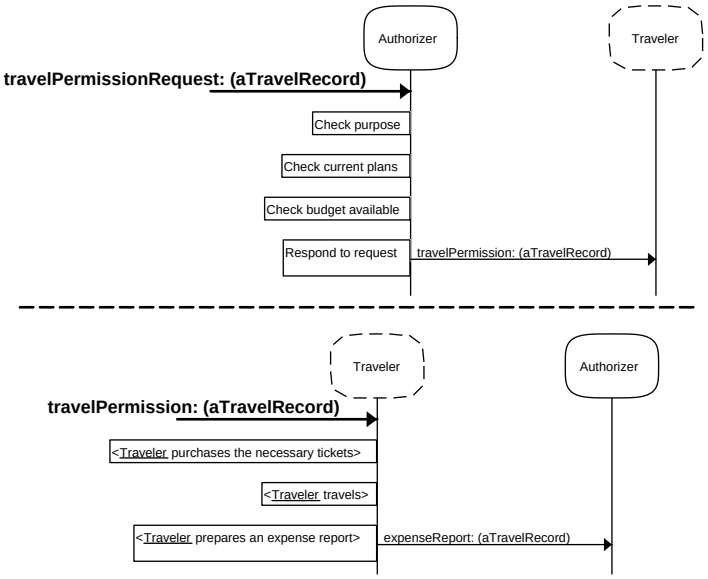

Figure 13: Two methods

If we move our observation point inside the object, we see its internal details. It is the observation point that yields the code; the class, the variables, and the methods. It is the observation point most commonly

adopted by popular object oriented methodologies. It is close to the implementation details, but far from the system as a whole. Method views and and possibly Finite State Diagrams are useful from this observation point. Figure 13 shows two method views, one for an Authorizer method and one for a Traveler method. These views are abstractions on the corresponding method codes expressed in some programming language.

4 *AirlineBooking:* Another System User Model

Our idea of *separation of concern* implies that we can create different role models on one and the same object structure. We illustrate this by indicating a model covering the purchasing of airline tickets:

> *Area of concern:* Airline tickets are ordered by a booking clerk and paid directly to the travel agent. The traveler is to show the cost on the expense report as an expense, and as an advance since the tickets are not paid by the traveler.

We show the collaboration view and a typical scenario in figure 14 without further comment.

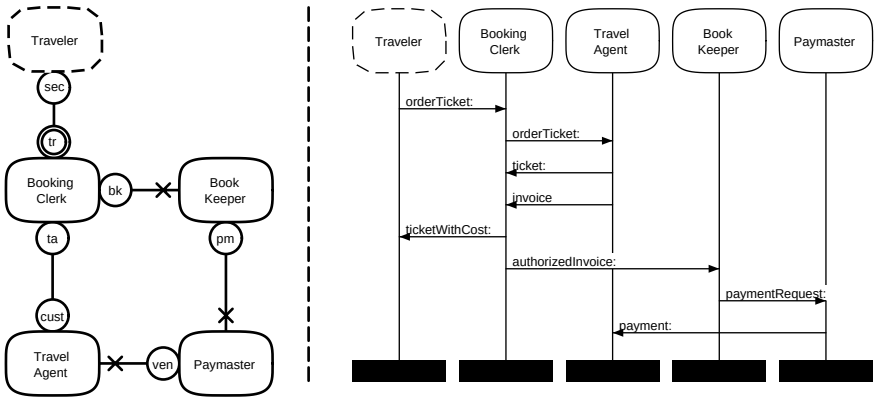

Figure 14: AirlineBooking collaboration and scenario views

5 *DerivedTravelExpense:* Model inheritance by synthesis

We have seen how the role model abstraction can be used for separation of concern; we can crate any number of models that describe different areas of concern from a potentially large and complex object structure.

We shall now study the opposite operation, called *synthesis.* We mentioned the key to this operation in section 3.1; an object can play several roles, and a role can be played by several objects. As an example, we will create a composite model that handles both TravelExpenses and AirlineBooking:

> The area of concern is the procedure for travel management including the purchase of tickets.

The new model is called the *derived model.* It is derived from two *base models:* The TravelExpense model and the AirlineBooking model. The synthesis operation is done by requiring the roles of the derived model to play the roles of the base models as illustrated in figure 15.

The two base models are shown shaded in figure 15. There are two synthesis operations, one for each base model. The derived model in the middle inherits all static and dynamic properties from these base models:

Static properties:

1. The semantics of the derived roles is consistent with the semantics of the corresponding base roles.
2. The responsibility of the derived roles includes the responsibility of the corresponding base roles.
3. The attributes of the derived roles include the attributes of the corresponding base roles.

4. All communication paths of the base role models must be found as corresponding communication paths in the derived model.
5. The message interfaces of the base ports must be included in the interfaces of the corresponding derived ports.
6. If a base role has been implemented as a class, a class implementing the derived role can be its subclass.

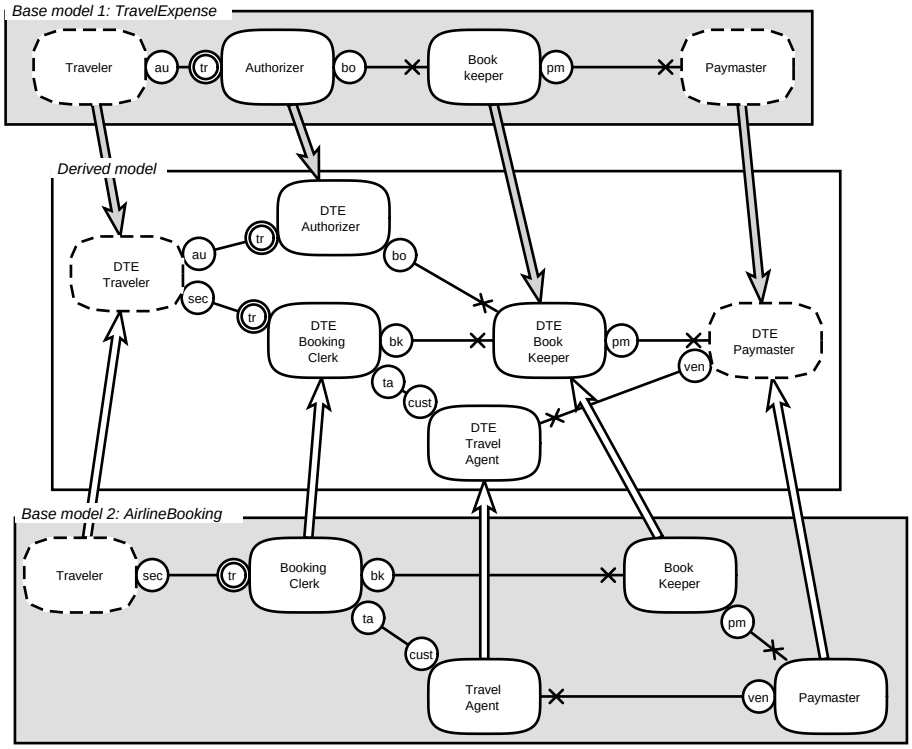

Figure 15: Deriving the composite model

Dynamic properties

1. The activities (use cases) of the base models are found in the derived model.
2. The message flows of the base models are found as corresponding message flows in the derived model.
3. The methods and possible finite state diagrams describing role behavior in the base models are embedded in the corresponding methods and finite state diagrams of the corresponding derived roles.

This idea of *model inheritance* is more powerful than the more common idea of *class inheritance.* The static inheritance properties are similar to those found in class inheritance, but the description of the dynamic properties of the base model object pattern can only be inherited when we consider the model as an entity.

The activities of the base models are repeated in the derived models. The activities can either be found as independent activities, the base model stimulus message then becomes a stimulus in the derived model. Alternatively, a base model activity can become part of another activity in the derived model. The base model environment role then becomes a regular role in the derived model; and the base model stimulus message is called from within a method in the derived model.

In our example, the ticketing activity is inserted into the travel expense activity. This is illustrated in the composite scenario of figure 16.

The actual coupling between the two activities takes place in the derived method for *travelPermission* in the derived Traveler role. The corresponding base model was shown at the bottom of figure 13, it included a comment <Traveler purchase necessary tickets>. The AirlineBooking model gives the details of this operation, and the corresponding activity is triggered as a subactivity in the derived method as

shown in figure 17.

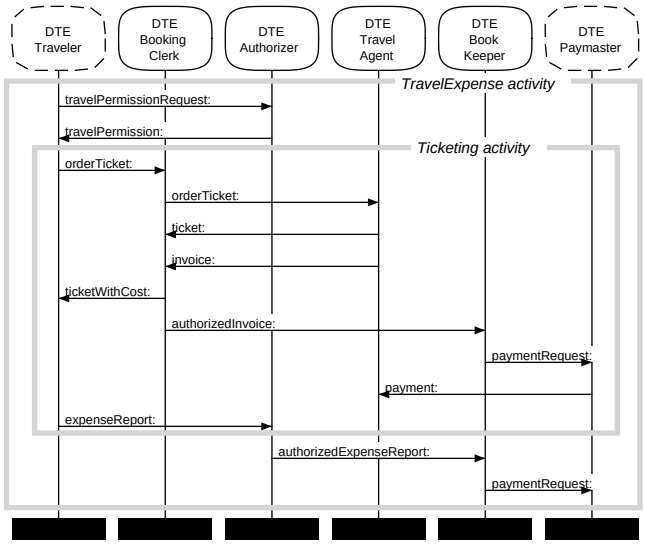

Figure 16: Composite interaction scenario

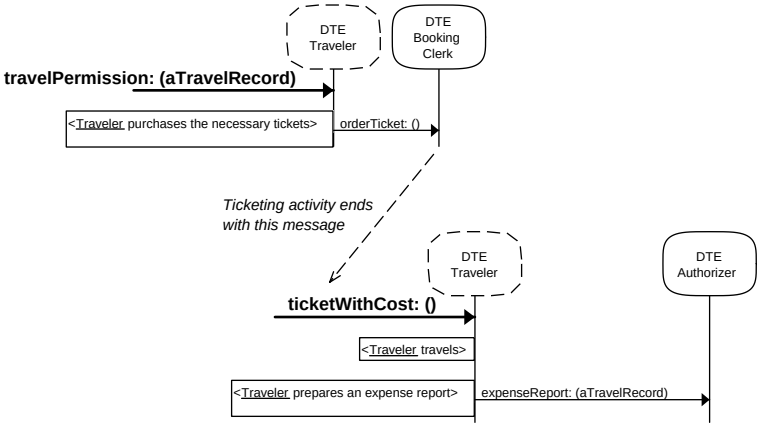

Figure 17: Derived model method sends base model stimulus message

Notice the fundamental difference between the creation of an external database schema and the synthesis of a composite model. The external database schema is created by filtering from a single, global schema. The creation of a derived model is the very opposite: the derived model is composed from a number of initially independent base models:

1. The specifications from the base models are initially orthogonal.
2. Attributes and Methods (actions) integrate behavior:
   - Traveler sends *orderTicket* as part of *travelPermission* method
   - Traveler learns airfare from AirlineBooking model
   - Traveler uses airfare when preparing expense report

The synthesis operation is used to support three popular abstraction mechanisms as illustrated in figure 18:

1. *Peer model composition.* An example could be that we have a base model for our budgeting operation, and another for expense control. A cost management model could be derived from these two models.
2. *Hierarchical decomposition and aggregation.* An example could be a materials management model that has a *PurchaseOrder*-role. Opening this role, we could see that it enclosed a role model with roles such as *PurchaseOrderItem*.

3. *Specialization/generalization.* We could, for example, have a base model describing general project management. We could then derive a model for large projects and another model for small projects.

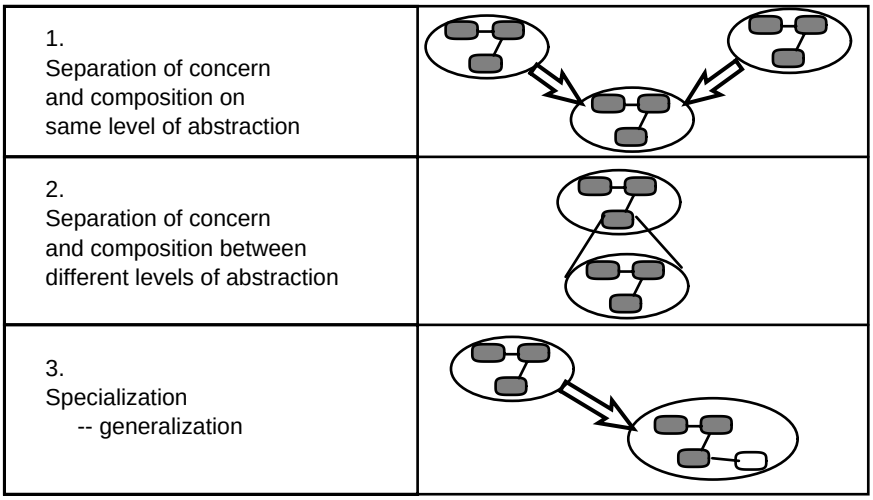

Figure 18: Three common applications of synthesis

6 *Travel Authorization* Tool Model

The methods of the System User Model describe tasks to be performed by the users. The methods can, therefore, be used as specifications for the computer-based tools to be provided for the users. As an example, figure 19 shows a possible tool for the Authorizer when determining to authorize a travel request. The method is outlined in the upper half of figure 13; it states that the authorizer shall check the application against current plans and budgets. The tools is designed for a user who does this kind of task only rarely, and who wants to make a decision within a minute. The tool collects all relevant information, and provides push-buttons for authorizing or rejecting the request. All communication and recording of the decision shall be made automatically.

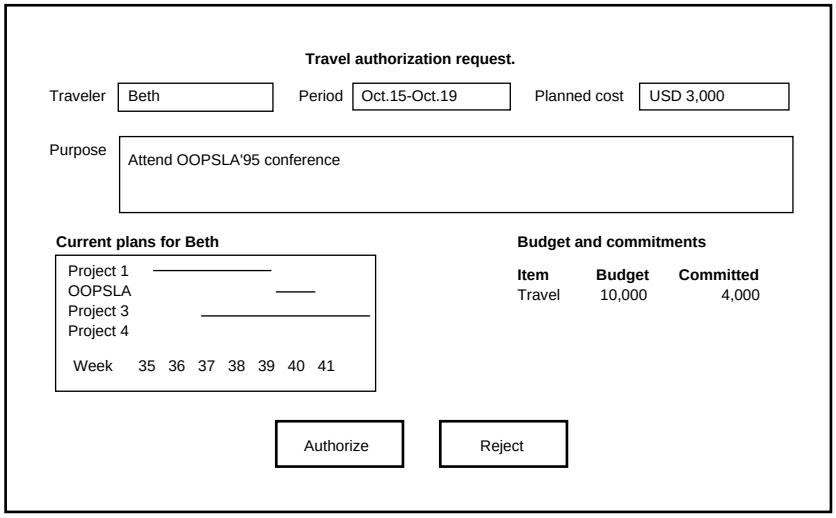

Figure 19: Travel Authorizer user interface tool

An interesting aspect of this interface is that it accesses three distinct databases as indicated in figure 20. The databases are the travel account service, the budget service, and the planning service. We believe this to be fairly typical for decision-making tasks. One of the great advantages of our three model architecture and client-server distributed systems is that they permit the creation of task-oriented tools independently of the domains of the required information.

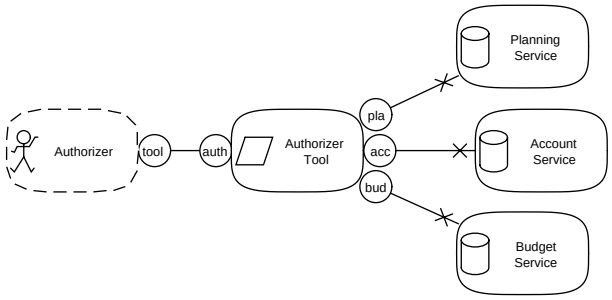

Figure 20: Travel Authorization Tool Model collaboration view

7 *TravelExpense:* An information model

The Information Model describes the universe of discourse of the User System Model; i.e., its message parameters and role attributes.

> The area of concern is modeling the information contained in travel expense accounts. We focus on the expense account itself and do not model details about the user interfaces.

The information model can frequently be implemented as a relational database; and the database schema can be defined with an appropriate framework. An object oriented database could be required if the Tool Models require a rich functionality. If all tools are as functionally simple as the tool of the previous section, a relational database will be very satisfactory for our TravelExpense example.

We will here only give a first sketch of a possible model. Figure 21 shows a semantic view of the most important parts, more work will be needed to specify the complete database.

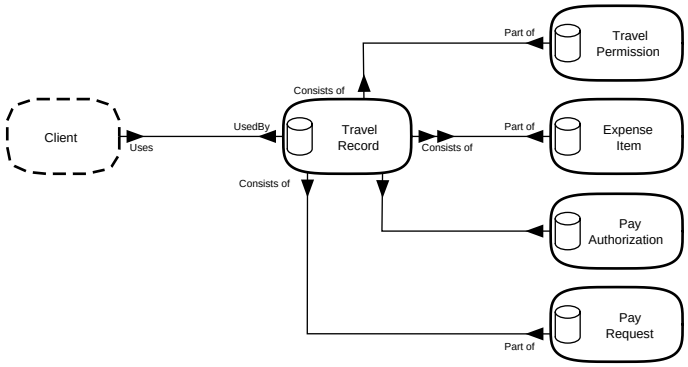

Figure 21: Semantic view

8 Conclusion

In this paper, we have shown how the proper application of object-oriented concepts and the OOram technology can help us open the boundaries of databases and application programs. By moving from closed to open systems, we can create multiple, interdependent models focusing on different aspects of the enterprise. We can model the user organization and its procedures separate from the information model, and the users' information tools independently of their information needs. Finally, we can study the interdependencies between these models, creating an overall understanding of the enterprise information handling needs and capabilities.

The OOram technology offers many advantages:

1. A role model can have several views highlighting:
   - static, data-oriented aspects
   - dynamic, function-oriented aspects
2. A role model can be seen from different observation points:
   - from the system environment observing its external properties
   - from the inter-object space observing the objects and the message flow
   - from inside an object, observing its construction
3. Most methodologies describe message interfaces on the level of objects or classes. The OOram technology specifies message interfaces on the level of communication paths, supporting the specification of distributed systems.
4. The OOram technology supports Separation of Concern, so that different models can be created for different purposes. These models are not limited by a hierarchical straight-jacket; each model can depict any pattern of objects:
   - a complex system of objects can be broken down into simpler role models through decomposition
   - simple base models can be combined into more complex, derived models through synthesis
5. The synthesis operation provides model inheritance. This preserves the static and dynamic properties of complete patterns of objects, rather than the single-object considerations supported by class inheritance.
6. The OOram technology offers a seamless bridge from analysis through implementation.

9 References

1. [E.And] Egil P. Andersen, Trygve Reenskaug: *System Design by Composing Structures of Interacting Objects.* ECOOP '92, Utrecht, Holland.
2. [Elmasri] Ramez Elmasri, Shamkant B. Navathe: *Fundamentals of Database Systems.* Benjamin/Cummings 1994. ISBN 0-8053-1748-1
3. [Etzioni] Amitai Etzioni: *Modern Organizations.* Prentice-Hall 1964, pp. 53-54
4. [HallFagan] Hall, Fagan: *General Systems, Yearbook of the Society for general Systems Research.* Ann Arbor, Michigan, Vol. I-X, 1956-65.
5. [Holbæk-Hanssen] Erik Holbæk-Hanssen, Petter Håndlykken, Kristen Nygaard:*System Description and the Delta Language*. Norwegian Computing Center, Oslo 1977.
6. [Jacobson] Ivar Jacobson et.al.: *Object-Oriented Software Engineering. A Use Case Driven Approach.* Addison-Wesley 1992. ISBN 0-201-54435-09
7. [J.And 91] Jørn Andersen, Trygve Reenskaug: *Operations on sets in an OODB.* OOPS Messenger, **2**, 4 (October 1991) pp. 26-39.
8. [J.And 95] Jørn Andersen: *The Dynamic Model.* Object Magazine **5**, 4 (July-August 1995) pp44-53
9. [Ree 92] Trygve Reenskaug, Egil P. Andersen, Arne Jørgen Berre, Anne Hurlen, Anton Landmark, Odd Arild Lehne, Else Nordhagen, Eirik Næss-Ulseth, Gro Oftedal, Anne Lise Skaar, Pål Stenslet: *OORASS: Seamless Support for the Creation and Maintenance of Object-Oriented Systems.* Journal of Object-Oriented Programming, October 1992, pp 27-41.
10. [Ree 95] T. Reenskaug, P. Wold, O.A. Lehne: *Working With Objects*. Manning/Prentice Hall, 1995. ISBN 1-884777-10-4