

# Enterprise Modeling using class and instance models

**Rakesh Agarwal**  
Infosys Technologies Ltd.,  
Near Planetarium, N.H.5,  
Bhubaneswar - 751013,  
India  
rakesh\_a@inf.com

**Giorgio Bruno**  
Dip.Automatica e Informatica  
Politecnico di Torino  
C.so Duca degli Abruzzi, 24  
10129 Torino, Italy  
+39 011 564 7003  
bruno@polito.it

**Marco Torchiano**  
Dip.Automatica e Informatica  
Politecnico di Torino  
C.so Duca degli Abruzzi, 24  
10129 Torino, Italy  
+39 011 564 7003  
torchiano@polito.it

## ABSTRACT

Current object-oriented formalisms, such as UML, focus on describing class models and use instance models only for depicting scenarios. Little attention is being devoted to defining how complex systems can be structured in order to conform with those class models and, further, which constraints class models must adhere to so that such conformity is affordable.

This paper focuses on instance models, which are models of actual systems and as such they are made up of instances, and discusses the relationships between instance models and class models. A large number of applications, ranging from generic data modeling to enterprise modeling, require instance models. The importance of hierarchical composition is emphasized: its meaning and implications are discussed in both contexts.

A novel approach for building templates out of aggregates of instances is also presented. A case study concerning enterprise modeling shows the application of the approach.

## Keywords

Object-oriented modeling, enterprise, instance models

## 1 INTRODUCTION

The object-oriented paradigm has gained a wide range of applications both in software engineering and in related fields. In particular, UML[1] provides a common notation and semantics for object-oriented models and poses itself as a new form of communication.

Most object-oriented methodologies are class-centered: in fact they are structured around class-relationship diagrams, called class models[1] or object models[2]. With a little simplification we can say that the key concept of most object-oriented methodologies is the class.

*Published in Proceedings of IEEE 7<sup>th</sup> Asia-Pacific Software Engineering Conference (APSEC 2000). Singapore, December 3-5, 2000.*

The class-centered approach gives good results when applied strictly to software development, but there are many applications, ranging from data modeling, to process modeling to enterprise modeling, which demand for a different approach.

Those applications produce models that consist of a large number of objects: instance models are then required.

Models formed by objects inherently are made up of a larger number of entities than class-centered models; thus a strong organizing principle is needed. Hierarchy is a natural and widely adopted organization policy.

This paper presents the basic ideas for extending current object-oriented methodologies in order to support the construction of hierarchical instance models.

The construction of a hierarchical instance model can be split into two phases: the modeling of the problem domain and the modeling of the specific system. The former leverage current state of the art object-oriented methodologies and tools to produce a suitable class model; the latter requires both new techniques and new tools.

In our view the construction of an instance model must be a matter of applying the rules defined in the class model. But this is possible only if the class model is built adopting a set of guidelines, which guarantee the class model is suitable for the construction of a hierarchical instance model.

The construction of an instance model poses some instances related to presentation and operational features of an instance oriented modeling tool. At first glance such issues are strictly pragmatic but they hold deep conceptual roots.

The remainder of this paper is organized as follows. At first the basic ideas underlying the instance models are presented, and guidelines are given to produce a suitable class model. Then an overview of enterprise models is given together with a description of the business process case study. The class model of the case study is described, which conforms to the guidelines defined above. Based on the class model an instance model is then developed,

highlighting the main representation and operational issues. At last a comparison with related work is presented together with a summary of the main contributions of this paper.

**2 FOUNDATIONS OF INSTANCE MODELS**

Even if it is possible to adopt a formal approach for an object-oriented methodology[13]; the approach adopted in this section to describe hierarchical instance models is partially formal for reason of space and simplicity.

The key step of the object-oriented paradigm is to identify the classes that compose a system. A class describes the common characteristics of a set of similar objects; it identifies a well-defined category of basic blocks that make up the system. An object that conforms with a class is said to be an instance of that class or, as an alternative, the object is said to belong to that class.

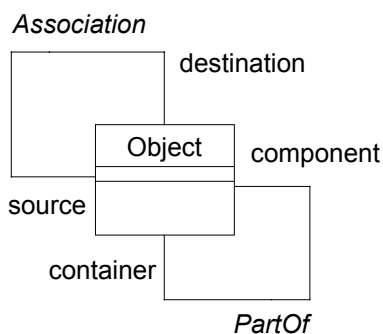
No object is an island: they can be related to each other in many complex ways. It is important to capture such links between objects and describe them as relationships between the corresponding classes.

Common object-oriented methodologies [1] [2] deal mostly with class models, which describe a system from an abstract point of view. Typically the model of a system consists of a class diagram[1], which describes the classes and their relationships. A concrete perspective of the system is provided by instance models. An instance model depicts a system as a collection of objects together with their associations.

An instance model is made up of objects and associations and is based on a class model. Each object is an instance of a class defined in the class model and each association between objects is an instance of a relationship defined in the class model between the corresponding classes.

An instance model is valid for a given class diagram if contains only instances of classes and relationships defined in it [1].

In the following we will define the basic concepts that can be used to build instance models for fairly complex systems.



**Figure 1: Meta-model of hierarchical instance models.**

**Hierarchical models**

We are interested in a particular kind of instance models: hierarchical instance models. The structure of a hierarchical instance model is shown in Figure 1. The PartOf relationship defines parent-child (component and container) roles: the source class plays the child (component) role and the destination class plays the parent (container) role.

In common methodologies, a complex model is described by means of a set of class diagrams. For clarity purposes we define an additional concept: a schema is a conceptual entity corresponding to the union of all the class diagrams. The name comes from an analogy with database schemas, which define the structure of a database and can be split into several ER diagrams for understanding purposes.

The relationships that appear in a schema can be partitioned into two categories:

- hierarchical relationships: they link parent (container) and children (components) in the hierarchy;
- associative relationships: they express logical associations between objects.

We will use the term “PartOf relationship” to denote any hierarchical relationship. In the following models we will use the aggregation notation to represent such kind of relationship.

In fact object-oriented methodologies use another important relationship: inheritance. The inheritance relationship is not considered in this paper both for simplicity and because instance models do not contain any element that is direct instance of it. However with a little effort, all reasoning presented in this paper can be proved to be compatible with inheritance.

The notion of PartOf relationship alone is not sufficient to build a hierarchical model: a set of constraints must be satisfied. In summary the set of PartOf relationships in a schema must define a connected DAG on the classes.

Some restrictions or rules must be imposed on the class diagrams so as to make the schema suitable for a hierarchical instance model.

A set of PartOf relationships is suitable for building hierarchical models if and only if it define a tree:

- the directed graph made up of classes and PartOf relationships does not contain cycles, i.e. it’s a DAG;
- the graph is connected;
- there is only one class that plays no component role in any PartOf relationship: it is called the root class.

In a schema resulting from a model built adopting a generic OOA technique, possibly we cannot find the above described PartOf relationship. Although it could be a good

modeling practice to start modeling with the idea of building a PartOf relationship inside the model, such approach is not mandatory. In fact it is possible to modify a schema that has no PartOf relationship so that it has one, without altering concepts encoded in the original schema. The steps to perform such transformation are as follows:

1. add a new root class;
2. build the complete set of PartOf relationships:
  - for each class that takes no component role in any PartOf relationship:
    - ◆ add a new PartOf relationship between that class and the root class

The classes Organization, BusinessProcess and Department, together with their relationships, which are depicted without shadow in Figure 2, form a model that does not comply with the above requirements.

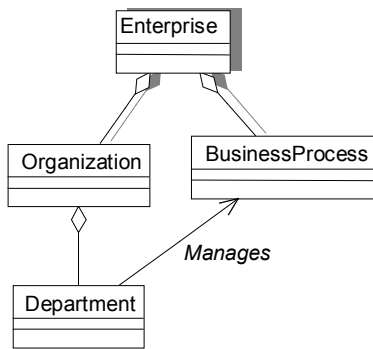


Figure 2: Modification of a schema.

A new root class named Enterprise has been added; the two PartOf relationships has been added: one from class Organization to the root class and the other from class BusinessProcess to the root class.

Once a suitable schema is available, a hierarchical instance model can be built by means of the following two operations: Add and Link.

The two operations differ in the kind of relationship they deal with: the former sets up the PartOf relationship, the latter connects objects by means of an associative relationship. Both operations preserve the compliance with the schema the original instance model is based on.

The Add operation creates a new object and puts it into the model setting up a PartOf association to a pre-existing container object. The Link operation sets an associative relationship between objects already present in the model. We can say that the Add operation creates the hierarchy of the model while the Link operation enriches the model with logical associations between objects.

For completeness a third operation should be mentioned: the start operation. Such an operation creates a new root object that will contain the objects created by the add

operation. The purpose of the start operation is to initialize the instance model. We will not further refer to that operation, assuming an implicit initialization of the instance model.

The Add operation can be applied to an instance model. It takes as arguments:

- an object  $o_1$  belonging to the instance model,
- an object  $o_2$  that does not belong to the instance model and is an instance of a class defined in the schema of the instance model.

It modifies the instance model to which it is applied to as follows:

- it adds object  $o_2$  to the model;
- it adds a PartOf relationship from  $o_2$  to  $o_1$ .

In the resulting model, the added object  $o_2$  will play the component role, while  $o_1$  will play the container role in the same new PartOf association.

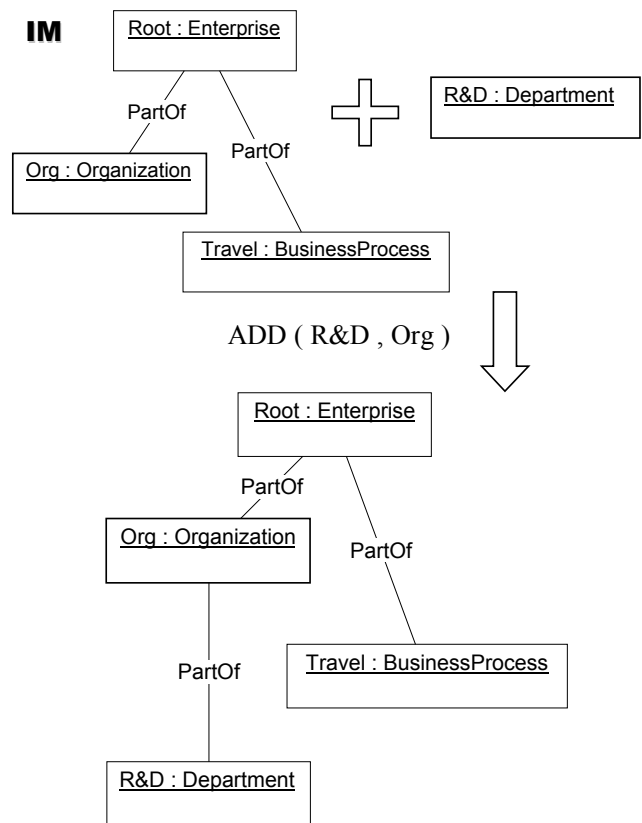


Figure 3: Example of Add operation.

An example of the Add operation is shown in Figure 3. Both the original and result instance models are based upon the schema shown in Figure 2. The add operation is applied to the model shown in the upper left side of the figure, the component object is the *Org* object of class *Organization*; the other object argument of the operation is a new object,

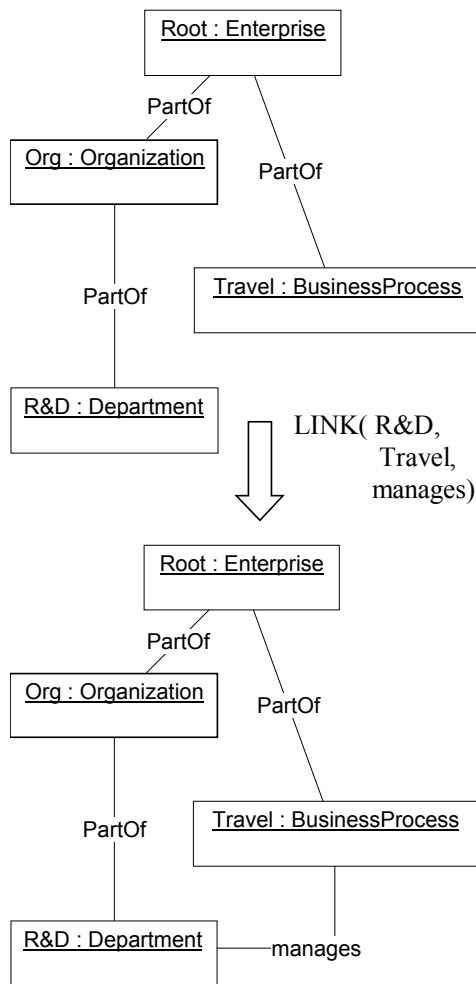
R&D of class *Department*. The resulting model, shown in the right side of the figure, is made up of the same objects and associations as the original plus object *R&D* and a *PartOf* relationship from *R&D* to *Org*.

The Link operation can be applied to an instance model. It takes as arguments:

- two objects  $o_1$  and  $o_2$ , both belonging to the instance model,
- an associative relationship  $r$ , whose source class is the class  $o_1$  belongs to and whose destination class is the class  $o_2$  belongs to.

It modifies the instance model it is applied to as follows:

- it adds an association, instance of the relationship  $r$ , from  $o_1$  to  $o_2$ .



**Figure 4: Example of Link operation.**

An example of the Link operation is shown in Figure 4. Both the original and the resulting instance models are both based upon the schema shown in Figure 2. The Link operation is applied to the model shown in the left side of the figure; an instance of relationship *manages* is set from

object *R&D* of class *Department* to object *Travel* of class *BusinessProcess*. Obviously relationship *manages* has been defined in the schema from class *Department* to class *manages*.

### 3 ENTERPRISE MODELS

While there is an increasing number of enterprises that are changing their business operations from a functional approach to a process-oriented approach, enterprise modeling is perceived as a necessary step to perform this change.

Current modeling technology allows the use of enterprise models to go far beyond descriptive purposes[3]; in fact, such models can play a major role in defining and managing business processes and in mapping business processes to support systems such as workflow systems[4][6][7].

Enterprise models can be used for different purposes, but it is important to observe that, at least, they act as a repository that semantically organizes the knowledge about the enterprise; this knowledge can be used to achieve specific goals by means of suitable tools. Models describe processes, their interactions and their relationships with the organization and the information system and provide qualitative and quantitative results during the whole lifecycle of the process.

However, building effective enterprise models is difficult as it requires handling a large number of concepts which are needed to express the functional, information, organization and resource views and managing a complex hierarchical architecture where many interactions take place between subsystems.

A general reengineering effort affects the whole enterprise: from design to manufacturing, from customer management to supplier management. This complex and wide reengineering activity, because of the great number of factors influencing enterprise processes, cannot be achieved without a sound methodological approach.

Experience[4] [7] has shown the benefits of suitable models as tools for process analysis, optimization, test and management. But all that is not enough to ensure consistency of the efforts devoted to optimizing specific aspects and/or processes with the global goals of the enterprise itself.

The solution to the above-mentioned problem requires the integration of all enterprise processes (in terms of activities, information, organization and resources) in a global model to better evaluate the global achievement.

#### The case study

Corporate policy states that any employee who needs to buy something, or to use company funds, should fill in a standard *expense proposal* form, which has to be signed and approved by the manager of the department the

employee belongs to. The form contains information about the expense such as purpose, description, amount, and provider.

After the form has been signed, it is sent back to the request originator to notify him/her he/she is allowed to perform the expense. After the originator has performed the expense, he/she can possibly provide the details for the payment and submit the form to an employee belonging to the payment office to make the effective payment.

As an example, let's consider the case in which an employee of the R&D department wants to buy a new workstation; he/she starts an expense approval business process and fills in the request form. If the employee happens to be the manager of the R&D department then there is an implicit approval, otherwise the manager's approval is required. If the approval has been obtained the employee can go and order the workstation; when it has been delivered the expense is completed and the requiring employee can fill the details of the bank transfer payment. Finally the request form can be given to a payment employee, possibly working for the finance department, who will request the bank transfer.

**4 SCHEMA**

The case study described in the previous section requires a hierarchical instance model. According to the guidelines presented above a suitable schema must be provided in order to build the instance model.

In this section we will describe a schema that forms the domain model of the case study. It is important to understand that the purpose of this schema is to enable a domain expert to describe a system with as little effort as possible. In fact we can identify two phases in the construction of instance models:

- domain modeling: the construction of a schema,
- system modeling: the construction of an instance model conforming to the schema.

The more a schema is well thought the more it can be reused to build models of systems in the same domain.

A modeling environment supporting such an approach should be made up of two different tools: a class level tool, whose functionality is similar to current UML CASE tools, and an instance level tool. The former should be used to define a schema; the latter can be used to build instance models.

The schema for the case study is divided into three class diagrams; the UML[1] graphical notation has been adopted. The first class diagram contains the classes describing the flow of activities; the second contains the classes referring to a simple organization; finally the third diagram is used to integrate the first two.

It is important, while defining the schema, to keep in mind

that enterprise models can have different purposes. As an example, a business process model can be used:

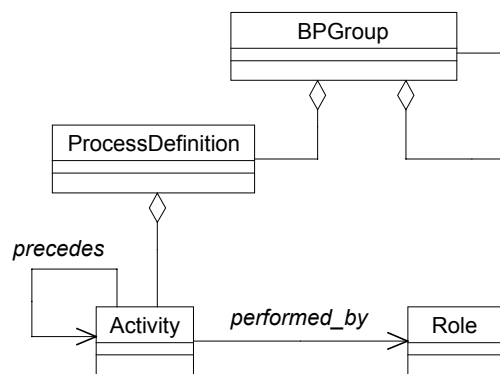
- to document the practice in the enterprise,
- for simulation, for resource usage optimization,
- as a basis for a workflow system,
- as a powerful tool for a BPR initiative.

**Business Process**

The class diagram shown in Figure 5 contains the classes dealing with the description of a business process as a flow of activities.

A large enterprise might want to divide its processes into financial processes and production processes, then further divide financial processes in national and international processes, and group the production processes for each plant the enterprise owns. A hierarchical model is the most natural solution to such an issue: a group of objects representing process definitions can be gathered together as components of a container object: a group of business processes. Such process groups can be organized in a classifying hierarchy.

Classes *BPGroup* and *ProcessDefinition*, together with their relationships described in the class diagram shown in Figure 5, can be used to carry out the structure just depicted. *ProcessDefinition* objects are components of *BPGroup* objects, which are containers; in addition a *BPGroup* object can play the component role if it becomes part of another *BPGroup* object. Using the Add operation a tree of *BPGroup* objects can be built representing some form of classification for the business processes enacted in the enterprise. A *ProcessDefinition* object is made up of a set of *Activity* objects linked by a precedence relationship.



**Figure 5: Process definition class diagram.**

**Organization**

The organization is made up of the people working for the enterprise. The functional description is very common: the organization is divided into organization units, each of which plays a different function and can be divided into other organization units. An organization unit can contain a number of organization elements.

The elements contained in a unit usually play different roles, e.g. a manager, a secretary, an engineer.

The class diagram shown in Figure 6 supports the definition of a model conforming with the approach described above. The *Organization* is composed by *OrgUnit* objects, which can be decomposed into other *OrgUnit* objects and can contain *OrgElement* objects. In addition the *Organization* can contain various *Role* objects.

Such a structure is somewhat simplified, in particular for what concerns roles, but it is very easy to understand.

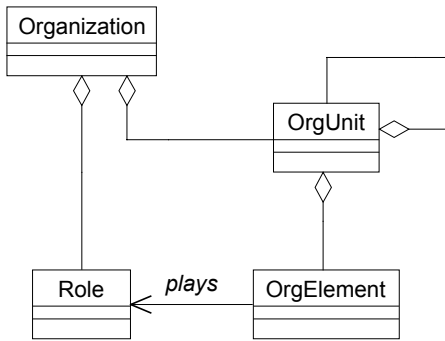


Figure 6: Organization model class diagram.

**Integration**

The two class diagrams are almost disjoint, except for the *Role* class, but in fact they describe two complementary features of the same system. An integration class diagram is needed to obtain a schema suitable for a hierarchical instance model.

Figure 7 shows the integration class diagram: the *Enterprise* class represents the whole enterprise, it is the root class and contains the *BPGroup* class and the *Organization* class.

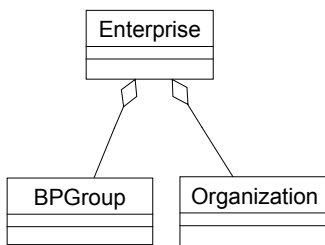


Figure 7: Integration class diagram.

**5 INSTANCE MODEL**

In this section the hierarchical instance model of the case study is presented. The model is based upon the schema presented in the previous section.

At first a purely hierarchical model, with no logical association, is described; this model can be obtained by means of the add operation.

**The hierarchy**

According to the schema presented in the previous section an *Enterprise* object is at the root of the hierarchy; it

contains *Processes* and *Org*, instances of the *BusinessProcesses* class and the *Organization* class respectively. They are the containers of the two main branches the enterprise is divided into: the processes and the employees performing them.

Figure 8 shows the hierarchy of the model of the case study. It is made up of a single business process and a simple organization containing three roles and two departments.

The *ExpensesApproval* business process contains five *Activity* objects corresponding to the steps to be undertaken to complete the process.

The organization (*Org*) contains two organization units, representing the *R&D* department and the finance department. In addition it contains three roles: the ones required by the process activities. The organization units contain three employees, represented by *OrgElement* objects, one for each role.

Building a model by adding nodes and leaves through a tree representation, like the one shown in Figure 8, is not affordable for many reasons:

- as a model grows, its graphical representation becomes less understandable;
- even if it is divided into sub-trees, the components are arranged in a list-like order which can be misleading;
- the logical associations between objects are not visible.

Thus the tree representation is not always the best one.

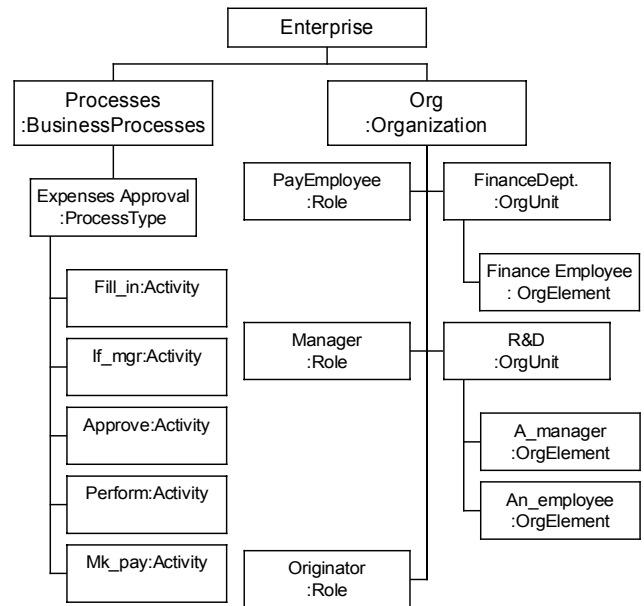


Figure 8: Hierarchy of the case study model.

### The business process

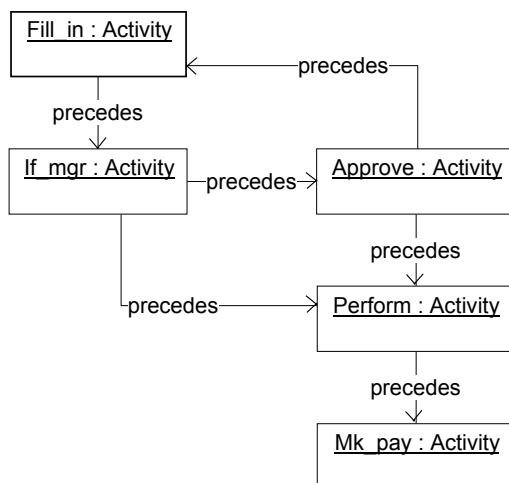
The business process of our case study shows very well the need for an alternative graphical representation. The ability to see all the components of a single container, possibly as a list, is a first approximation of a satisfactory solution.

Such a solution is not enough for a whole category of cases, like business processes, where each component of a container participates in at least one logical association with another component of the same container. Such components can be referred to as *clustered components*. In the case of business processes, at least one precedence relationships holds for each activity.

Each container should be associated with a graphical representation of its component objects. Such a representation allows the logical links between the objects of the same container to be easily defined and understood. The rationale for such a proposal is that clustered components are closely related to each other, so their relative roles can be better explained using a graphical notation.

A comprehensive representation of such a description, applied to the *expense approval* business process is shown in Figure 9. There are five activities:

1. Fill\_in: it is the first activity, it has to be performed by the request originator;
2. If\_mgr: if the originator of the mission is the manager of its department, the Approve step will be skipped;
3. Approve: if the previous test fails then the manager must approve the request, if it is rejected then the request has to be submitted again by its originator;
4. Perform: once the expense has been carried one the originator can fill in the details of the payment;
5. Mk\_pay: at last payment can be made by the appropriate office.



**Figure 9: The expense approval business process.**

Performed\_by associative links do not appear in the above figure because, according to the schema described in the previous section, Role objects are hosted in a different container. Such associations are set directly in the instance model; in particular they are as follows:

Fill\_in → Originator ,                      Approve → Manager  
 Perform → Originator ,                      Mk\_pay → PayEmployee

### 6 RELATED WORKS

Harel and Gery present [9] an approach for building executable object-oriented models. They poorly address the instance model problem: the authors attempt to solve the problem forcing object instantiation to be constrained by a cardinality defined at a class level. Such an approach is not affordable for a complex model: if we think of a business process, we cannot say how many activities make up a generic one nor can we say which relationships exists between each other. Our approach is more focused on instance models and presents effective techniques for building large models.

Seiter, Palsberg and Lieberherr present [10] an approach which supports dynamic evolution of objects. They propose an extension to UML in order to provide a class-level support and an extension to Java, which provides an instance-level support. They identify problems rooted in the class-centered approach of current object-oriented methodologies. They find out that the basic problem with class based models is the lack of adequate constructs for modeling dynamic behavior. We address a different problem: the construction of large instance models. The two approaches are complementary in extending current object-oriented techniques.

The importance of hierarchy and aggregation in object oriented models is proved by a large number of works dealing with the semantics of composition, part-of and part-whole relationships. In [8] a seminal classification of composition relationships using three semantic concepts is presented. In [11] a survey of part-whole relations and their use in object-oriented methodologies is presented. Another recent work [12] focuses on the importance of aggregation in object-oriented methodologies and stresses the various different meanings that aggregation can have. In our work hierarchy plays an different role: it forms the backbone of any instance model, its main purpose it to make an instance model well structured; in addition in certain cases PartOf relationships can assume a more specific semantics.

In [14] a formal semantics of domain models is presented, which can be translated into executable code by suitable tools. The focus of such work is mainly on dynamic behavior of the system, our approach's focus is mainly on large static models. In addition we address the problem of the construction of models by providing an operational semantics to domain models.

## 7 CONCLUSIONS

The weakness of most object-oriented methodologies lies in their class-centered approach. But modeling large systems requires an instance-based approach. We presented a novel approach for building hierarchical instance models of large systems.

A set of requirements for class models has been defined; the compliance with such requirements enables the construction of a hierarchical instance model.

Given a suitable class model, as a result of a domain modeling phase, system modeling can be carried on by means of two simple operations, add and link, which permit an incremental approach to system model building.

A complex case study points the need for specific support tools. A new graphical representation, which address clustered components, should be devised.

The proposed ideas are the result of many years of research mainly in the area of enterprise modeling. They are currently being implemented and tested in a modeling environment. They proved to be very effective in handling complex models[7], in particular modeling a number of business processes with FIAT Auto[4].

Further study will be devoted to operational issues deriving from the real use of instance tools. Such issues will provide the basis for new extensions to mainstream object-oriented methodologies.

## REFERENCES

- [1] OMG, Unified "Modeling Language Specification, Version 1.3", June 1999.
- [2] J.Rumbaugh et al. "Object-Oriented Modeling and Design", Prentice Hall, 1991.
- [3] G.Bruno, R.Agarwal "Modeling the Enterprise Engineering Environment", IEEE Transactions on Engineering Management, (44)1, February 1997.
- [4] G.Bruno, C.Reyneri, M.Torchiano. "Enterprise Integration: operational models of business processes and workflow systems" in Enterprise Engineering and Integration: Building International Consensus: Proceedings of ICEIMT'97, Torino, Italy, October 28-30, 1997, K.Kosanke and J.G.Nell editors, pages 408-419, Springer-Verlag, 1997.
- [5] R. Agarwal,G. Bruno and M. Torchiano. "Object-Oriented architectural support for developing complex systems" in Proc. of IEEE 23rd Annual Int.Computer Software and Applications Conf. (COMPSAC'99), pages 259-264, Phoenix, AZ, USA, October 27-29, 1999.
- [6] G. Bruno, M. Torchiano, "Making CIMOSA Operational: the experience with the PrimeObjects Tool". Computers in Industry 40(2-3), pages 279-291, Elsevier Science, November 1999.
- [7] R. Agarwal, G.Bruno and M.Torchiano. "An Operational Approach to the Design of Workflow Systems", Information and Software Technology 42(8), pages 547-555, Elsevier Science, May 2000.
- [8] Winston, Morton, R.Chaffin, D.Herrmann, "A Taxonomy of Part-Whole Relations," Cognitive Science, 11, 1987, pp. 417-444.
- [9] D.Harel, E.Gery, "Executable Object Modeling with Statecharts", in IEEE Computer 30(7), July 1997.
- [10] L.M.Seiter, J.Palsberg, K.J.Lieberherr, "Evolution of Object Behavior Using Context Relations", IEEE Transactions on Software Engineering, 24(1), January 1998.
- [11] Artale A., Franconi E., Guarino N., Pazzi L., "Part-Whole Relations in Object-Centered Systems: An Overview" in Data and Knowledge Engineering 20(3), October 1996.
- [12] Motschnig-Pitrik R., Kaasboll J. "Part-Whole Relationship Categories and Their Application in Object-Oriented Analysis", IEEE Transactions on Knowledge and Data Engineering, 11(5), September/October 1999.
- [13] R.H.Bourdeau, B.H.C.Cheng. A formal Semantics for Object Model Diagrams, in IEEE Transactions on Software Engineering 21(10), October 1995.
- [14] S.A.DeLoach, T.C.Hartrum. "A Theory-Based Representation for Object-Oriented Domain Models", IEEE Transactions on Software Engineering, 26(6), June 2000.

