# Workflow modeling: concepts of activity, generalization and aggregation.

**Duncan Dubugras Alcoba Ruiz**

Pontifícia Universidade Católica do Rio Grande do Sul - PUC-RS
Instituto de Informática
Pós-graduação em Informática
Porto Alegre - RS - Brazil
E-mail: duncan@inf.pucrs.br

**José Palazzo Moreira de Oliveira**

Universidade Federal do Rio Grande do Sul - UFRGS
Instituto de Informática
Pós-graduação em Ciência da Computação
Porto Alegre - RS - Brazil
E-mail: palazzo@inf.ufrgs.br

## ABSTRACT

Workflow modeling aims the representation of dynamic computer-supported activities. The representation of the dynamic aspects of *workflows* is a central issue in software engineering but this issue is hardly supported by traditional models. In this paper it is described a *workflow model* employing two primitive concepts: *activities* and *office objects* and two abstraction mechanisms: *generalization* and *aggregation*. A specific workflow modeling language is developed to represent *activity diagrams*. These diagrams are employed to specify *workflow* activities, representing the objects' flow, the transformations performed by agents on those objects, and the *decision making* involved in these activities. The dynamics of an application is described following the organizational and functional solutions adopted by an enterprise, as well as the assignments and responsibilities of the agents working in the organization. The main contribution of this paper is the precise definition and formalism for the workflow representation.

## 1. INTRODUCTION

The *workflow* concept evolved from the notion of process in manufacturing and the office environment. A *workflow* may be characterized as a set of *activities* – performed either automatically or by humans – and by the associated enabling and integrity rules. This kind of system needs a comprehensive design methodology. However, the traditional methodologies used in software engineering are dedicated to the description and the development of the automated procedures, not for the description of a complete information system including the automated and the human parts of work. The use of modeling concepts as *generalization* and *aggregation* and a precise graphical and formal workflow description allows us to capture the particular workflow structure.

Activity is any task executed in an office that consumes time, that employs resources, and of which the execution responsibility is assigned to an agent or group of agents. Time is related to the dedication of an agent for the accomplishment of a task. A particular characteristic of production environments is that, although it is possible to clearly identify the agent responsible for each task, it is impossible to consider activities in isolation, since there exists a great interdependency among them. The set of activities in a workflow application represents the underlying dynamics of an enterprise, characterizing the existing parallelism among them, and emphasizing the synchronization of the executed work.

Activities performed are not always clearly defined. In many situations, the knowledge necessary to carry out a task, i.e. the routines used to properly manipulate objects, is private knowledge to the responsible for the task, and sometimes this person do not wish to make this knowledge public. In other situations, there may exist certain particularities on the activities that make it impossible to pre-determine the sequence of routines to be used. The same activity can be performed in alternative ways reaching an acceptable result. Several related works share a similar point of view: [WEI 89] and [ANG 92] state that

office activities are often ill-structured; [ELL 80] and [ELL 91] point out that the information required for the fulfillment of tasks is frequently incomplete in office environments, and that exceptions happen quite often. Indefiniteness are intrinsic to workflow applications, in the sense that one knows about the existence of an activity, the objects it manipulates and how the initial and final contents of these can be described – the business rules – and the agent or group of agents responsible for it execution, but sometimes one ignores the complete activities sequence to reach the final content produced from the initial. Therefore, representation models must adequately support these indefiniteness.

The objective of our research is to develop both a conceptual model and an environment for the analysis, design and implementation of systems including human tasks in offices as an integrated part of the computerized environment - workflow systems.

This paper is structured as follows: section 2 characterize the working environment of the research. Section 3 introduces a rigorous graphical model for workflow representation followed by the formal model specification in section 4. Section **Error! Reference source not found.** describes an environment for the description, administration and use of modeled office applications and some remarks about activities' reuse are outlined. Conclusions are presented in Section 5.

## 2. THE ACTIVITY MODEL

The purpose of our research work is to develop both a conceptual model and an environment for the analysis, design, implementation and component reuse in workflow systems. The model for representing activities in applications is targeted at the modeling of the dynamic aspects of such applications. The activity model concepts are: activities and office objects. The modeling of application activities, which corresponds to an activity diagram, represents the flow of objects, the transformations applied on them by agents, and the decision-making involved in the activities [RUI 89]. It describes the dynamics of the application, considering the organizational and functional structure of the enterprise, as well as the assignments and responsibilities of the agents working in this organization.

In the particular case of administrative applications, *office objects*, besides being complex, are largely standardized in their descriptions due to legal reasons (e.g. official letters and petitions, attorney letters), or due to commercial reasons (e.g. minutes, receipts, invoices). As a consequence, office objects constitute a domain of classes that are intensively reusable in the modeling of office applications. To achieve the reuse of office classes belonging to the problem domain, the office class library, independently of the particularities of each office application, must be present in an OODBMS, and in its associated development environment.

### 2.1 Activities

Activity is the central component of the model. The interaction of an agent with the application permit office objects to be created, queried, updated, destroyed and sent to other agents. The flow of office objects is represented in terms of the possible sequences of *places* that these objects may reach. The data structure and methods allowing office objects to manipulate *places* are included in these objects by multiple inheritance from a special class called *Office Object Place*. In order to be conveniently employed in the representation of the interaction between classes and activities, the methods belonging to the public interface of office objects must be classified by the interaction type (query, construction, update, destruction), as well as by the interaction context (class methods and instance methods). Besides the interaction among objects, it is possible to represent, in an activity, the conditions to be met for its execution, as well as the constraints in the production of results.

An *activity diagram* is an annotated graph composed of activities and office objects, where each activity is connected through branches exclusively to objects, and objects are connected through branches solely to activities. The branches indicate the different ways in which office objects are manipulated by activities. By examining the activity diagram, it is possible to identify the activities that can be performed independently of other ones, those where conflicts arise in the treatment of objects, and also those that

have a relative execution order defined for them. In this way, it is possible to represent the decentralization of activities, as well as the asynchronies among them. The model construction for a complex reality is supported by a diagram decomposition technique, allowing the designer to divide the problem into smaller parts.

### 2.2 Activity abstractions

If a model is expected to support and promote adequate reuse of its components it must make available to the designer mechanisms allowing the reuse of already existing modeling constructs, and this reuse must result in a gain of time and quality. Besides classification, the main concepts that enable these gains are *aggregation* and *generalization*. These concepts, for which the work of J.M. Smith & D.C.P. Smith [SMI 77] is the seminal reference, are already traditional in database conceptual modeling. The present model for representing office activities makes these two abstraction techniques available, providing the necessary support for employing *activities* as reusable components.

In object-based models, those concepts are employed also for modeling the behavioral aspects intrinsic to classes. However, object-based models do not commonly allow the use of these concepts for modeling the interaction between classes. Indeed, they assume that class interaction aspects can be represented basically by message exchanges between objects. Works on conceptual modeling, such as [PER 89], [EDE 94] follow this trend, even though they recognize that information systems modeling requires that the communication pattern among agents, as well as the role these play, must be represented using more suitable, specific mechanisms. A consequence of this situation is that, in these models, the successful conclusion of a *user transaction* depends on the adequate behavior of the participant objects. In these works, such a concept is called *activity*. Works as [LIU 92], [KUN 93] and [KAP 91] also propose the representation of transactions by a special concept. However, none of these references consider the activity concept, as a first-class object, orthogonal to other application objects. The model presented in this paper considers *activities* as first-class objects, orthogonal in respect to other application objects, being thus potentially reusable.

The only description of application components as classes is not a sufficient condition to make them automatically reusable [McG 92]. The experience has demonstrated that stable and reusable classes are not defined from the beginning. Instead, these are the result of a repetitive process that involves tests and improvements [JOH 88]. The main conclusion is that the development of a class library must be oriented towards reuse. Empirical studies showed that component reuse increases significantly the productivity of system development teams, and that with the use of OO environments, and therefore with the reuse of classes, the benefits can increase even more [LEW 91].

## 3. THE GRAPHICAL MODEL

The graphical representation given for components in the activity diagram has the following characteristics :

- For office classes, a symbol similar to the one representing the concept *class&object* of the Object Oriented Analysis of Coad & Yourdon [COA 91] is used. This symbol, Figure 1, was adopted since it distinguishes between class and instance.

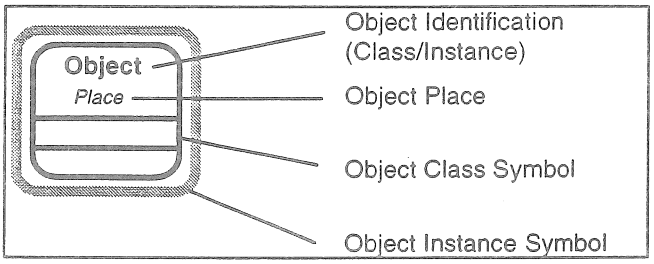


Figure 1 : Office Class Representation.

- Activities are graphically represented by a three-layer rectangle, Figure 2. In an activity, the expression corresponding to an enabling rule is a first-order formula (returning true or false), and the one

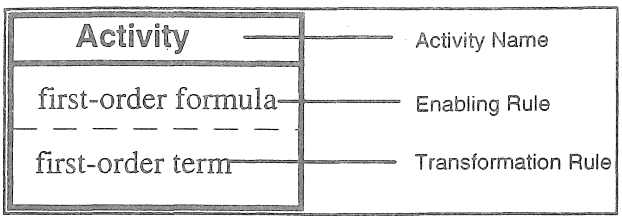


Figure 2 : Activity Representation.

describing a transformation rule is a first-order term (set of assignments).

- Connection branches are represented by lines, which are labeled and sometimes directed, as shown in Figure 3. In this picture, v:=m(v1,...,vn) stands for the generic call of a method. When the method labeling the branch corresponds to a class method, the method identifier is prefixed with the symbol $. The combination of the branch type and the method type labeling it must be consistent, i.e. a branch that connects a class to an activity may only have as label a class method, and a branch that connects an instance to an activity may be labeled solely by an instance method. Inclusion branches are the only exception, since a constructor method, although constructor of objects, is a class method.

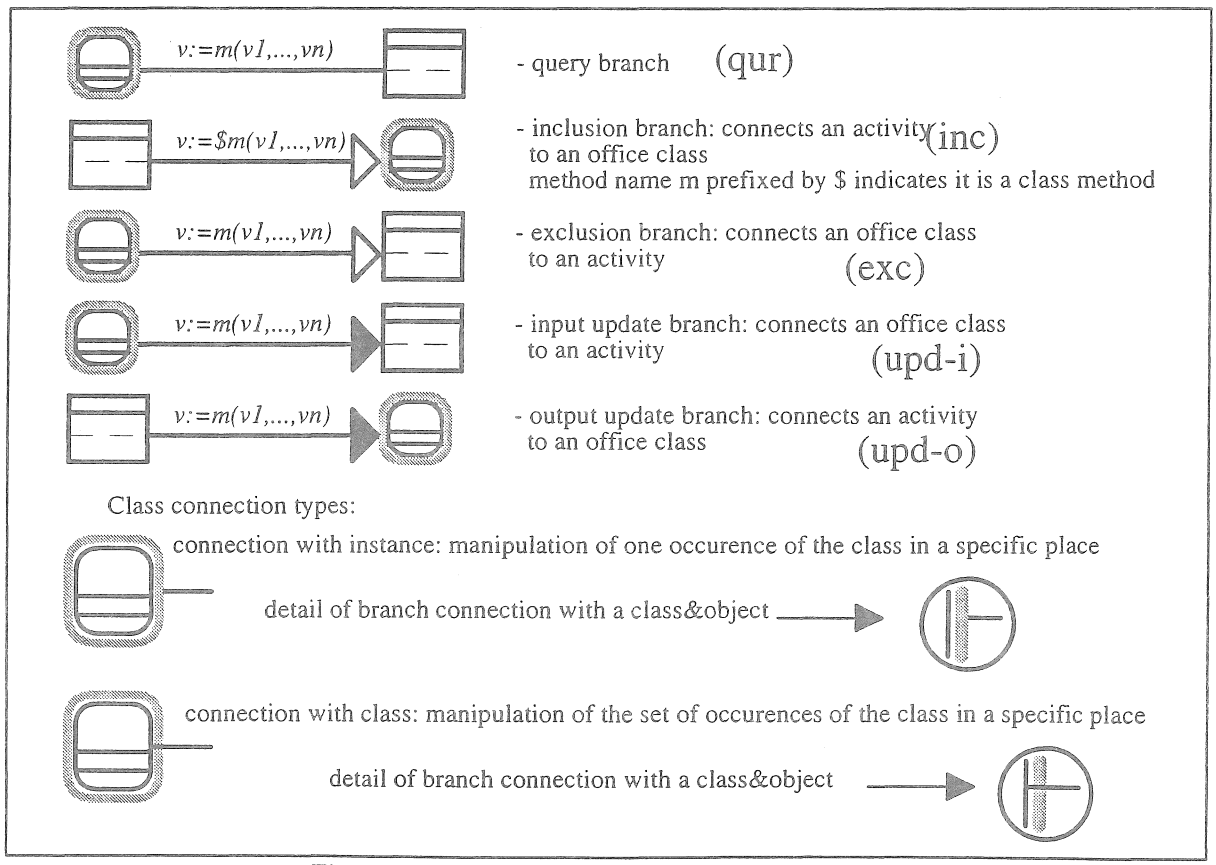


Figure 3 : Graphical Representation of Branches.

Update branches always occur in pairs, i.e. an input update branch coming into the activity, with the corresponding output branch leaving the activity. Since an office class is always a subtype of *office object place*, being therefore capable of handling its own location, its update can imply either :

a) the update of both the content and place of an object, depicted in Figure 3 as the *upd-i* and *upd-o* update branches;
b) the update of the object place only: this update has a simplified graphical representation, shown in Figure 4. The output branch of the pair representing the update is labeled only by variable v, which is the receiver of the result produced by the method m labeling the input branch, since place update is an implicit service performed by activities;
c) the update of the object content only: this update is also represented in a simplified diagrammatic form by a single line, as presented in Figure 5, where v:=m(v1,..., vn) corresponds to the input update branch label, and w:=n(w1,..., wn) stands for the label of the output update branch.

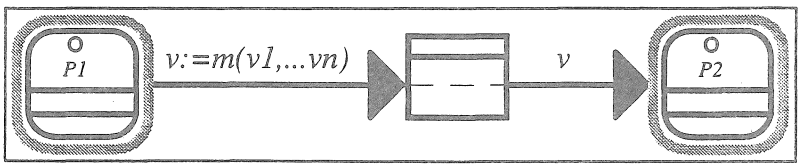


Figure 4 : Simplified Diagrammatic Representation for Object Place Update Only.

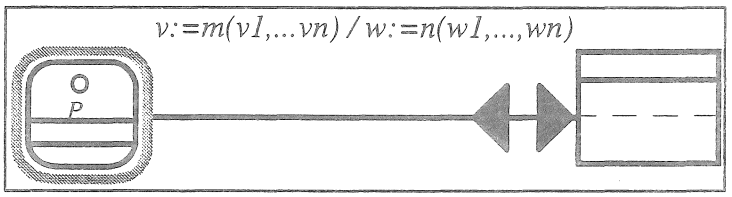


Figure 5 : Simplified Diagrammatic Representation for Object Content Update Only.

An activity is defined as an object with its own attributes and methods, and it is fully reusable. An activity can be carried out only if the input objects are present in their respective entry places, the output objects are absent from their exit places, and the enabling rule, if one exists, is satisfied. The actual execution of an activity is determined by the agent or set of agents that have this assignment, respected the interactions between the variables that label the branches and the transformation rule. Agents are not explicitly represented in the diagram. Each instance of an activity class represents a transaction carried out by an agent. Figure 6 presents an activity diagram.

The sequence of places that an object may be at in the diagram corresponds to the flow of documents in the office, such as, for example, *"P1"*, *"P2"* and *"P3"* for the object **O2**. The diagram defines a relative order among the activities, represented by an activity using an object/place produced by another one. In Figure 6, Activity 7 queries instances of **O2**-*"P2"* only after these are produced by either Activity 3 or Activity 4. Likewise, Activity 6 queries instances of **O3**-*"B"* after these are produced by Activity 3.

## 4. THE FORMAL MODEL

The complete activity model formalization is presented in [RUI 95]. This formalization is based on:

a) [SIL 94] for the definition of *type, class, method signature, class hierarchy, subtype relationship, consistent hierarchy of classes, defined attributes, inherited attributes, defined methods, inherited methods, consistent set of method signatures, property scope* and *well-defined scope of a rule*,
b) [LEC 88], for the definitions of *value, object, consistent set of objects, equality* and *interpretation*, and
c) [HEU 93], for the definitions of *branch labeled net, node, link, domain, mark, change, branch, border, entry mark, exit mark, marking, step, enabled activity, immediately reachable marking, reachability relation, marking class, reachability class, initial marking, compact CEM net system, concurrently enabled transitions*, and in *conflict transitions*.

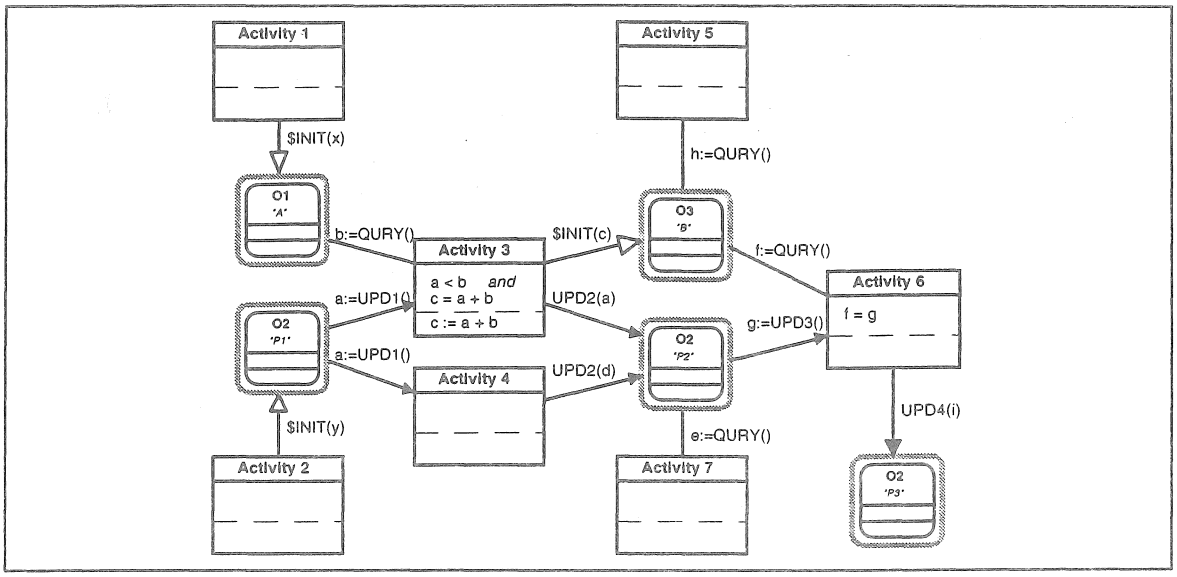


Figure 6 : Activity Diagram Example.

The first two sets of definitions above are originally related to the formalization of the DBMS O2 data model, whereas the third one underlies the formalization of compact nets for Condition/Event Modeling (compact CEM nets), a class of high-level Petri nets. Additionally, the following definitions for the activity model are presented in [RUI 95]: *method call, scheme of classes, structurally consistent scheme of classes, office object place, scheme of office classes, office class, office object, set of office objects for a place, net of office classes, activity, activity net scheme, activity net, decision-making in an activity for a marking M, activity with decision-making, activity context, generalization and aggregation.* All description and behavioral aspects of the office activity model are based on these definitions.

### 4.1 Generalization

Generalization is an abstraction technique allowing the description of the properties common to a set of entities in a higher-order entity. In the case of activities, it allows the description of the common properties shared by a set of different activities. These properties can be the objects/places handled, connected to the activity by labeled branches, and the activity enabling and transformation rules, in case they exist. After presenting the formal definition for the generalization concept, an activity generalization is exemplified in Figure 7, which is based on the diagram of Figure 6.

**Definition 1: (Generalization)**

Let $n_{a1}, n_{a2} \in A$ be activity names, and let $\Delta = (C, Tip, \prec, M, Tn)$ be a **scheme of office classes**.

Then, an activity $a_1=(n_{a1}, t_1, A(t_1), l_{t1}, A(l_{t1}))$ is a ***generalization*** of an activity $a_2=(n_{a2}, t_2, A(t_2), l_{t2}, A(l_{t2}))$ if and only if:

1. $\forall f_1, f_1 \in l_{t1}, f_1 = (c_1, t_1, b_1), \exists f_2 \in l_{t2}, f_2 = (c_2, t_2, b_2)$ such that:
   a) $b_1 = b_2 \wedge (c_1, c_2 \in C \wedge (c_1 = c_2 \vee c_2 \prec c_1))$
   b) $A(f_1) = A(f_2), A(f_1) \in A(l_{t1}) \wedge A(f_2) \in A(l_{t2})$
2. $A(t_{1h})$ is a *sub-formula* of $A(t_{2h})$ (sub-formula definition from [CAS 87]).
3. $\forall u_1 \in A(t_{1r}), u_1$ being a term, $\exists u_2 \in A(t_{2r}), u_2$ being a term, such that $u_1 = u_2$.

Based on [SIL 94], $\Delta = (C, Tip, \prec, M, Tn)$ is configured as follows:

(C, Tip, $\prec$) is a consistent hierarchy of classes, where C is a finite set of class names, and where these classes are heirs to **office object place** class;

M is a **consistent set of method signatures** over C;

Tn is a **set of named types**.

If Figure 7-b) is taken to illustrate these concepts, then :

$n_{a1}$ = Activity 4;

$n_{a2}$ = Activity 3;

$t_1$ and $t_2$ correspond to the activity icons, i.e. the three-layer rectangles;

$A(t_1)$ = ; (empty label, i.e. without neither a formula nor a term; $A(t_{1h})$ returns true);

$A(t_2) = (A(t_{2h}), A(t_{2r}))$

$A(t_{2h})$ = a < b *and* c = a + b *and* d = a; (enabling rule)

$A(t_{2r})$ = c := a + b; d := a; (transformation rule)

$l_{t1}$ corresponds to the following set of branches related to Activity 4: update input branch coming from O2 - *"P1"* and update output branch going to O2 - *"P2"*;

$l_{t2}$ corresponds to the following set of branches related to Activity 3: update input branch coming from O2 - *"P1"*, update output branch going to O2 - *"P2"*, query branch linked to O1 - *"A"*, and inclusion branch going to O3 - *"B"*;

$A(l_{t1})$ = a:=UPD1() and UPD2(d);

$A(l_{t2})$ = a:=UPD1(), UPD2(d), b:=QURY() and $INIT(c).

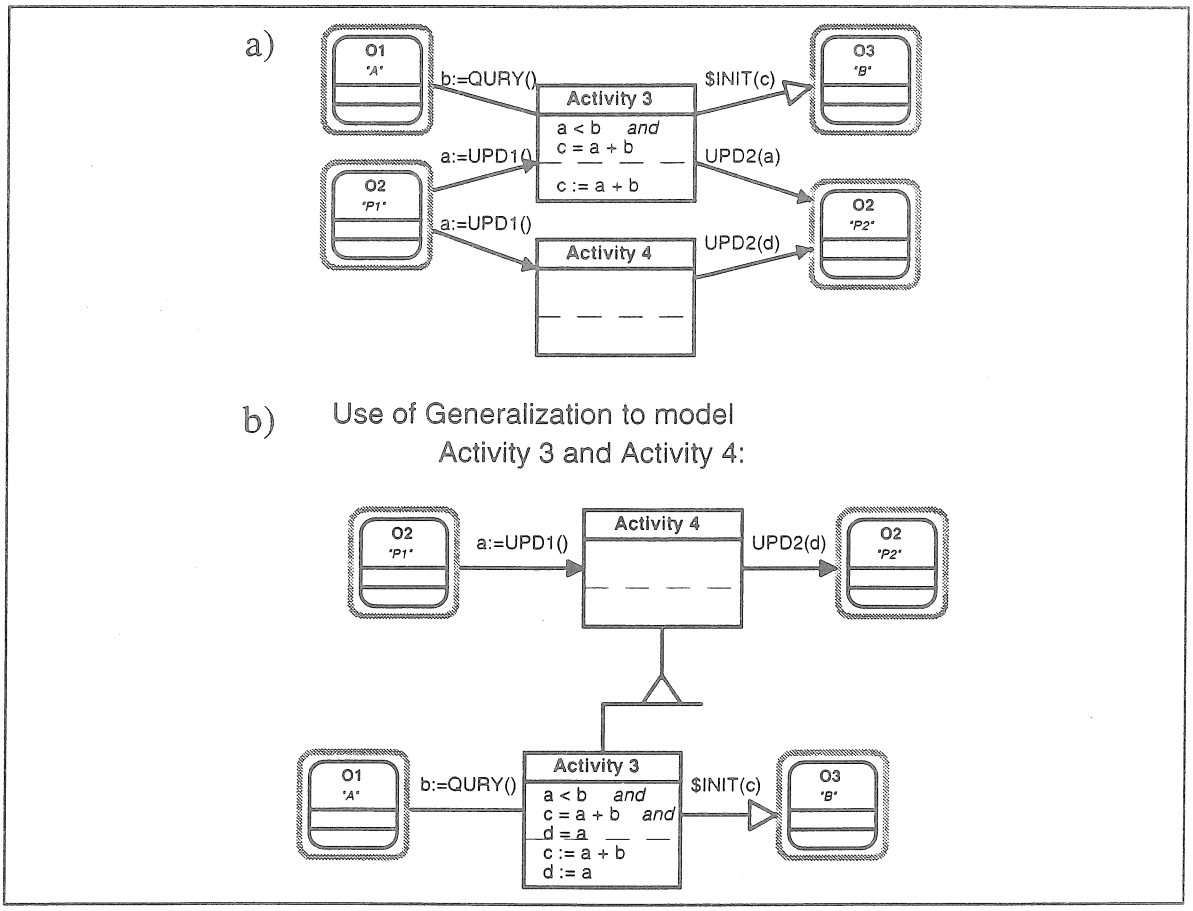


Figure 7 : Generalization Activity Diagram for a Hypothetical Application.

The pre-conditions of an activity are described by both the expressions labeling the branches, and the enabling rule. If there is a validation performed at the level of branch labels in a generic activity, this validation must be considered in all specializations of that activity as well. In Figure 7-a), Activity 3 updates the document place, from O2-*"P1"* to O2-*"P2"*: the variable associated to the branch label is a. Activity 4 does not present such a verification. In the diagram transformation of Activity 3 and Activity 4 this verification is described in the enabling rule (d=a), in order to allow Activity 4 to be a generalization

of Activity 3. In a specialized activity, the enabling rule is the one described locally, in addition to the one inherited from the generic activity (in case it exists). In Figure 7-b), Activity 4 does not have an enabling rule. Thus, the enabling rule for Activity 3 is a < b *and* c = a + b *and* d = a.

In conclusion, an activity A can be a generalization of an activity B if : (1) all objects and respective connection branches present in A are also included in B, (2) the enabling rules of activity A are also included in B, and (3) the transformation rules of A are also included in B.

### 4.2 Aggregation

Aggregation is a technique enabling the description of a set of entities of a reality as a higher-order entity, with its own characteristics. In the activity model, the aggregation technique is used to define properties a set of activities must satisfy. These properties are defined by the branches and their respective labels, as well as by the enabling and transformation rules (if these exist) in the aggregate activity. The local properties of the activities composing the aggregate are described directly in those activities. The formalization of this concept in the office activity model is presented below. Next, Figure 8 shows a possible aggregation of activities derived from the activity diagram depicted in Figure 6.

**Definition 2: (aggregation)**

Let $N_a = ((S, T, \beta; F); A, LA, D)$ be an **activity net**, let $a=(n_a, t, A(t), l_t, A(l_t))$ be an **activity**, let the pair $(a, C_a)$ be the **activity context** of $a$, and let $N_a' = ((S', T', \beta'; F'); A', LA', D')$ be another **activity net**, from now on called ***aggregated net*** of $a$. Consider that $M(C) \subseteq M$ is the set of marks associated to the set of office classes $C$.

The pair $((a, C_a), N_a')$ is an ***aggregation*** if and only if:

1. $\forall c \in C_a \rightarrow c \in S'$.
2. $\forall f, f \in l_t, f = (c, t, b), \exists f', f' \in F', f' = (c, t', b) \mid A(f) = A(f')$.
3. $\forall f', f' \in F', f' = (c', t', b'): c' \in C_a \rightarrow \exists f, f \in l_t, f = (c, t, b) \mid A(f') = A(f) \wedge c' = c \wedge b' = b$.
4. $\forall c' \in S' : c' \notin C_a \rightarrow c' \notin S$.
5. $\forall j \in Jl_t, j$ is a **change** defined by $l_t$,

**IF** there is $M_1$, a **marking** of $N_a$, such that $j$ is **enabled**, and if $M_2$ is the marking **immediately reachable** from $M_1$ through $j$ ($M_1[j>M_2$), with $M_1'(C_a) = M_1(C_a)$ and $M_2'(C_a) = M_2(C_a)$, $M_1'$ and $M_2'$ being markings of $N_a'$,

**THEN**, $M_2'$ must be **reachable** from $M_1'$, i.e. there is a **reachability relation** $R$ such that $M_1' \, R \, M_2'$.

For $N_a = ((S, T, \beta; F); A, LA, D)$ one has (based on [HEU 93]):

$(S, T, \beta; F)$ is an **office class net**:

The elements of $S$ are **office classes**,

$\beta$ = {**qur**, **exc**, **inc**, **upd-i**, **upd-o**}. The branch labels determine the five types of branches used in **office class nets**: query branches (**qur**), exclusion branches (**exc**), inclusion branches (**inc**), updating input branches (**upd-i**) and updating output branches (**upd-o**),

the elements of $F$ are *branches*,

the elements of $T$, $t$ are *transitions* such that for every $t$, $t \in T$, the tuple $(n_a, t, A(t), l_t, A(l_t))$ is an **activity**, for some $n_a \in A$;

LA is a first order language, called (net) ***annotation language***;

$A:(S \cup T \cup F) \rightarrow LA$ is called the ***annotation*** of $N_a$;

$D \subseteq F$, is a set of *basic definitions* or *axioms* that restrict the variety of acceptable interpretations of LA. F here is the set of formulas over LA; an interpretation I conforms to axioms D, or is ***conforming*** if and only if every formula in D is true under I.

$(a, C_a)$ is the context of activity $a$ , i.e. $C_a$ is the set of office classes connected to activity $a$.

If Figure 8 is taken to illustrate these concepts, then :

$a$=(Activity, $t$, ((a<b *and* i=a), (i:=a)), $l_t$, {b:=QURY(), a:=UPD1(), UPD4(i)});
$t$ is the activity icon (3-layer rectangle);
$l_t$ is the set of branches between Activity and O1 - *"A"*, O2 - *"P1"* and O2 - *"P3"*;
$C_a$ = {O1 - *"A"*, O2 - *"P1"*, O2 - *"P3"*};
$M(C_a)$ is a marking for the set $C$ of office classes, i.e. the set of office objects present in the places in a given moment.

In Figure 8, Activity is an aggregation of Activity 3, Activity 4, Activity 5, Activity 6 e Activity 7. The subdiagram inside the dashed circle is thus the detailed description of Activity. Notice however that Activity itself is an ordinary activity, in the sense that, to be carried out, it requires the presence of the input objects/places, the absence of output objects and the enabling rule fulfillment. Therefore, the properties represented in the aggregate activity do not need to be stated explicitly in the corresponding subdiagram.

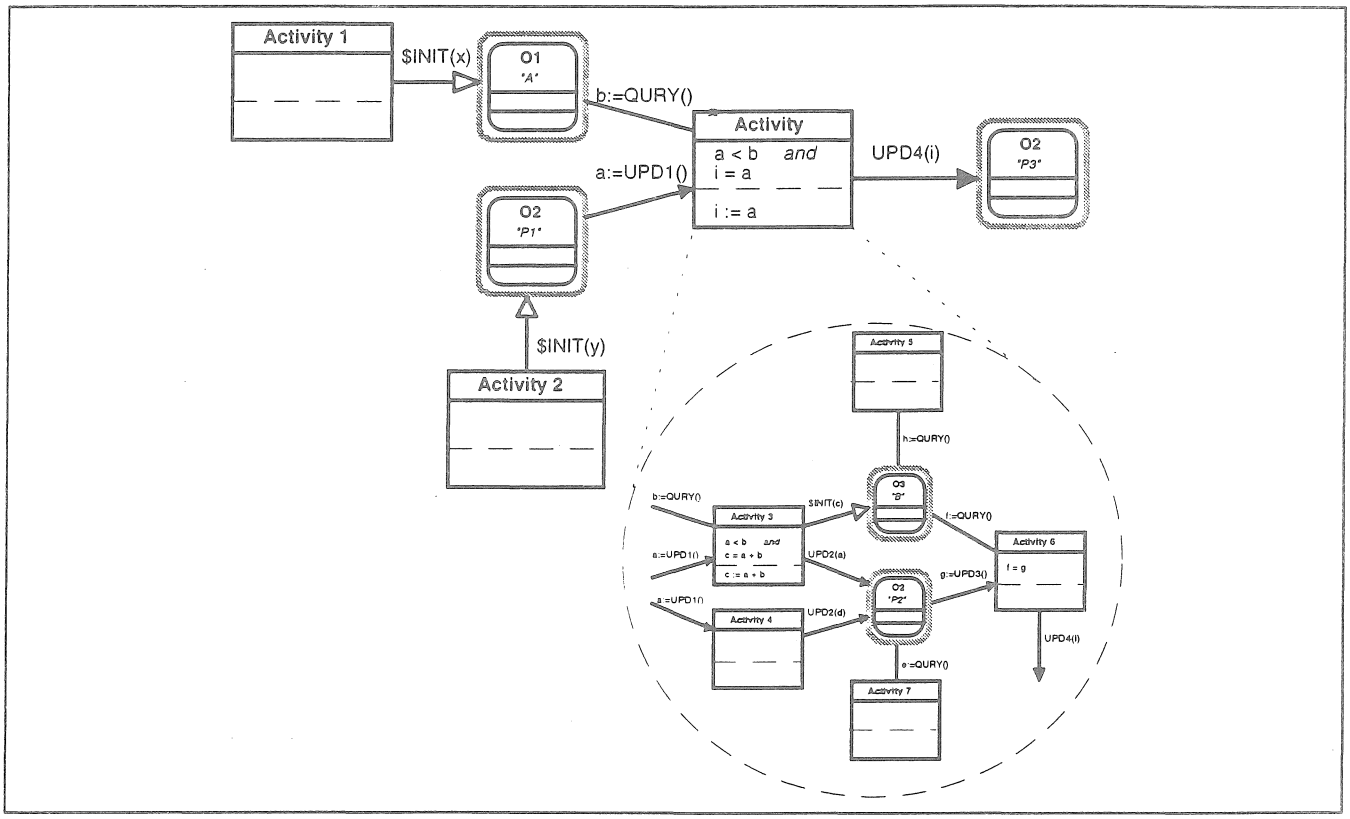


Figure 8: Aggregate activity and the corresponding activity subdiagram.

The execution of an aggregate activity obeys the sequence of steps described below, which are illustrated using Figure 8 :

a) the documents to be used by an activity are selected (e.g. an instance of O1 - *"A"* and an instance of O2 - *"P1"*), and the absence of documents in the output objects/places is checked, in case these can directly be mapped using the expressions labeling the input and output branches;
b) the enabling rule (a < b *and* i = a) is tested. This test comprises the comparison of the values assigned to variables a e b, as well as the verification that it does not exist instances in O2 - *"P3"*, for which the value of variable i is equal to the variable a in O2 - *"P1"*.

c) only after the above conditions are met, the execution of the activities in the subdiagram (Activity 3 or Activity 4) is started. After their execution is resumed, it is then tested the inclusion of an instance of O2 - *"P3"*, for which the value of variable i is the same one as the excluded instance in O2 - *"P1"*.

Consequently, a subdiagram that details an aggregate activity is only aware of the objects/places selected to the execution of the activity, instead of all present objects/places. Additionally, objects/places internal to the subdiagram represent work areas for the execution of the subactivities, or their execution "memory" (storage of the contents of previous executions).

The objects/places identified in the subdiagram that details an activity, correspond either to working documents, or to documents that support the execution of the subactivities (e.g. O3 - *"B"* ), or to intermediate stages in the flow of documents (e.g. O2 - *"P2"* ). These objects/places are local to this subdiagram, and they can only appear in the new subdiagrams that in turn detail this subdiagram.

The other elements mentioned in the definition are analogously associated to other elements of Figure 8.

## 5. CONCLUSIONS

This paper described the concepts of *generalization* and *aggregation* defined for the activity representation model for workflow applications. A specific workflow modeling language was developed and formalized to represent *activity diagrams*. Being the activity model an object-oriented model, these abstraction concepts enable the intensive reuse of application modeling results in similar or related contexts. The natural association of abstractions over a reality, together with the structuring and functioning characteristics intrinsic to organizations, allow the effective gain in quality and productivity resulting from the use of an IS specification tool based on the activity model. The dynamics of an application is described following the organizational and functional solutions adopted by an enterprise, as well as the assignments and responsibilities of the agents working in the organization. All the graphical representation is supported by the precise definition and associated formalism.

The modeling tool presently being evaluated aims at offering to designers a computer-based support appropriate to the development of workflow applications. Particular emphasis is given to component reuse, involving both static (objects) and dynamic (activities) aspects. The model is being used for real life case studies representation and mappings from the model concepts to workflow commercial analysis tools.

## 6. REFERENCES


[ANG 92] Ang, J.S.K. et al. Analyzing Information Systems using Petri Nets: Operations-Oriented Methodology. In: IInd. Intl. Conf. on Dynamic Modelling of Information Systems, Washington, D.C., 18-19, July, 1991. Proceedings. Amsterdam, H.G.Sol & R.C.Crosslin (eds.), North-Holland, 1992. p. 329-352.

[CAS 87] Casanova, M. A. et al. **Programação em Lógica e Linguagem Prolog** (Logic Programming and Prolog Language). São Paulo: Edgard Blücher, 1987. 461p. (in Portuguese)

[COA 91] Coad, P. & Yourdon, E. **Object-oriented Analalysis**, Second Edition, Prentice Hall, 1991.

[EDE 94] **T-ORM: Temporal aspects in objects and roles**, N. Edelweiss, J. Palazzo M.de Oliveira, J.M.V. de Castilho, E. Peressi, A. Montanari e B. Pernici, in Proceedings of the ORM-1 Conference, Magnetic Island, Austrália, July 4-7, 1994, p. 18-27.

[ELL 80] Ellis, C. A. & Nutt, G.J. Office Information Systems and Computer Science. **Computing Surveys**, New York, 12(1):27-60, Mar. 1980.

[ELL 91] Ellis, C. A. et al. Groupware: Some Issues and Experiences. **Communications of the ACM**. New York, 34(1):39-58 Jan. 1991.

[GIB 90] Gibbs, S. et al. Class Management for Software Communities. **Communications of the ACM**. New York, 33(9):90-103, Sept. 1990.

[HEU 93] Heuser, C. A. et al. Towards a Complete Conceptual Model: Petri Nets and Entity-Relationship Diagrams. **Information Systems**, Oxford, GB., v.18. n.5, p.275-298, July 1993.

[JOH 88] Johnson, R.E. & Foote, B. Designing Reusable Classes. **Journal of Object-Oriented Programming**. New York, 1(4):22-35, June-July 1988. Apud [GIB 90].

[KAP 91] Kappel, G. & Schrefl, M. Object/Behavior Diagrams. In: IEEE Data Engineering Conference. Kobe, Japan, 8-12 April, 1991. **Proceedings**. Los Alamitos, Ca., USA, IEEE Press, 1991. p. 530-9.

[KOR 93] Korth, H.F.; Silberschatz, A. **Database Systems Concepts**. New York: McGraw-Hill, 1991.

[KUN 93] Kung, D.C. The Behavior Network Model for Conceptual Information Modeling. **Information Systems**. London, GB., 18(1):1-21, Jan. 1993.

[LEC 88] Lécluse, C. et al. $O_2$, and Object-Oriented Data Model. In: ACM INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD'88, Chicago, June 01-03, 1988. **Proceedings...** New York, ACM Press, 1988. 446p. p.424-433.

[LEW 91] Lewis, J.A. et al. An Empirical Stydy of the Object-Oriented Paradigm and Software Reuse. In: OOPSLA'91 Conf. on Object-Oriented Programming Systems, Languages and Applications, Phoenix, 6-11, Oct., 1991. Proceedings. **ACM Sigplan Notices**, New York, 26(11):184-196, Nov. 1991.

[LIU 92] Liu, L. & Meersman, R. Activity Model: a Declarative Approach for Capturing Communication Behaviour in Object-Oriented Databases. In: VLDB'92, 18th. Intl. Conf., Vancouver, CA., 23-27, Aug., 1992. **Proceedings**. San Mateo, Li-Yan Yuan (ed.), Morgan-Kaufmann, 1992. p. 481-493.

[McG 92] McGregor, J.D. & Sykes, D.A. **Object-Oriented Software Development: Engineering Software for Reuse**. New York, Van Nostrand Reinhold, 1992.

[PER 89] Pernici, B. et al. C-TODOS: An Automatic Tool for Office System Conceptual Design. **ACM Transactions on Information Systems**, New York, 7(4):378-419, Oct. 1989.

[RUI 89] **Electronic Forms: Modeling the System Dynamics.** D.D.A. Ruiz & J. Palazzo M. de Oliveira, In: Proceedings of ICCI '89 - International Conference on Computing and Information, Toronto, Canada, May 23-29, 1989, Canadian Scholar Press.

[RUI 95] Ruiz, D. D. A. **Um Modelo para Representação de Atividades em Aplicações de Escritórios** (A Model for Representing Activities in Office Applications). Porto Alegre, PGCC da UFRGS, 1995. Doctoral Dissertation. (in Portuguese)

[SIL 94] Silva, M. P.; Lanzelotte, R. S. G. Formalizando a Evolução de Esquemas em SGBDs Orientados a Objetos (Formalizing the Scheme Evolution in OODBMS). In: SIMPÓSIO BRASILEIRO DE BANCO DE DADOS, 9., 5-7 Set., 1994, São Carlos, SP. **Anais**... São Carlos - SP: USP-São Carlos, 1994. 381p. p. 157-172. (in Portuguese)

[SMI 77] Smith, J. M.; Smith, D. C. P. Database Abstractions: Aggregation and Generalization. **ACM Transactions on Database Systems**, New York, v.2, n.2, p.105-133, June 1977.

[WEI 89] Weiser, S.P. & Lochovsky, F.H. OZ+: An Object-Oriented Database System. In: **Object-Oriented Concepts, Databases and Applications**. New York, W.Kim el al. (eds.), ACM Press, 1986. p. 309-337.