A Reference Model for Team-Enabled Workflow Management Systems

W.M.P. van der Aalst[1] and A. Kumar[2,3]

[1] Faculty of Technology Management, Eindhoven University of Technology,
PO Box 513, NL-5600 MB, Eindhoven, The Netherlands.
[2] Database Systems Research Department, Bell Laboratories,
600 Mountain Ave., 2A-406, Murray Hill, NJ 07974, USA.
E-mail: w.m.p.v.d.aalst@tm.tue.nl, akhil@acm.org

Abstract

Today's workflow systems assume that each work item is executed by a *single* worker. From the viewpoint of the system, a worker with the proper qualifications selects a work item, executes the associated work, and reports the result. There is usually no support for teams, i.e., groups of people collaborating by jointly executing work items (e.g., the program committee of a conference, the management team of a company, a working group, and the board of directors). In this paper, we propose the addition of a team concept to today's workflow management systems. Clearly, this involves a marriage of workflow and groupware technology. To shed light on the introduction of teams, we extend the traditional organizational meta model with teams and propose a *Team-enabled Workflow Reference Model*. For this reference model and to express constraints with respect to the distribution of work to teams, we use OCL (Object Constraint Language).

Keywords: Workflow management systems, Team-enabled Workflow Reference Model, Computer supported cooperative work, groupware, organizational models.

1. Introduction

Most publications on workflow management focus on the process (or control-flow) perspective, neglecting the representation of organizational structures and the distribution of work [26], as they relate to a workflow management system. Thus, there is a lack of consensus on the type of organizational structures to be supported. For example, consider how the Staffware system supports the concept of a so-called work queue. Both workers and work items are assigned to work queues. A worker may be linked to multiple work queues and a work queue may be visible to multiple workers, in which case it is called a group queue. Each worker also has a personal queue. Personal work queues can be used to support a *push* mechanism, i.e., work items are assigned to specific workers. On the other hand, group queues can be used to support a *pull* mechanism, i.e., multiple workers can view a shared pile of work items and select specific work items. Other workflow systems use other paradigms: IBM's MQ Series Workflow [20] supports both organizations and roles instead of one queue mechanism. Another example is the workflow management system COSA [8], which supports arbitrary organizational dimensions (e.g., groups, roles, authorization, etc.) and merges all relevant work items into one personalized list. The fact that the available systems are quite different with respect to their handling of organizational issues is demonstrated by the varying support for delegation: Systems either have no support for delegation or offer rather specific functionality. Another aspect in which systems are quite different is the distinction between authorization and work distribution. In many systems authorization (the ability to execute a

[3] On leave from the University of Colorado, Boulder.

work item) and distribution (assigning tasks to workers) coincide. (Recall the work queue paradigm in Staffware.) Other systems such as FLOWer by Pallas Athena allow for a clear separation of authorization and work distribution. This lack of consensus is also illustrated by the absence of any proposals from the Workflow Management Coalition (WfMC, [24]) concerning the representation of organizational structures and the distribution of work. Although there is a working group on resource modeling (WfMC/WG9), no standards have been proposed. The absence of consensus is an important problem and has been addressed recently by some authors [26,27].

The scope of this paper is limited to the representation of organizational structures and the distribution of work *in the context of team support*. To the best of our knowledge, all commercial workflow products assume a functional relation (in the mathematical sense) between (executed) work items and workers, i.e., from the *viewpoint* of the workflow management system each work item is executed by a single worker. A worker selects a work item, executes the corresponding actions, and reports the result. It is not possible to model or to support the fact that a group of people, i.e., a *team*, executes a work item. Note that current workflow technology does not prevent the use of teams: Each step in the process can be executed by a team. However, only one team member can interact with the workflow management system with respect to the selection and completion of the work item. Thus, current workflow technology is not cognizant of teams. This is a major problem since *teams are very relevant when executing workflow processes*. Consider for example the selection committee of a contest, the management team of a subdivision, the steering committee of an IT project, and the board of directors of a car manufacturer. In addition to providing explicit support for modeling teams, it is also important to recognize that individuals typically perform different roles within different teams. For example, a full professor can be the secretary of the selection committee for a new dean, and the head of the selection committee for tenure track positions. These examples show that modeling of teams should be supported by the future generation of workflow products. In this paper, we explore concepts and technologies for making workflow management systems team enabled.

Groupware technology ranging from message-based systems such as Lotus Notes to group decision support systems such as GroupSystems offer support for people working in teams. However, these systems are not equipped to design and enact workflow processes. Based on this observation a marriage between groupware technology and workflow technology seems to be an obvious choice for developing team-enabled workflow solutions. Systems such as Lotus Domino Workflow [28] provide such a marriage between groupware and workflow technologies. Unfortunately, these systems only partially support a team working on a work item. For example, in Lotus Domino Workflow, for each work item one needs to appoint a so-called activity owner who is the only person who can decide whether an activity is completed or not, i.e., a single person serves as the interface between the workflow engine and the team. Clearly such a solution is not satisfactory.

As a starting point for investigating team-enabled workflow management systems, we take a basic *organizational meta model*. This model serves as a reference model for the basic functionality offered by today's workflow management systems. This model is a simplification of the meta models of existing workflow management systems and the meta models proposed in literature (cf. [6,23,26,27]). Next, we focus on the addition of teams. Therefore, we take the "greatest common divisor" of existing organizational meta models and add the concept of teams. We use UML class diagrams to represent the basic and extended meta model. Moreover, we clearly define the constraints in terms of OCL. OCL (Object

Constraint Language, [30,31,39]) is an integral part of the UML (version 1.1 and upwards, [16,32]). OCL is a powerful language to describe constraints at the meta level. For example, it is possible to specify that a worker not having the required role cannot execute a work item. OCL can also be used to describe constraints specific for the organization or the workflow process, e.g., "The department head should either approve or pre-check each purchase." and "Insurance claims involving more than $5000 should not be handled by an office clerk but by a trained expert." Examples will show that many constraints at the meta, organizational, and process level can be expressed quite easily using OCL.

Based on the team-enabled reference model we explore various aspects of work distribution in the presence of teams. For example, teams can vote on the outcome of a successfully completed work item. In fact, the completion of a work item executed by a team could be subject to discussion, e.g., there can be a conflict: Some team members may dispute the completion of work item reported to be finished by other team members. In the traditional setting, one worker indicates the completion of a work item. This is not necessarily the case for teams. Other issues related to the operation of a team are: working at same time/different time, same place/different places, scheduled/ad-hoc meetings, etc. We will classify and structure these issues in more detail later and discuss possible realizations of the team concept. Clearly, team-enabled workflow management systems should borrow concepts or components of existing groupware technology. Therefore, we propose a marriage between workflow and groupware technologies, and give an architecture for it.

The remainder of this paper is organized as follows. First we introduce the basic workflow concepts, a simple organizational meta model, and OCL as a language to express workflow constraints. In Section 3, we introduce the team concept and extend the organizational meta model to incorporate support for teams. We also provide generic (i.e., meta level) and specific (i.e., organizational/process level) constraints. In Section 4, we explore team allocation mechanisms. Then we discuss possible realizations using groupware technology. Section 6 concludes this paper.

2. Workflow management and organizational models

In this section we first introduce our terminology such as case, task, resource, role and work item, and then present a meta model for organizational modeling of work distribution. We also show how OCL can be used to model both generic and specific constraints. Note that the team concept is introduced only in Section 3.

2.1. Workflow management concepts

The fundamental property of a workflow process is that it is *case-based* [1]. This means that every piece of work is executed for a *specific case*. Examples of cases are an insurance claim, a tax declaration, a customer complaint, a mortgage, an order, or a request for information. Thus, handling an insurance claim, a tax declaration, or a customer complaint are typical examples of workflow processes. Cases are usually generated by an external customer. However, it is also possible that a case is generated by another department within the same organization (an internal customer). A typical example of a process that is not case-based, and hence not a workflow process, is a production process such as the assembly of bicycles. The task of putting a tire on a wheel is (generally) independent of the specific bicycle for which the wheel will be used. Note that the production of bicycles to order, where procurement,

production, and assembly are driven by individual orders, can be considered as a workflow process.

The goal of workflow management is to handle cases as efficiently and effectively as possible [23,24]. A workflow process is designed to handle large numbers of similar cases. Handling one customer complaint usually does not differ much from handling another customer complaint. The most important aspect of a workflow process is the *workflow process definition* [1]. This process definition specifies the *order* in which tasks must be executed. Alternative terms for workflow process definition are "procedure," "workflow schema," "flow diagram," and "routing definition". Tasks are ordered by specifying for each task the *conditions* that need to be fulfilled before it may be executed. In addition, it is specified which conditions are fulfilled by executing a specific task. Thus, a partial ordering of tasks is obtained. In a workflow process definition, standard routing elements are used to describe sequential, alternative, parallel, and iterative routing thus specifying the appropriate route of a case. The workflow management coalition (WfMC) has standardized a few basic building blocks for constructing workflow process definitions [24]. An *OR-split* is used to specify a choice between several alternatives; an *OR-join* specifies that several alternatives in the workflow process definition come together. An *AND-split* and an *AND-join* can be used to specify the beginning and the end of parallel branches in the workflow process definition. The routing decisions in OR-splits are often based on data such as the age of a customer, the department responsible, or the contents of a letter from the customer. For these basic workflow patterns we refer to [24]. For more advanced patterns we refer to the workflow patterns web site www.tm.tue.nl/it/research/patterns (cf. [2]).

Many cases can be handled by following the same workflow process definition. As a result, the same task has to be executed for many cases. A task that needs to be executed for a specific case is called a *work item*. An example of a work item is the order to execute task "send refund form to customer" for case "complaint of customer Baker". Most work items need a *resource* in order to be executed. A resource is either a machine (e.g., a printer or a fax) or a person (participant, worker, or employee). Besides a resource, a work item often needs a *trigger*. A trigger specifies who or what initiates the execution of a work item. Often, the trigger for a work item is the initiative of the resource that must execute the work item. Selecting a work item from a work list (work queue or in-tray) corresponds to generating a so-called *resource trigger*. Other common triggers are *external triggers* and *time triggers*. An example of an external trigger is an incoming phone call of a customer; an example of a time trigger is the expiration of a deadline. A work item that is being executed is called an *activity*. If we take a photograph of the state of a workflow, we see cases, work items, and activities (see Figure 1). Work items link cases and tasks. Activities link cases, tasks, triggers, and resources.

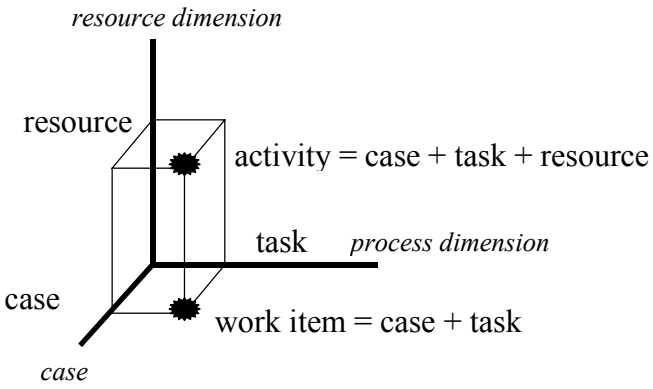


Figure 1: The relation between cases, resources, tasks, work items, and activities.

From the viewpoint of maintenance and flexibility, it is not particularly wise to assign work items to specific people. Rather, it is better to decouple the workflow process definition and the organizational structure and population. Resources, ranging from humans to devices, form the organizational population and are mapped onto *roles*. In office environments, where workflow management systems are typically used, the resources are mainly human. However, because workflow management is not restricted to offices, we prefer the term *resource*. To facilitate the allocation of work items to resources, resources are grouped into roles. A *role,* also referred to as *resource class*, is a group of resources with similar characteristics. There may be many resources in the same role and a resource may be a member of multiple roles. A role may be based on the capabilities (i.e., functional requirements) of its members. The classification into roles may also be based on the structure of the organization, e.g., team, organizational unit, branch, or department. Note that we use the term "role" in a broader sense than is common, say, in CSCW/workflow literature, where the role concept is typically restricted to a classification based on functional requirements. In this paper, *any class of resources (human or other) can serve as a role.*

Constraints also play an important role in workflow management, especially when the structure and policies of an organization are taken into account and security considerations are important. Considerable work on constraints has been done in the context of the RBAC (Role-Based Access Control) model [15,33,34]. The salient features of RBAC are that permissions are associated with roles and users are made members of roles thereby acquiring the associated permissions. Extensions of RBAC and other constraint related issues and algorithms are discussed in [3,4,5,7,29,36,40]. However, we will not go into the details here since that is not the main focus of this paper.

2.2. Organizational meta model

The workflow concepts just introduced will be structured using a meta model represented by a UML class diagram [16,32]. Several authors have developed meta models for structuring workflow concepts. Consider for example the work by Zur Mühlen [26,27] who evaluated the organizational capabilities of workflow products using meta models. For the purpose of this paper it is not meaningful to construct a detailed organizational meta model. There is no consensus on the functionality of workflow management systems with respect to organizational modeling and work distribution. Therefore, any attempt to construct a detailed organizational meta model would rule out most of the existing products. Instead of providing a detailed model, we give a basic model supported by most of today's workflow management systems.

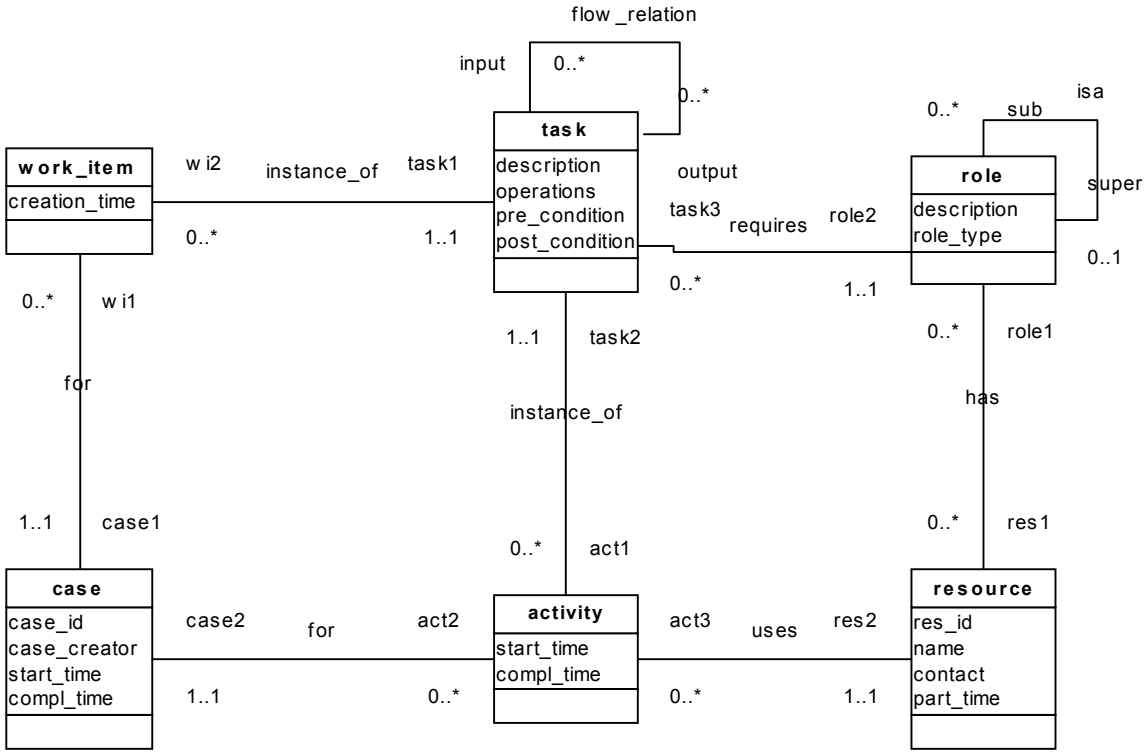


Figure 2: Basic organizational meta model.

Figure 2 shows the UML class diagram representing the basic meta model. The class diagram consists of classes (denoted by squares) and relationships (associations and generalizations denoted by lines and arrows). It is assumed that the reader is familiar with the UML notation. The four classes on the left-hand-side of the diagram correspond to the concepts case, task, work item, and activity. A work item is an instantiation of a task for a given case. A work item corresponds to precisely one task, i.e., the multiplicity of the role[1] *task1* of association *instance_of* is *1..1*. A task may correspond to arbitrarily many work items, i.e., the multiplicity of the role *wi2* of association *instance_of* is *0..**. Similarly, a work item corresponds to one case and one case may correspond to multiple work items. Work items correspond to concrete pieces of work, while tasks are abstract and defined at the level of the workflow process definition. An activity is also an instantiation of a task for a given case and corresponds to the actual execution of a task. Precisely one resource is associated with an activity denoted by the multiplicity *1..1* of role *res2* of association *uses*. The class *activity* is related to the class *work_item*. A work item becomes an activity when a worker starts executing the corresponding task for the corresponding case. Therefore, we could have given a different class diagram with an association between the two classes (instead of the *for* and *instance_of* associations). We did not do this to simplify the navigation using OCL and to avoid the dependency of activities on work items (e.g., a work item may be removed when the corresponding activity is launched). The class *role* is used to specify the mapping of tasks onto resources. Generally, a role concept is used to decouple the workflow process definition from concrete resources, e.g., directives such as "Task *approve contract* should be executed by *Bill Smith*" should be avoided. Therefore, each task is assigned to a role, e.g., "Task *approve contract* should be executed by someone with the role *manager*". Moreover,

[1] Note that the term role is used in UML to denote the end of an association.

resources are mapped on roles, e.g., "*Bill Smith* has the role *manager*". One resource can have multiple roles and several resources may share the same role.

The multiplicity of role *role2* of association *requires* is *1..1*, indicating that a task is mapped onto one role. In several workflow management systems it is possible to combine roles using expressions such as "The union of roles A and B", "Someone with role A or B, but not role C". To keep the meta model as simple as possible and consistent with systems such as Staffware [37], which allow for just one role (called queue) per task, we assume that each task can be mapped onto one role. Note that any expression in terms of roles can be mapped onto a virtual role whose members are determined by calculating the expression.

Figure 2 shows that roles can be related via the *isa* association. One role can be a subclass or superclass of another role. If role A is a subclass of role B, then resources with role A can execute tasks mapped onto role B. Thus, a subclass role is "superior" to its parent class role, in the sense that it can perform all the tasks of its parent classes and more. Several workflow management systems allow for such a relationship between roles (e.g., COSA) and the concept is quite useful for organizational modeling, cf. the role hierarchies in RBAC [33]. We will refer to this concept using the term *role inheritance*. Note that the *isa* association is a generalization at the instance level and not at the class level. Therefore, the *isa* association is not a generalization in UML terms.

Association *flow_relation* refers to causal dependencies between tasks. These flow relations are used to model sequential, conditional, parallel, and iterative routing. Note that the focus of Figure 2 is on organizational aspects of workflow management. Therefore, we limit ourselves to mentioning the existence of such flow relations, and refrain from detailed and system specific discussions about AND/OR-splits/joins, etc.

Figure 2 also lists some representative attributes. These attributes are illustrative and not exhaustive. In particular, we draw attention to the *role_type* attribute in class *role*. As indicated in Section 2.1, a role may be based on functional requirements (e.g., qualifications, capabilities, or competences), organizational requirements (e.g., teams, organizational units, branches, or departments), or positions within organizational entities (e.g., department head, dean of a faculty). Therefore, examples of role types include *qualification*, *position,* and *competence*. An example of a role of type qualification is "full professor", an example of a role of type position is "vice president of the board of directors", and an example of a role of type competence is "speaks Dutch". The distinction between these subclasses is rather arbitrary and not relevant for the remainder. However, it is important to note that roles are associated with the actual execution of work and not with issues like responsibility and accountability.

Let us consider a few workflow management systems and map these systems onto the meta model shown in Figure 2.

Staffware 2000 (Staffware PLC, [37]) supports the concept of work queues. A work queue can be compared to a role. Each worker (i.e., resource) has a private work queue and may have multiple group queues. The work items in the group queue are visible to all group members. Staffware does not allow for role inheritance ("Role A is a subclass of role B") or role expressions ("Someone with role A and role B"). Therefore, subclassing and role expressions need to be handled explicitly, i.e., by adding resources of the subclass to the superclass and by adding an additional role for each expression.

MQ Series Workflow (IBM, [20]) allows for the definition of roles and organizations. Organizations are organized hierarchically and a worker can be a member of only one organization. For each task and process it is possible to specify criteria based on organizations, roles, and levels. Organizations can be considered as special roles that are grouped hierarchically. Role inheritance is only supported for these organizational roles. Advanced role expressions ("Someone with role A and B but not role C") are not supported and, similar to Staffware, these role expressions need to be handled by additional roles.

COSA 3.0 (Ley GmbH, [8]) allows for advanced constructs with respect to organizational modeling and work distribution. COSA uses groups and group expressions to distribute work. COSA groups correspond to roles. Using the COUE (COSA User Editor) tool one can define an arbitrary number of role hierarchies (e.g., role hierarchies based on qualification, position, and competence), which can be deployed in parallel. It is possible to use arbitrary complex role expressions and role inheritance is supported.

Lotus Domino Workflow (Lotus/IBM, [28]) allows for the definition of workgroups, departments, and roles. People can be assigned to workgroups, departments, and roles. A person can belong to only a single department but can be assigned to multiple workgroups and roles. The Lotus Domino Workflow concepts workgroup, department, and role can be seen as special cases of the role concept used in Figure 2. Lotus Domino Workflow is one of the few systems which actually supports a team concept. A team is a group of people working on one activity. Unfortunately, all interaction between the team members and the workflow is through a so-called activity owner. The activity owner is the only person who can decide whether an activity is started or completed. Moreover, there is neither explicit modeling of teams nor any support for people working in teams. The only team-related functionality supported by Domino Workflow is the sharing of documents.

These four workflow products are quite representative of the current generation of production workflow systems. Only one of these systems (Lotus Domino Workflow) supports the concept of teams. However, team interaction is mainly limited to the sharing of documents in this system.

2.3. Modeling organizational constraints with OCL

The organizational meta model shown in Figure 2 does not express constraints that need to be satisfied, e.g., any resource may be attached to a particular work item. As indicated in the introduction, we use OCL (Object Constraint Language, [30,31,39]) to express constraints at the meta, organizational, and process level.

OCL is an integral part of the UML (version 1.1 and upwards, [16,32]). It has been developed within IBM and allows for the definition of integrity constraints. OCL has also been used to formalize the meta model of UML. OCL is based on set theory and can be used to specify invariants on classes and the relationships among classes in UML class diagrams.

2.3.1. Generic OCL constraints

Consider the organizational meta model shown in Figure 2. The model shows several multiplicity constraints, also referred to as cardinality constraints, e.g., every activity uses exactly one resource. An example constraint, which cannot be expressed as a simple cardinality constraint, is the requirement that activities should only be executed by resources

having the proper role, i.e., the resource should have the role associated with the corresponding task. Ignoring role inheritance, let us express this constraint in OCL.

activity
(C.1) self.task2.role2.res1–>includes(self.res2)

The name of the class underlined is the *context* of the constraint, an occurrence of *self* in it refers to any instance of that class. For example, in constraint (C.1) above, *self* refers to an instance of class *activity*. Starting from a specific object, we can navigate a path along a series of associations in the class diagram (Figure 2) to refer to other objects and their properties. For navigation along associations, we use the role[2] names on the opposite association end point, e.g., *self.task2* is the task corresponding to the activity represented by an instance of *activity* and *self.res2* is the resource corresponding to *self*. If the multiplicity of the association end point is 0..1 or 1..1, then the value of such an expression is an object. If this is not the case, navigation will result in a set of objects. For example *self.res2.act3* gives all activities executed by the resource *self.res2. self.task2.role2.res1* is the set of resources having role *self.task2.role2,* and *self.task2.role2* is the role of the task corresponding to activity *self*. OCL provides many operators/operations for reals, integers, strings, booleans and enumerations. OCL also provides several set operations. One of these operations is *includes* which tests the presence of an element in a given set. For example, (C.1) is a boolean expression which is true if resource *self.res2* is an element of the set of resources given by *self.task2.role2.res1*. Clearly this boolean expression corresponds to the desired constraint "Activities should only be executed by resources having the proper role".

One central concept in OCL is that of *collections*. There are three types of collections: sets, sequences, and bags. The operation *includes* used in (C.1) is defined on collections, and operations defined for collections can also be applied to sets, sequences, and bags. The arrow "**–>**" is used to access collections. Within OCL all collections of collections are automatically flattened, i.e., {{1,2},{3,4},{5,6}}={1,2,3,4,5,6}. The dot "." notation can be used to navigate along associations (and other relations), and also to access attributes of objects.

To illustrate OCL in more detail, consider a few examples in the context of class *case* (i.e., *self* is a contextual instance of *case*).

- *self.case_creator* is the initiator of the instance.
- *self.act2.res2.name* is the collection of all names of resource working on *self*.
- *self.act2.res2–>select(part_time < 0.50)* is the collection of all resources working on case *self* and working less that 50 percent.
- *self.act2.res2–>select(part_time <0.50)–>size* is the number of resources working part-time (i.e. less than 50 percent) and working on *self*.
- *self.act2.start_time* is the set of all execution dates of activities corresponding to case *self*.
- *self.act2.forall(start_date > self.start_date)* requires that all activities belonging to case *self* be executed after the start date of the case.

Constraint (C.1) did not take into account role inheritance. For example, if role A is a subclass of role B, then tasks with role B can be executed by resources having role A. There are two ways to deal with this constraint. Implicit inheritance assumes that if role A is a subclass of role B, then all resources having role A also have role B. For explicit inheritance we do not

[2] To avoid confusion, we use the UML term "role" as little as possible. Instead we use the term "association end".

make this assumption, i.e., it is possible to have resources which have role A but not role B. We will formalize both types of constraints using OCL.

For *implicit role inheritance* we use the following OCL constraints:

activity
(C.2) self.task2.role2.res1–>includes(self.res2)
role
(C.3) self.res1–>includesAll(self.sub.res1)

(C.2) is same as (C.1). (C.3) takes an arbitrary role *self* and states that the set of resources having this role (*self.res1*) contains all resources of all its subclasses (*self.sub.res1*).

For *explicit role inheritance* we need to relax (C.2) and limit the scope of (C.3). Although role A is a subclass of role B, there may be resources which have role A but not role B. Nevertheless, since role A is "superior" to role B, these resources can execute tasks which require role B. To allow for such a form of inheritance, we can use the following OCL constraint:

activity
(C.4) self.task2.role2.res1–>includes(self.res2) or self.task2.role2.sub.res1–>includes(self.res2)

This constraint is a two-part boolean expression separated by boolean **or**. The first part corresponds to (C.2). The second part takes subclasses into account. *self.task2.role2.sub* is the set of all subclasses of the required role. Note that inheritance is limited to one level, i.e., only the direct subclasses are taken into account as opposed to *all* subclasses. This corresponds to the notion of *limited inheritance* introduced in [33]. Sometimes, it is useful to not limit inheritance to one level, and allow two or more levels. While it is easy to specify, using OCL, that inheritance is limited to a fixed number levels, it is very hard to specify a constraint which allows for an arbitrary number of levels because transitive closure is not supported in OCL [25]. Mandel and Cengarle [25] have shown that OCL is not equivalent to a Turing machine and that the OCL expression for the transitive closure of a binary relation is very long, tricky, and far from intuitive. Therefore, we use the shorthand notation *sub** which corresponds to the transitive closure of relation *isa* in Figure 1 projected onto association end *sub*. Using this shorthand notation full explicit role inheritance is specified as follows:

activity
(C.5) self.task2.role2.sub*.res1–>includes(self.res2)

2.3.2. Specific OCL constraints

The OCL constraints discussed thus far are generic, i.e., at the meta level. OCL can also be used to reflect constraints specific for a particular organization or a particular workflow process.

Let us first consider an example of constraints at the organizational level. The following OCL constraint specifies a rather rigorous separation of duty rule:

activity
(C.6) not(self.case2.act2–>excluding(self).res2–>includes(self.res2))

self.case2.act2 are all activities of the case activity *self* belongs to. *self.case2.act2–> excluding(self).res2* is the set of resources executing *other* activities of this case. This set should not include the resource executing *self*. Hence, the OCL constraint is a boolean expression requiring that *no resource should execute multiple activities for the same case* (separation of duties idea).

Assume that the organization has two workers named "Bill" and "Al", who must not work on the same case. Assuming an attribute *name*, this can be expressed as follows:

case
(C.7) not(self.act2.res2.name–>includesAll(Set{'Bill', 'Al'}))

The standard OCL operation *X–>includesAll(Y)* returns true if the entire collection Y is included in collection X. Similarly, it is possible to express constraints specific for a given workflow process. Assume that class task has the attribute *name*. The following constraint specifies that *the two tasks, "transfer money" and "make decision", should not be executed by the same resource*.

case
(C.8) (self.act2–>select(a | a.task2.name = 'transfer money')).res2 –>
intersection((self.act2–> select(a | a.task2.name = 'make decision')).res2) –> isEmpty

The standard OCL operation *select* is a filter such that $X\text{–>}select(x \mid Bx) = \{ x \in X \mid Bx \}$. The constraint requires that the *intersection* set of the resources performing these two tasks on a case should be *empty*.

The examples above show that OCL can be used to express constraints at the level of the organizational meta model, the organizational level, and the process level.

3. Adding teams to organizational workflow models

In this section we introduce the team concept. First, we discuss the concept. Then, we provide the meta model and the corresponding OCL constraints at the meta level. Finally, we illustrate the capability of OCL to express organization and process specific constraints.

3.1. The team concept

In the previous section, we discussed the functionality of today's workflow management systems with respect to organizational modeling and work distribution. As far as we know, none of the commercial systems available supports the concept of *teams*. The Webster dictionary defines team as "a number of people working together on a common task." If we translate this to workflow terminology, a team can be defined as a group of resources (i.e., workers, participants) working together on a single work item. In existing workflow systems work items are distributed over resources. Although a work item may be offered to many resources, from the perspective of the workflow management system, a work item is still executed by one resource. Consider the association *uses* in Figure 2: Each activity, i.e., a work item being executed, corresponds to one resource. Clearly, it is possible to bypass the workflow management system and have the work item executed by a team. This means that one team member acts as the liaison between the team and the workflow management system. This team member selects the work item from his in basket and reports the completion of the

corresponding activity to the workflow system. However, the absence of a team concept is a serious deficiency, and hence, workflow management system should support teamwork. From a security viewpoint, it is important to be able to specify requirements on the team structure and team members. For enactment, it is important to have mechanisms to support team collaboration and to support team decision processes. For management and accounting purposes, it is important to be able to trace team membership and contributions of individual resources. Therefore, we propose team-enabled workflow management systems. Examples of teams that could benefit from such a workflow management system are:

- the program committee of a conference consisting of a chair and 12 members,
- the management team of a subdivision consisting of a general manager, an engineer, a sales representative, and a secretary,
- the multidisciplinary team of medical specialists treating patients with Parkinson's disease, and
- the design team of a new car.

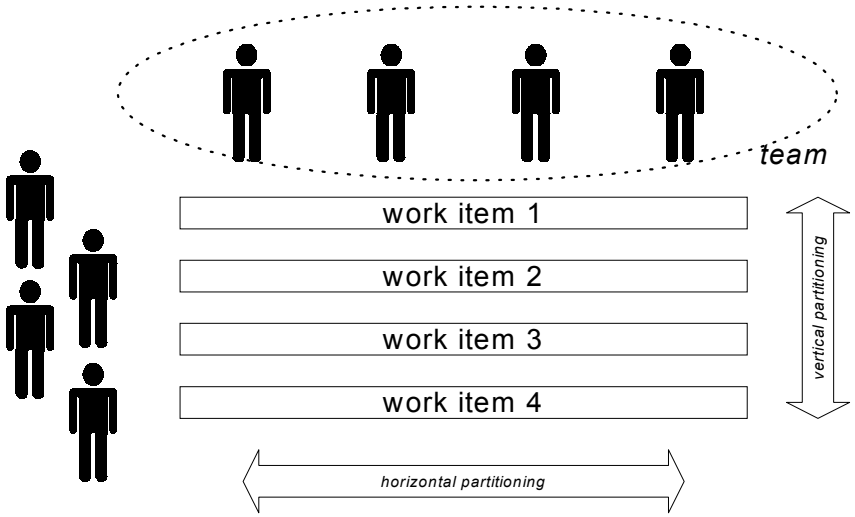


Figure 3: Horizontal partitioning versus vertical partitioning.

To illustrate the essence of teamwork, as opposed to ordinary work distribution supported by the current generation of workflow products, consider Figure 3. Today's products support only a vertical partitioning of work, i.e., work items are distributed over resources and, eventually, every work item is executed by one resource. As Figure 3 shows there is another dimension when it comes to the distribution of work. For a horizontal partitioning of work, multiple resources are involved in an activity, i.e., the execution of a single work item. For example, the members of the selection committee execute the work item "Select new dean of the Computing Science Faculty."

To decouple the workflow process definition from concrete resources, the concept of *role* was introduced as explained in the previous section. For teams, we use a similar concept and distinguish between *teams* and *team types*. Within a team there can be several *positions* (such as manager, director, VP, etc.) and team members produce concrete *contributions* which lead to the completion of a task. To be able to model activities executed by teams, we introduce teams, team types, team positions, and contributions.

- A *team* is a group of resources. A team can have several members and one person can be a member of many teams. Some teams are created on-the-fly, i.e., the team is created the

moment an activity requires a team of a specific type. Other teams are of a more permanent nature and handle many activities.

- A *team type* does not refer to specific resources but can be seen as the role concept extended to teams. A team type refers to a structure which corresponds to a group of resources having certain properties with respect to the composition of the team in terms of sizes and roles of its members.
- A *team position* is a specified role within a team. For example, consider a policy that "the *chair* of the selection committee should be a full professor, while other *members* should be full-time faculty of any rank." In this example, the chair and member have different roles within the team.
- *Contributions* are produced by resources within the context of an activity and link team positions to concrete resources. Without such a notion, the relationship between resources within a team and team positions is undefined.

A *role* can be considered as a special team type consisting of only one team position.

3.2. Overview of Organizational meta model extended with teams

Figure 4 shows the organizational meta model extended with teams. (See Appendix 1 for a larger diagram.) Four new object classes have been added: *team*, *team type*, *team position*, and *contribution*. For now we focus on the first three of these object classes. The object class *contribution* is discussed in the second half of this section.

Since a role can be considered as a special team type with just one team position, the association between *task* and *role* is replaced by an association between *task* and *team_type*. The association *requires* links each task to a team type. Similarly, the association between *activity* and *resource* is replaced by an association between *activity* and *team*. Association *uses* links each activity to a team. Every activity corresponds to one team, i.e., the team executing the corresponding activity. Note that a *team* is an instance of *team_type*. Objects of class *team_position* relate team types and roles. The association *instance_of* relates each team to its corresponding team type. The association *instance_of* shows that a team is more than just a group of resources: A team instantiates a team type and the same set of resources can correspond to several teams. The new association *isa* which relates team types is similar to the generalization which relates roles. If team type A is a subclass of team type B, then teams of type A can execute tasks mapped onto team type B.

Figure 4 shows several attributes. Again, these attributes are representative rather than exhaustive. For instance, note that the class *team_position* has an attribute *cardinality*. This attribute specifies the number of resources in a given position within a team.

Also note that now there is no association between *task* and *role*. Therefore, for tasks requiring one resource, having a specific role, a singleton team type is introduced, i.e., a team type with one position of cardinality 1. Therefore, a role can be considered as a special team type consisting of only one team position. This choice is made to simplify the meta model. An alternative is to add another association *requires* with association ends *role3* and *task4*, which link class *task* with task *role*. In this case the cardinality of association end *role3* is 0..1 instead of 1..1 because only tasks requiring a single resource are directly linked to a role. Moreover, the cardinality of association end *tt3* is 0..1 instead of 1..1 because tasks requiring a single resource are not linked to a team type.

In the next section, we show how constraints are modeled for this meta model in OCL.

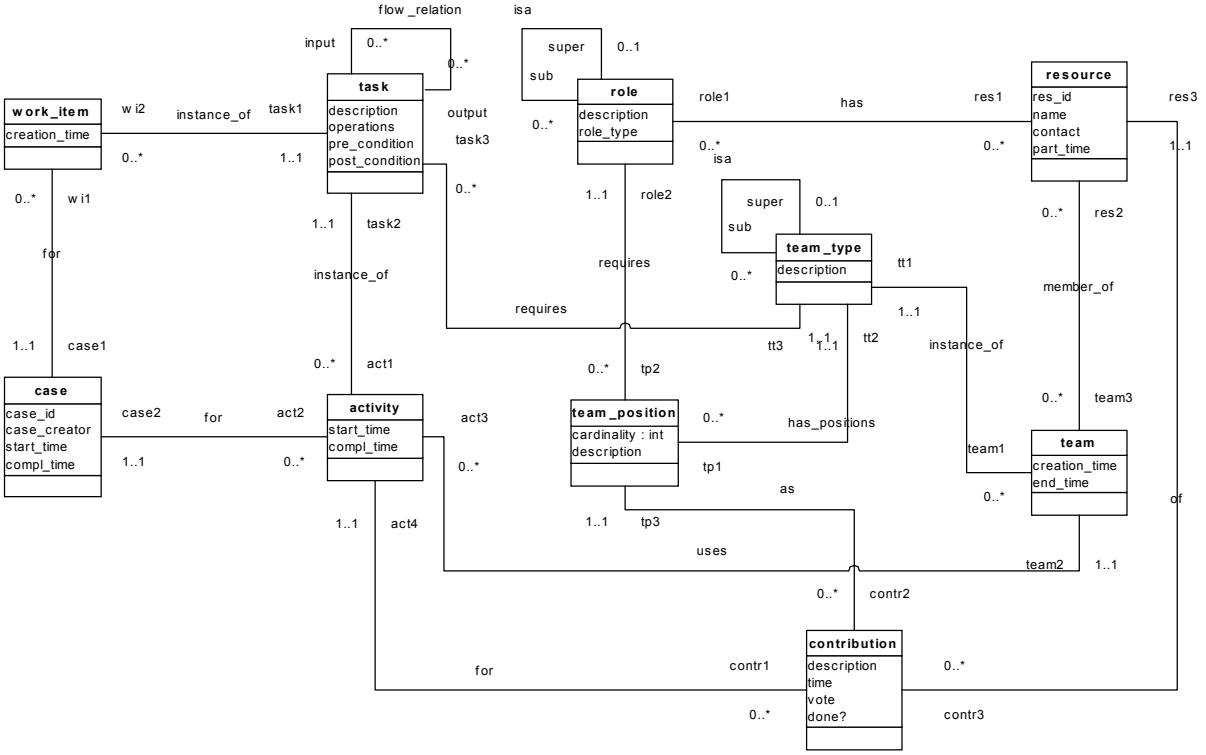


Figure 4: The organizational meta model extended with teams (also see Appendix).

3.3. Modeling constraints for the team-enabled model in OCL

Next, we use OCL to express various constraints related to the team-enabled meta model. Consider a constraint to ensure that the team executing the activity is of the proper type. This constraint is expressed as follows:

activity
(T.1) self.task2.tt3.team1–>includes(self.team2)

Here *self.team2* is the team executing the activity *self*, while *self.task2.tt3* is the team type of the task associated with activity *self*. Clearly, the team executing the activity should be an instance of this type.

Each task is mapped onto one team type and each team type has a fixed number of team positions. Sometimes it is useful to have alternative team configurations. Consider, for example, alternative selection committees consisting of (1) a full professor and two associate professors, (2) two full professors and one assistant professor, or (3) a full professor and three assistant professors. We use the association *isa* to allow for alternative team configurations. Let the context of *self* be *team_type*. *self.sub* is a set of team types: These team types are subclasses of the team type *self* and correspond to alternative configurations. A task which requires *self* can also be executed by teams of one of the types in *self.sub*. The generalization allows for tree-like structures. Therefore, it is possible to represent sets and lists of possible team types.

Above, we did not allow inheritance. Now, we consider a limited form of inheritance of roles in the next constraint:

activity
(T.2) self.task2.tt3.team1–>includes(self.team2) or
self.task2.tt3.sub.team1–> includes(self.team2)

This constraint is a two-part boolean expression which allows for only one level of inheritance, i.e., only the direct subclasses can act as an allowed team configuration. For full inheritance, we need to take the transitive closure of the *isa* relation. As discussed previously [25], we resort to the notation *sub** to refer to the transitive closure. This is shown below.

activity
(T.3) self.task2.tt3.sub*.team1–>includes(self.team2)

Team types have a fixed number of team positions. A team position specifies a role within the context of a team. Sometimes, one needs multiple team members having the same role (for example, *two* managers may be required on a team). Therefore, the object class *team position* has the attribute *cardinality*. This attribute specifies the number of resources having a specified role. Clearly, the number of actual team members of each type should match the specified number of team members of a given type. In fact, also the total number of team members should match the number specified. Therefore, we add the following OCL constraints:

team
(T.4) self.res2–> size = self.tt1.tp1.cardinality–>sum
(T.5) self.tt1.tp1 –> forall(x | x.cardinality <=
x.role2.res1 –> intersection(self.res2)–>size)

(T.4) Specifies that the actual number of team members *(self.res2–>size)* matches the specified number *(self.tt1.tp1.cardinality–>sum).* (T.5) is more involved. *self.tt1.tp1* is the set of all specified team positions within a given team, and *x* represents one of these team positions. *x.cardinality* is the required number of resources having role *x.role2*. *x.role2.res1* is the set of resources having the required role. *x.role2.res1 –> intersection(self.res2)–>size* is the number of resources in team *self* having the required role. Since one resource can have multiple roles, *x.cardinality* does not need to be equal to *x.role2.res1 –> intersection(self.res2)–>size*, i.e., there can be more resources with the required role. (T.5) does not take role inheritance into account. To allow for full role inheritance, we could replace (T.5) by:

team
(T.6) self.tt1.tp1 –> forall(x | x.cardinality <=
x.role2.sub*.res1 –> intersection(self.res2)–>size)

The inequality in this constraint (<=) still leaves the constraint weak in the sense that it may not model the desired team accurately. Consider a team type with two team positions: one requiring a resource performing role A and one requiring a resource performing role B. Moreover, suppose there are two resources: one resource having both roles and another having neither of these roles. A team consisting of these two resources does satisfy the above OCL constraints. Nevertheless, it is clear that this team is not a proper instance of the team type. The constraints given are weak since they do not take the specific contribution of a team

member into account. Therefore, to express the desired constraint in OCL more precisely, we introduce the object class *contribution*, shown in Figure 4.

Contributions link team positions to concrete resources. Without this class, the relation between resources within a team and team positions is undefined. Consider for example a team type with two positions requiring different roles, A and B, and a team consisting of two workers, X and Y, each having both roles. Without the class *contribution*, it is not clear whether X has position A or B within the team context.

An object of the class *contribution* corresponds to one activity (association end *act4*), one resource (association end *res3*), and one team position (association end *tp3*). Figure 4 shows some attributes for objects of the class *contribution*. These attributes can be used to reflect the status and outcome of a contribution, e.g., *done?* indicates whether, from the viewpoint of the contributor, the activity is finished, *vote* indicates the vote of the contributor, *time* indicates the time of completion. Note that these attributes are just examples.

The presence of the class *contribution* allows for a more precise specification of the constraints mentioned before.

activity
(T.7) self.team2.res2=self.contr1.res3
(T.8) self.team2.tt1.tp1=self.contr1.tp3

(T.7) specifies that the set of team members *(self.team2.res2)* should match the set of contributors *(self.contr1.res3).* (T.8) states that the set of team positions taken by the contributors should match the set of team positions in the corresponding team. The latter constraint does not take the cardinality of team positions into account. Moreover, none of the above constraints guarantees that team members have the required role. Therefore, we need two additional OCL constraints as follows:

contribution
(T.9) self.tp3.cardinality=self.act4.contr1–>select(x|x.tp3=self.tp3)–>size
(T.10) self.tp3.role2.res1–>includes(self.res3)

(T.9) specifies that the specified number of team members holding a position *(self.tp3.cardinality)* should match the actual number of contributors linked to this position *(self.act4.contr1–>select(x|x.tp3=self.tp3)–>size).* Note that the context of *self* is a contribution and *self.act4.contr1* is the set of contributions within the context of one activity. (T.10) guarantees that team members have the required role: *self.res3* is the resource taking care of the contribution *self*, and *self.tp3.role2.res1* is the set of resources having the role required for the corresponding team position. Clearly, *self.res3* should be included in *self.tp3.role2.res1*. The second constraint does not take role inheritance into account. Both limited and full inheritance can be taken into account, e.g., (T.10) should be replaced by *self.tp3.role2.sub*.res1–>includes(self.res3)* to allow for full inheritance.

The meta model of Figure 4 is a key contribution of this paper. We call this model the *Team-Enabled Workflow Reference Model*. Using OCL, we have specified all reasonable constraints. To add other meaningful constraints requires the explicit formulation of attributes, e.g., assuming attributes *start_time* and *compl_time* for both class *case* and class *activity*, as shown in Figure 4, we can add the following constraint:

activity
(T.11) self.case2.start_time <= self.start_time and
self.case2.compl_time >= self.compl_time

This constraint specifies that all activities should be executed during the lifetime of a case. We could have added many such constraints. However, we decided to focus on the more fundamental and structural requirements.

3.4. Specific OCL constraints

All constraints involving teams described thus far are at the level of the meta model. In this subsection, we show some examples of constraints at the organizational level. These examples illustrate the potential of OCL for specifying such constraints.

case
(T.12) self.completed implies self.act2.team2.res2–>size > 10
(T.13) self.act2.contr1.res3.role1–>select(x | x.description='manager')–>notEmpty

(T.12) specifies that more than 10 people need to have been involved in the execution of an activity. This constraint assumes that class *case* has an attribute *completed*. (T.13) specifies that someone with the role "manager" should execute at least one of the steps in the process. This constraint uses an attribute *description* in class *role*. The two constraints are at the organizational level. Similarly, we could also have added constraints that are specific to a given process (see Section 2.3.2).

4. Team Allocation Mechanisms

As noted earlier, previous research on workflow has only considered assignment of work to individuals, neglecting several important issues we shall consider here in this context. In this section, we discuss how work is offered to teams and confirmation of completion received from them. We also discuss various attributes required to model team behavior, and illustrate them in the context of a complete example.

4.1. Offering work to a team as opposed to an individual

Normally workflow systems employ two mechanisms to offer work to individuals: the *push* and *pull* approaches. In the push approach the workflow system assigns work to a specific worker, while in the pull approach the work item is offered to multiple workers and, after one of them accepts it, it is withdrawn from the others. The pull approach is more common and desirable because it prevents a work item from getting blocked in the queue of a worker who may not be available. Moreover, the push approach can be seen as a special case of the pull approach where a work item is offered to just one worker. Consider for example the work queues in Staffware. From a conceptual and technical point of view, the group queue and the private work queue are identical. In this subsection we shall therefore consider how the pull approach would work in the context of teams.

A workflow system can offer work to a team by sending a notification to all eligible workers who can fill the positions of various members of the team. For instance if a team must have 2 full professors and 1 associate professor, then the workflow system can offer the work item to

all the full professors and all the associate professors. As soon as one associate professor has accepted the task, it would be withdrawn from all the other associate professors. Similarly, once two full professors have accepted the task, it would be withdrawn from the other full professors also.

The drawback with this simple approach is that it could result in teams consisting of incompatible individuals. This is an important aspect of social organizations. Therefore, another alternative is that the workflow system would offer the work item to multiple teams, where each individual would know who the other team members are. In accepting to serve on a team, an individual would be doing so on the condition that the other named individuals accept as well. Upon receiving acceptances from all potential members of a specific candidate team, the workflow system would assign the work item to that team. Moreover, if any acceptances had been received from other potential members whose team did not get selected, they would be notified accordingly. Lastly, the work item would be withdrawn from the queues of workers who failed to respond.

4.2. Receiving confirmation of work completion from a team

When an individual completes his or her task, they would press a button on their screen to notify the workflow system that a task has been done. The workflow system determines the next task for this instance and assigns it to a worker. In the case of teamwork, completion by the team can be notified to the workflow system in two ways. One, each team member would inform the system separately. As shown in Figure 4, the contribution entity has an attribute called "Done?" whose value is set to yes when a resource completes its own part of the work related to this team activity. Thus, when all the members of this team instance set this variable to "yes", the workflow system can treat the corresponding team activity as completed. On the other hand, an alternative is to assign one individual as a team coordinator and give him or her the responsibility for setting the value of this variable. Note that the latter corresponds to the way teams are supported in Lotus Domino Workflow. As noted earlier this is often undesirable.

4.3. Attributes or dimensions for modeling team behavior

In this subsection, we discuss various attributes associated with a team. Two important attributes that have been introduced in groupware systems are time and place, i.e., are the participants in a group interacting in the same time/different time dimension, and how are they physically located in the (same/different) place dimension (see [10,12,21]). These two dimensions are important for a team as well. In addition, it is also necessary to consider several other dimensions as follows.

4.3.1. Time/Place

One paradigm for organizing team activities is in terms of the time and place dimensions as shown below.

	Same Place	*Different Place*
Same Time	Meetings, Whiteboards	Videoconferencing, document sharing
Different Time	Mailbox, bulletin board	Electronic mail, workflow

We adopt this from literature on groupware [10,12,21] because it lends itself well in our environment also for modeling behavior of different teams. There are four boxes corresponding to the combinations of same/different time/place. Each box gives examples of technologies that belong in it. Also, note that the notion of space includes not just physical space, but also virtual space. Therefore, a bulletin board is considered as "same place," in a virtual sense.

4.3.2. Decision criterion/Quorum size

It is important to model the decision-making criterion or criteria, and communicate them to the team members. For example, a common criterion is voting, i.e., each member votes on the issue being discussed. If voting is used, it is necessary to inform the members whether the voting is anonymous or public, can a member see the other members' votes before casting their own, the window during which the votes can be cast, what kind of majority is required (50%, 67%, unanimous, etc.) for the issue to pass and the size of the quorum, i.e., what is the minimum number of votes that must be cast.

There are a variety of other non-voting based criteria also. For example, by discussion and recommendations, team interaction can bring a case to the next level without any explicit voting.

4.3.3. Team task duration (Time out/No time out)

A time out indicates a deadline by which a decision must be made by the team. This is a more common situation. The alternative is no stated time out, thus giving the team flexibility to complete its work. For example, occasionally, a team may collaborate on a writing project where the deadline is somewhat variable. Individual tasks have deadlines too, and if the deadline for a task expires, the individual who was supposed to perform the task is notified. However, in the case of team tasks, it is necessary to notify either the coordinator of the team (if one exists) or every member of the team in order to ensure that appropriate action is taken.

4.3.4. Team interaction style (Structured/ad hoc)

This indicates whether the meeting is formally scheduled (structured) or whether it is ad hoc (unstructured). In the context of the time/place dimension discussed above, a formally scheduled meeting would be classified as *same time, same place* or *same time, different place*. On the other hand, an unstructured interaction could take place by means of electronic mail (*different time, different place*) or even by an electronic discussion board *(same place, different time*). Other aspects of structure could relate to issues like whether transcripts of the meeting are stored and minutes are prepared. The team might also need additional tools to facilitate the meeting, e.g., specialized software for demonstrations, charts and graphics.

4.3.5. Rounds (single/multiple)

It would be useful to predetermine whether the team will operate in a single round or multiple rounds. Often, as the example below will illustrate, a team may issue a preliminary report and then invite comments from constituents before issuing a final report. In this case, the team would perform two rounds or even more if there is need for multiple iterations. In other situations a single meeting or round is considered to be final. Therefore, this is another useful aspect of team behavior from a modeling standpoint.

4.3.6. Team instances (single/multiple)

A team may create instances of itself or sub-teams. Often large teams tend to break up their work into sub-committees that perform various tasks and report to the team. The workflow system should be able to provide support for such situations so that activities of sub-teams can be coordinated with one another and with the entire team. This kind of sub-division of work is usually done on an ad hoc basis.

The next section will tie together these ideas through a detailed example.

4.4. An example

There are several activities that involve considerable team operation. A conventional workflow system would not be very effective in such situations. To illustrate, we consider the workflow of how a tenure case is reviewed at a typical state university in America. The various steps are as follows:

1. The department head appoints a fact-finding team (sub-committee) of three professors.
2. This task force or team reviews all the materials of the candidate, invites letters of reference from external professors and past students, and based on a review of all the documents, prepares a report. The task force also votes.
3. This report is shared with all the other tenured members of the department (including the department chair) at least one week prior to a date when they all must meet.
4. A meeting of the tenured department members is held to discuss the case.
5. Afterwards, they cast their vote on the case in a secret ballot. A vote of 67% or more is considered positive; otherwise, it is negative.
6. The department chair prepares his/her report independently without knowledge of the department vote.
7. The next stage is the dean's advisory committee (DAC), another team. This team consists of four individuals who are appointees of the dean. They must meet with the dean to discuss the candidate and give their vote to the dean. Here a vote greater than 50% is considered positive.
8. The matter is reviewed by the college dean. If the DAC disagrees with the department vote, the dean may at his/her discretion send the case back to the department for reconsideration.
9. The dean writes his/her report and recommendation, and sends it to the vice-chancellor.
10. In the vice-chancellor's office, it is first reviewed by the vice-chancellor's advisory committee (called VCAC - yet another team) that consists of eleven members.
11. The VCAC discusses the case and gives a recommendation to the vice-chancellor along with a vote.
12. The vice-chancellor then makes a decision and sends it to the Chancellor.
13. There are three more levels that we will abbreviate: the Chancellor, the President and the Regents (the last team!).

In addition there are various important constraints that apply to this workflow.

1. No individual may cast his/her vote at more than one time. The only exception is the task force. If a member of the task force casts a vote in the task force and then once again, this is allowed.

2. No member may participate in the discussions at more than two levels of review.
3. At any level, the recommendation of the previous level may be overturned.
4. If a certain level disagrees with the recommendation of the previous level, it may at its discretion send the matter back to the previous level for reconsideration.
5. There must be one tenured, full professor in the fact-finding committee. Similar constraints related to committee composition apply at all levels.
6. The workflow must reach the office of the vice-chancellor by January 31 of the academic year.
7. The candidate must be notified of the final decision latest by May 15 of the academic year, i.e., the workflow must finish by this deadline.
8. At each stage there is a long checklist of tasks that must be completed before the workflow can proceed.

Task	**Team or Individual**	**Time/Place**	**Decision Type**	**Deadline**	**Rounds**
Appointment of Task force	Individual	Not applicable (N/A)	Individual	Not fixed	1
Task force report	Fixed-size Team	Diff./diff.	Reporting of anonymous votes	Not fixed	1
Department Meeting	Variable-size Team	Same/same	Discussion	Not fixed	1 or more
Department vote	Variable-size Team	Diff/diff.	67% majority, secret ballot	Not fixed	1 or more
Chairman report	Individual	N/A	Individual	Not fixed	1 or more
DAC meeting and recommendation	Fixed-size Team	Same/same	50% majority, partly secret	Not fixed	1 or more
Dean Report	Individual	N/A	Individual	Jan 31	1 or more
VCAC meeting & recommendation	Fixed-size Team	Same/same	50% majority, partly secret	Not fixed	1 or more
VC decision	Individual	N/A	Individual	Not fixed	1
Chancellor decision	Individual	N/A	Individual	Not fixed	1
President decision	Individual	N/A	Individual	Not fixed	1
Regents decision	Fixed-size Team	Same/same	50% majority, public ballot	May 15	1

Table 1: Features of various steps in the tenure evaluation workflow process

It is evident that this important application requires *considerable* teamwork from start to end because there are several team-based activities along the way. At each stage there are teams of fixed and variable sizes involved along with constraints on how the teams should be formed. Moreover, the decision-making criteria are also different. Table 1 summarizes this workflow, showing the various steps in sequence, and gives the features of each task. For

brevity two of the aspects mentioned in Section 4.3 (team task duration and team interaction style) were omitted from Table 1.

4.5. Mapping Teams into our Framework

In this section we discuss how the tenure review example may be modeled using the terminology introduced in Section 3. Table 2 shows the various teams that are required in processing this workflow. Each row of this table shows the *team_type, team_position, role qualification, team, contribution*, which are various entities shown in Figure 4 and discussed in Section 3. Moreover, each *contribution* entity is linked to three entities, i.e., *activity, team position* and *resource*. These are also shown in Table 2 along with two sample attributes of Contribution, "Done?" and "Vote", which denote whether a resource is done with his/her contribution to the team task, and also what his or her vote was. Thus, Table 2 represents the case or a workflow instance of "John Doe's tenure review."

As an example, the first team shown in Table 2 is the Special Task Force. This team has three positions (one full professor and two professors of any rank). The instance of this team is the task force for Dick Jones Tenure case. Further position 1 on this team is filled by the resource Jim A. Similarly, the other two positions are also filled by suitable resources. The value for Jim A. for the "Done?" attribute is 'yes' and for the "Vote" attribute is also 'yes.'

Team_type	**Team position/ Role/ Qualification**	**Team (instance)**	**Contribution**				
			Activity	**Team Position**	**Resource**	**Attribute: Done?**	**Attribute: Vote**
Special Task force	1. Chair 2. Any Prof. 3. Any Prof. (all tenured)	Task force for John Doe's Tenure case	Task force report and vote	1. 2. 3.	Jim A. Dawn B. Jill M.	Yes Yes No	Yes No None
Department Meeting	Prof 1 Prof 2 ... (All tenured)	Department team for John Doe's tenure case	Dept. Meeting and vote	1. 2. 3.	Mary C. Diane M. John. D.	No No No	None None None .....
DAC meeting and recommend-ation	1. Full Prof 2. Full Prof 3. Full Prof. 4. Full Prof.	Members of Dean's Committee	DAC meeting and vote	1. 2. 3.	Don L. Jo M. Chin D.	No No No	None None None .....
VCAC meeting & recommend-ation	11 Professors from different colleges	Members of VCAC Committee	VCAC meeting and vote	1. 2. 3.	Sam P. Mel C. Su D.	No No No	None None None .....
Board of Regents	7 Elected Regents	Members of Board of Regents	Regents vote	1. 2. 3.	Jane C. Jan M. Mike D.	No No No	None None None .....

Table 2: Work case for John Doe's tenure review

Lack of support for teamwork is a major reason that it takes such a long time for various workflows such as the one above to complete. We foresee that a team-enabled workflow system can provide support by determining the appropriate members of a team and offer work to them, managing work completion notifications by a team, verifying prerequisites, enforcing

constraints, arranging team meetings, managing team documents, and provide support for team activities. The last two items are important activities in themselves and there is already an existing body of work in the area of groupware. Therefore, we devote the next section to discuss our ideas on how groupware support can be integrated with a workflow system.

5. Team-enabled workflow management systems and the link with existing groupware products

Previous sections have motivated the need for effective solutions to the problem of supporting team-based workflow systems. We foresee a major need for a "marriage" between team enabled workflow systems and groupware systems in order to realize the next generation of workflow systems. This section will describe our vision for achieving such integration. The area of groupware encompasses a variety of systems that fall in the general area Computer Supported Cooperative Work (CSCW), a term coined by Irene Greif and Paul Cashman in a pioneering 1984 Workshop. A brief overview of groupware systems follows first.

There is a large variety of groupware systems (see [21] for an introduction). The simplest example of a groupware system is an electronic mail system or a bulletin board system such as USENET. More sophisticated message-based systems are Lotus Notes, Novell Groupwise and Microsoft Exchange. Another category of groupware systems is audio conferencing systems based on MBONE [14] and Sun ShowMe from Sun, and video conferencing systems like CU-SeeMe from Cornell University. There has also been considerable work in developing groupware systems for electronic meetings to support brainstorming and decision-making. These systems are often called Group Decision Support Systems (GDSS). Some pioneering work in this area was done at the University of Arizona in the 1980s [9]. Another well-known electronic meeting system is Object Lens from MIT [22]. Subsequent examples of newer electronic meeting and decision support systems are Microsoft NetMeeting, Colab from Xerox, etc. There are also toolkits for groupware development such as COAST [35] and GroupKit [17] from the University of Calgary. See [38] for a more detailed and recent survey. For some not-so-recent, but interesting, surveys and historical perspective, see [12,19]. The systems where all participants interact in real-time are called synchronous groupware systems and performance considerations (such as fast response time) are especially important here. On the other hand, e-mail and messaging systems are examples of asynchronous groupware.

An interesting classification of collaborative technologies is given in [13]. There Ellis presents a taxonomy dividing collaborative technologies into four classes of functionality:

- *Keepers* support the access and change to shared artifacts. Typical issues that are of primary concern to keepers are access control, versioning, backup, recovery, and concurrency control. Examples of keepers include the vault in a Product Data Management (PDM) system, a repository with drawings in a CAD/CAM system, and a multi media database system.
- *Coordinators* are concerned with the ordering and synchronization of individual activities that make up the whole process. Typical issues addressed by coordinators are process design, process enactment, enabling of activities, and progress monitoring. The key functionality of a workflow management system is playing the role of coordinator.
- *Communicators* are concerned with explicit communication between participants in collaborative endeavors. Typical examples are electronic mail systems and video

conferencing systems, and basic issues that need to be addressed are message passing (broadcast, multicast, etc.), communication protocols, and conversation management.

- *Team-agents* are specialized domain-specific pieces of functionality. A team agent is typically a system acting on behalf of a specific person or group and executing a specific task. Examples include an electronic agenda and a meeting scheduler.

This classification assists in developing an approach for marrying team enabled workflow systems with groupware systems. The functionality of workflow management systems is usually limited to the coordinator role. On the other hand, groupware systems (i.e., excluding workflow technology) tend to be weak on the coordination dimension, and stronger on the keeper, communicator, and team-agent functions. Many groupware systems provide various kinds of support for group decision-making, but they do not have any notions of workflow. Lotus Notes is an exception, which does provide all four functions, but it is not a full-fledged workflow system. For instance, a team does not know what action the previous team in the workflow took. Therefore, it is important to link workflow and groupware technologies more closely. We see groupware support as an add-on or plug in, whereby the workflow system can determine the members of a meeting, along with time and place issues and hand over all the relevant information for the meeting to the groupware system. The groupware system will then arrange the meeting, record its decisions/recommendations, also keep text, audio, video transcripts of the meeting and then send this information back to the workflow system. The workflow system will then determine the next step for the specific workflow instance and move the instance forward.

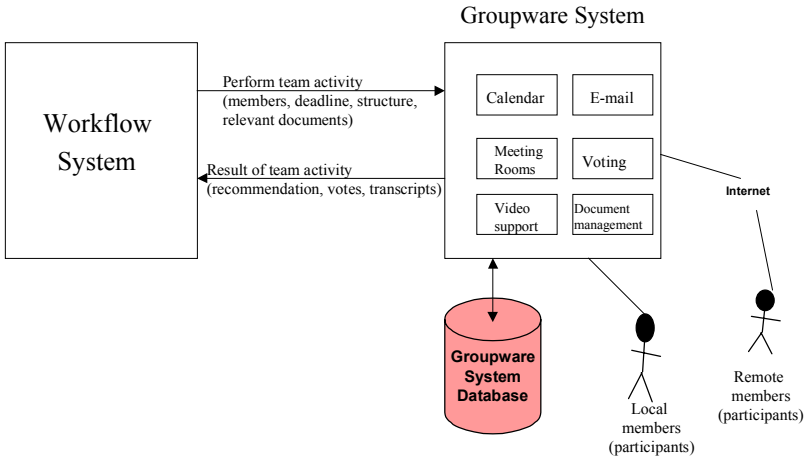


Figure 5: Architecture for integrating groupware support into a Workflow system

Figure 5 proposes an architecture for integrating groupware support into a workflow system for performing team activities. A groupware system has various standard modules such as a calendar for scheduling, e-mail support, and video support. In addition it can keep track of availability of meeting rooms. Some groupware systems also have modules for voting support in order to determine the votes of the participants.

When a team activity is to be performed, the workflow system will make an asynchronous call on the groupware system and provide details such as:

- Names of specific team members to be included
- Deadline for the completion of this meeting
- Any documents the team may need to perform its work
- Structure of the meeting (same/different time/place, video/audio/email)

- Quorum size (minimum number of participants)

After initiating the team activity, the workflow system may proceed with other tasks. Meanwhile, the groupware system will arrange the appropriate kind of meeting based on the schedules of the individuals, availability of meeting rooms, availability of video equipment, etc. In addition, the groupware system will keep a log and transcripts of all meetings in multi-media form (e.g., as video/audio transcripts and email text). It will also allow participants to cast their votes and store the results. At the end of this activity, a person (perhaps a coordinator, or chairperson) will inform the groupware support system that the activity is finished. At this time, the system will notify the workflow system that the activity has been completed and also return various information, such as all the transcripts, any new documents prepared by the team, the results of any votes that were taken, etc.

As shown in the architecture, all relevant information such as documents, logs, transcripts, results of votes, etc. would be stored in the database of the groupware system. Figure 5 also shows that the members of a team may either be directly connected to the groupware system or they may access the groupware system remotely over the Internet or even through another network.

Although the various modules shown in Figure 5 exist independently, the integration is often done manually, and not through a common interface. We, therefore, anticipate a two-phase approach to such realization. In the first phase, the workflow system would send all the information to a human coordinator. This coordinator would invoke various modules of the groupware system to perform various tasks and, at completion, send all the information back to the workflow system. This coordinator would also be responsible for ensuring that the activity is completed within the specified deadline.

In time, however, the interface would be automated so that the workflow system may make asynchronous API calls on the groupware system. These calls would allow the workflow system to initiate a team activity as shown in Figure 5. Additional calls would check status of the activity, modify an activity, cancel an activity, change membership, etc. Although a detailed implementation is beyond the scope of this paper, we anticipate that a toolkit kind of approach, such as Groupkit [17], would lend itself well for implementing our architecture by combining various modules into a complete solution.

6. Conclusion

This paper started with the observation that team support is missing in contemporary workflow management systems. Yet, there are numerous, real-world applications where tasks are performed by teams. Therefore, effective support for team activities in workflow systems is essential. Secondly, there is a great need for integrating groupware support within a workflow system. Unfortunately, there is hardly any literature on the "marriage" between team-enabled systems developed in the CSCW domain and workflow-enabled systems developed by workflow vendors. Although there is a working group on resource modeling (WG9), the Workflow Management Coalition (WfMC) did not consider activities executed by teams. Both researchers and software developers in the CSCW domain have developed a wide range of group support systems. These systems are team-enabled but do not explicitly model the business processes and organizational structures. Therefore, these systems are unaware of the workflow processes at hand and do not support the enactment of these processes. It should be noted that systems like Lotus Domino Workflow (Lotus/IBM, [28]) allow for teams. However, these systems use a so-called activity owner as a mediator between the workflow

management system and the team members. The team member can share information regarding the activity being executed. However, team positions within teams, requirements on the size and structure of the team, and termination conditions involving voting are not taken into account. The activity owner simply reports the completing of the task.

In this paper, we first proposed a *reference model* for making workflow management systems team-enabled and then developed an architecture for integrating workflow and groupware technologies. The reference model has been expressed in terms of a UML class diagram, which we named the *Team-enabled Workflow Reference Model* (Figure 4). This reference model is based on a core model involving standard concepts such as case, task, work item, activity, role, qualification, position, competence and resource, and has been further extended with *team-specific concepts* such as team, team type, team position and contribution. One of the interesting features of this model is the presence of detailed OCL constraints. OCL allows for the specification of generic (i.e., at the meta level) and specific (i.e., at the organizational/process level) constraints. We illustrated how this reference model can be applied to the case of the tenure evaluation process, an example application that requires considerable amount of team activity, and one that many of our readers can relate to.
Finally, an architecture for integrating groupware support into a workflow system by means of an API was discussed. In the Eindhoven Digital Laboratory for Business Processes [11], we currently use the workflow management system Staffware [37] and the group support system GroupSystems [18]. Both systems are used in several courses given at Eindhoven University of Technology and have been used to build research prototypes. To demonstrate the conceptual ideas in this paper, we plan to integrate Staffware and GroupSystems using the team-enabled workflow reference model and the architecture presented in this paper.

Acknowledgments
Some part of the work on this paper was done while the second author was visiting Eindhoven University of Technology, The Netherlands. He gratefully acknowledges support and hospitality of the research institute BETA and the Faculty of Technology Management, Eindhoven University of Technology. We would also like to thank the three reviewers for their comments that helped to improve this paper.

7. References

1. W.M.P. van der Aalst. The Application of Petri Nets to Workflow Management. The Journal of Circuits, Systems and Computers, 8(1):21-66, 1998.
2. W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Advanced Workflow Patterns. In O. Etzion and P. Scheuermann, editors, 7th International Conference on Cooperative Information Systems (CoopIS 2000), volume 1901 of Lecture Notes in Computer Science, pages 18-29. Springer-Verlag, Berlin, 2000.
3. V. Atluri and Wei-kuang Huang. An Extended Petri Net Model for Supporting Workflows in a Multilevel Secure Environment. DBSec 1996: 240-258.
4. V. Atluri, Wei-kuang Huang, and E. Bertino. An Execution Model for Multilevel Secure Workflows. DBSec 1997: 151-165
5. E. Bertino, Elena Ferrari, and V. Atluri. The Specification and Enforcement of Authorization Constraints in Workflow Management Systems. TISSEC 2(1): 65-104, 1999.

6. C. Bussler and S. Jablonski. Policy Resolution for Workflow management. In Proceedings 28th Hawaii International Conference on System Sciences (HICSS'95) Conference, January 1995.
7. S. Castano and M. Fugini. Rules and Patterns for Security in Workflow Systems. DBSec. 59-74, 1998.
8. Cosa Corporation, http://www.cosa.de.
9. A. Dennis, et. al.. Information Technology to Support electronic meetings, MIS Quarterly, 12, 4 (December 1988), 591-619.
10. G. DeSanctis and B. Gallupe. A foundation for the study of group decision support systems. Management Science, Vol. 33, No. 5, 1987, 589-609.
11. EDL-BP. Eindhoven Digital Laboratory for Business Processes. http://tmitwww.tm.tue.nl/research/edlbp
12. C. Ellis, et. al.. Groupware: Some issues and experiences, Communications of ACM, Vol. 34, No. 1, January 1991, 38-58.
13. C. Ellis. An Evaluation Framework for Collaborative Systems. Technical report CU-CS-901-00, Computer Science Department, University of Colorado, Boulder, 2000.
14. H. Eriksson. MBONE: The Multicast Backbone, Communications of ACM, Vol. 37, No. 8, August 1994.
15. D. F. Ferraiolo, J. Cugini and D.R. Kuhn: Role-Based Access Control: Features and Motivation. In Annual Computer Security Applications Conference, IEEE Computer Society Press, 1995.
16. M. Fowler and K. Scott. UML Distilled: Applying the Standard Object Modeling Language. Addison-Wesley, New York, 1997.
17. Group Kit 1998, GroupKit 5.0 Documentation, University of Calgary, http://www.cpsc.ucalgary.ca/projects/grouplab/groupkit/.
18. GroupSystems. GroupSystems.com/ Ventana Corporation. http://www.groupsystems.com
19. J. Grudin, "Computer-supported cooperative work: history and focus," IEEE Computer, Vol. 27, No. 4, May 1994.
20. IBM MQ Series workflow, http://www-4.ibm.com/software/ts/mqseries/workflow/.
21. S. Khosafian and M. Buckiewicz. Introduction to Groupware, Workflow and Workgroup Computing, Wiley, 1995.
22. L. Kum-Yew, et, al. Object Lens: A spreadsheet for cooperative work, ACM Transactions on Office Information Systems, Vol. 6, No. 4, October 1988.
23. S. Jablonski and C. Bussler. Workflow Management: Modeling Concepts, Architecture, and Implementation. International Thomson Computer Press, 1996.
24. P. Lawrence, editor. Workflow Handbook 1997, Workflow Management Coalition. John Wiley and Sons, New York, 1997.
25. L. Mandel and M.V.Cengarle. On the Expressive Power of OCL. In World Congress on Formal Methods, 854-874, 1999.
26. M. zur Mühlen. Resource Modeling in Workflow Applications. In: Becker, Zur Mühlen, and Rosemann: Workflow Management 99. Proceedings of the 1999 Workflow Management Conference. Muenster, Germany, November 9th 1999, Münster, 137-153, 1999.

27.M. zur Mühlen. Evaluation of Workflow Management Systems Using Meta Models. In R. Sprague, Proceedings of the 32nd Hawaii International Conference on System Sciences (HICSS'99). 1999.

28.S.P. Nielsen, C. Easthope, P. Gosselink, K. Gutsze and J. Roele. Using Lotus Domino Workflow 2.0. Redbook SG24-5963-00. IBM, Poughkeepsie, 2000.

29.M. Nyanchama and S.L. Osborn. The Role Graph Model and Conflict of Interest. ACM Transaction on Information and System Security (1): 3-33, 1999.

30.Rational Software. Object Constraint Language Specification, Version 1.1, Sept. 1997.

31.M. Richters and M. Gogolla. On formalizing the UML object constraint language OCL. In T.W. Ling, S. Ram, and M.L. Lee, editors, Proc. 17th Int. Conf. Conceptual Modeling (ER'98). Springer, Berlin, LNCS 1507, 1998.

32.J. Rumbaugh, I. Jacobson, and G. Booch. The Unified Modeling Language Reference Manual. Addison-Wesley, 1998.

33.R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-Based Access Control Models. IEEE Computer 29(2): 38-47, 1996.

34.R.S. Sandhu, V. Bhamidipati, and Q. Manuawer. The ARBAC97 Model for Role-Based Administration of Roles. ACM Transactions on Information and System Security, 2(1):105-135.

35.C. Schuckmann, et. al., Designing object-oriented synchronous groupware with COAST, Proceedings of the ACM 1996 Conference on Computer Supported Cooperative Work, page 30-38, November 1996, Boston.

36.R. Simon and M.E. Zurko, Separation of Duty in Role-based Environments. Proceedings of the 10th Computer Security Foundations Workshop (CSFW '97) 183-194.

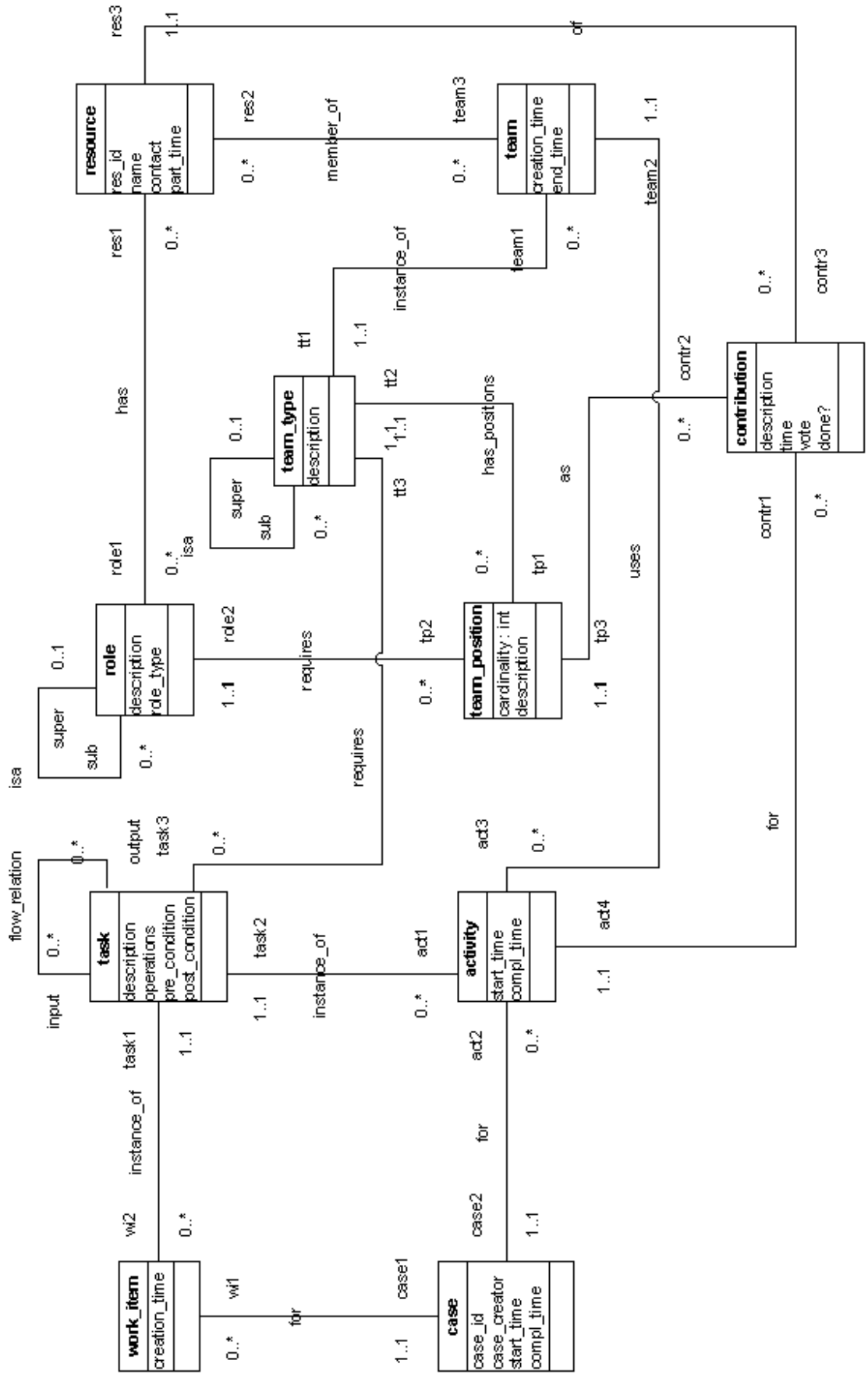
37.Staffware. Staffware PLC. http://www.staffware.com

38.S.Terzis, et. al., "The future of enterprise groupware applications," In Enterprise Information system (J. Filipe, editor), Kluwer Academic Publishers, 1999.

39.J.B. Warmer and A.B. Kleppe. The Object Constraint Language: Precise Modeling with UML. Addison-Wesley, 1999.

40.S. Wu, A. Sheth, and J. Miller: Task and Role Combined Access Control Model for Workflow Systems, University of Georgia, Technical Report, 2000.

Appendix: Meta model



Bio sketches

Wil M.P. van der Aalst is a full professor of Information Systems and head of the Department of Information and Technology of the Faculty of Technology Management of Eindhoven University of Technology. He is also a part-time full professor at the Computing Science department of the same university and has been working as a part-time consultant for Bakkenist for several years. His research interests are information systems, simulation, Petri nets, process models, workflow management systems, verification techniques, enterprise resource planning systems, computer supported cooperative work, and interorganizational business processes.

Akhil Kumar is currently a visiting researcher at Bell Labs, Murray Hill, NJ, on leave from University of Colorado, Boulder, where he is on the faculty of the College of Business. He holds a Ph.D. in Information Systems from University of California, Berkeley. In the past, he has served on the faculty at Cornell University and also worked in industry. He has published nearly fifty scholarly papers in top journals and leading international conferences in the database, and distributed and intelligent information systems areas. His current research efforts are focused in workflow systems and electronic commerce.