Managing Inter-Organizational Workflows With TEAM

Luiz Antônio M. Pereira
Banco Central do Brasil
Av. Presidente Vargas, 730 - 14th floor
Rio de Janeiro - Brazil - 20071-001
luiz.pereira@bcb.gov.br

Rubens Nascimento Melo
Pontifícia Universidade Católica do Rio de Janeiro
PUC-Rio. R. Marquês de São Vicente, 225
Rio de Janeiro - Brazil - 22453-900
rubens@inf.puc-rio.br

ABSTRACT

This work presents TEAM, an architecture, a workflow specification language and the events broadcasting mechanism for the management of workflows in distributed autonomous and heterogeneous environments, which are the predominant characteristics in groups of specialized organizations that collaborate for the accomplishment of common business goals. As in peer-to-peer (P2P) networks, the architecture is based on the functional similarity between execution nodes and in the direct communication (with no mediation) between peers. The distributed activities coordination is achieved by distributed workflow execution engines that process workflows specified in a declarative language that captures the factual, functional and temporal perspectives. We also present the collaborative mechanism used in the broadcasting of facts and in peers management.

Categories and Subject Descriptors

H.4.1 [**Information Systems Applications**]: Office Automation—*Workflow management*

General Terms

Management

Keywords

Interorganizational Workflow Management, Distributed Workflows, Workflow Specification

1. INTRODUCTION

Modern organizations seek ways to minimize their production costs in order to maximize their profits. One of the alternatives they adopt is to specialize themselves to reach production scale.

The (high) specialization demands the coordination of many distributed activities required to produce goods that are valuable to end users. A workflow management system (WfMS) would be helpful in this case, provided that it had communication support and capabilities to control artifacts and context info that cross the organizations borders. To these requirements must be added the ability to deal with the autonomy of the participating organizations and their (inevitable) heterogeneity. Other desirable features of this *interorganizational* WfMS would be the ability to support dynamism while providing the adaptability and flexibility (cf. Sadiq et al in [15]), needed to handle frequent and quick changes typically experienced by the markets nowadays [12].

This work presents an informal description of TEAM, an architectural model and the related technologies and techniques for managing weakly structured, distributed workflows in environments composed of heterogeneous and autonomous organizations.

TEAM was conceived to support environments composed of independent organizations that need to collaborate in the accomplishment of well defined macro (or global) goals. They must be guided by WfMSs capable of dealing transparently with distribution, technological heterogeneity and autonomy of the partners. These WfMSs must also easily interface with each eventually existing local WfMS.

The architecture and the declarative workflow specification language, which is also described, permit us to have, in the same operating environment, a set of open technologies required to handle the interorganizational portion of the global processes. We also present the mechanisms for (1) handling the *collaborative broadcasting* of the messages exchanged between execution nodes during the distributed execution of processes and (2) managing the execution nodes list.

The remainder of this paper is structured as follows: in section 2 we present the TEAM architectural characteristics; section 3 presents the declarative language used to specify processes in TEAM; in section 4 the mechanisms for collaborative dissemination of facts and execution control information, as also the management of the execution nodes list are described; section 5 discusses how interorganizational workflow coordination is accomplished in TEAM; in section 6 we evaluate important characteristics and features provided by TEAM, such as interoperability, flexibility, autonomy and the expressiveness of the specification language. Finally, in section 7, we present the current research situation at the Database Technology Research Lab of PUC-Rio – TecBD/PUC-Rio – in the workflow management area and

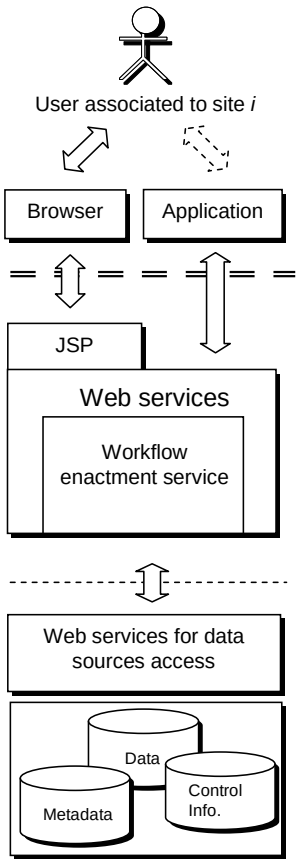

Figure 1: Three-tier architecture of a peer in TEAM

the concluding remarks of the paper.

2. TEAM ARCHITECTURE

TEAM-based environments are composed of sets of linked but independent and similar execution nodes also called *peers* or *pairs*. Each organization may have one or more active nodes, which operate as the environment interfaces to (local) users and other systems interfaces. They provide access authentication, interaction and execution control and also management of the global execution context. A *site* is a logical collection of peers that share the same business context, being associated, typically, to just one organization. There must exist at least one active peer in each participating organization.

There is no functional distinction between peers, although in [13, 14] we admitted that some of them could also work as interfaces to computational grids.

In TEAM environments, each participating organization is free to adopt different technologies for processes and data management as also different data models.

We designed the architecture based on the classic architectural pattern for heterogeneous databases integration consisting of wrappers and mediators ([21]).

In TEAM each peer is a stack of three layers. Figure 1 illustrates an *i-th* generic pair.

A web browser or a .NET application, for instance, may realize the user-environment interface, which is a set of web services. In figure 1, the double dotted arrow represents a possible user interface realized by an application other than a web browser. Stand-alone applications, software agents or local WfMSs may interact directly with the environment. The double dotted line represents the user-environment interface. If this interface is realized by a web browser, a JSP – *Java Server Pages* – intermediate layer may be required.

The second layer corresponds to the operating core of the peer and is composed of a workflow enactment service that may host one or more workflow engines. Each engine manages the *proper* portion of one whole workflow instance, processing rules and defining message contents and controlling task items for/of the users assigned to the corresponding peer. The workflow enactment service is wrapped by one set of web services that form the interface with the first layer and by other set of web services to accept requisitions from workflow engines of other peers in the network (interface 4, cf. the WfMC in [10]). Besides sending requisitions to other workflow engines, one engine may get, and possibly store, artifacts originated from other repositories of the network. Workflow engines are also responsible for synchronizing and/or migrating workflow execution states ([4]). Conceptually, the workflow enactment service layer, besides being a WfMS, also acts as a mediator ([21, 23]), providing to the layer above an integrated view of the distributed execution environment.

The bottom layer provides data persistency for the enactment services layer. This layer is also wrapped by a set of services that realize its interface, via JDBC, with the data sources. With this, peers may adopt diverse persistency technologies and data models. This layer deals, basically, with three data classes: standardized business objects metadata (XML descriptors, in Dublin Core, for instance), object data and workflow control information composed of workflow definition and state (facts). Other classes (e.g. ontologies for data semantic integration) may be added to the repositories as needed. The repositories of a peer may be directly accessed (via the web services wrapper) by workflow engines of other peers. The web services layer works as a XML wrapper, providing data integration and loose coupling.

Interoperability is also granted by the workflow enactment layer that conceptually works as a (distributed) mediator to provide peers´ data and services access in every peer.

Security in TEAM can be provided by the security mechanisms available in today´s web services technology.

The architecture, in a Software Engineering point of view, can be understood as a framework, as it contains invariant and variant aspects of environments in the scope of distributed collaborative applications. It can also be regarded as a multi-agent system, as each pair can be seen as a software agent. In the Database community point of view, the architecture can be regarded as a data integration mecha-

nism based on wrappers and multiple mediators.

TEAM may be applied not only in the public portion but also in the private portion of interorganizational workflows (cf. [16, 18]), easily coexisting with other WfMS.

3. THE WORKFLOW SPECIFICATION LANGUAGE

Many of the workflow specification languages that we studied ([1, 7, 8, 22, 2, 17, 19]), including the ones that provide higher levels of flexibility, use graphical constructs (and/or mappings to XML) to specify the workflows; they are, conceptually, graphs that define sets of partially ordered activities.

Despite the expressiveness of graphic languages, we believe that a single drawing is unable to capture all the dimensions (structural, functional and temporal) required to a precise model while keeping, at the same time, comprehensiveness by humans. This belief is based on, for instance, the need of at least three UML diagrams (commonly use cases, classes and sequences diagrams) to model a simple information system.

Due to this reason, inspired by a plans specification language described in [5] and by Mangan and Sadiq ([11]), we proposed a textual notation to specify workflows models in three layers:

1. Data (or factual) layer, which comprises the facts, i.e., information on the participating entities and on the process execution context;
2. Functional layer, which comprises the data transformation operations (activities that compose the processes) and the structure according to which they may be executed;
3. Temporal layer, which corresponds to the temporal restrictions to be applied to the execution of the operations.

This notation, briefly described in the following sections, specifies workflows by predicates, combining concepts and syntax found in the two references mentioned above. The complete description of the language can be found in [6].

3.1 The Factual Layer:

Data are specified using the following clauses:

1. **peer**($idPeer, aliasPeer$);
2. **role**($idRole$);
3. **executor**($idExecutor, idRole$);
4. **businessObject**($idObject, aliasObject$);
5. **setCondition**($condition$); or
6. **setCondition**($condition, scope$); or
7. **setCondition**($variable = value, scope$);

Peers are identified by an IP port number and the IP address of the host it is running on. An "@" is required to separate both information (e.g. 2458@192.168.7.15). The clause $peer(idPeer, aliasPeer)$; defines an *alias* $aliasPeer$, with local scope (valid within the model only), for the pair with ID $idPeer$. An example is $peer(3469@139.82.120.32, puc)$;. The use of *peer* clauses is optional.

Roles are declared using clauses *role*.

An executor is referenced by his/her ID followed by an "@" followed by the alias of his/her *home peer* (every executor is associated to a peer – called his/her home peer – that stores his/her profile and provides authentication). So, $executor(lpereira@puc, student)$; specifies that $lpereira$, which home peer is 3469@139.82.120.32, interprets the role *student*.

The clause $businessObject(idObject, aliasObject)$; permits the association of a business object to an alias by which this object may be referenced throughout the model. Objects IDs are formed by the concatenation of a local ID (unique in the scope of the peer), an "@" and the peer ID or alias where it is stored.

Clauses *setCondidion* may be used to establish initial execution state. They may be applied to a global condition (as a second parameter), to a unique executor (the scope is the executor ID) or to all executors that interpret a given role. As an example, the clause $setCondition(paa\_done, lpereira)$; specifies that paa_done is the condition to be associated to the executor $lpereira$ when the workflow instance begins. Conversely, if clause $setCondition(paa\_done, student)$; is specified, the initial condition paa_done will be set to all *students* in the model. A condition (corresponding to a fact in the database) can be also set by a statement of the form $variable = value$.

3.2 The Functional Layer:

In the functional layer we list tasks and their pre and post conditions. Tasks are specified by *activity* clauses, as follows:

activity($idActivity$,

$activityLabel$,

$executor$,

$preCondition$,

$action$,

$postConditions$);

where

- $idActivity$ identifies the activity;
- $activityLabel$ is a string containing the label of the

activity;

- *executor* is the executor's ID, which can be an individual or a role;
- *preCondition* is a boolean expression that, when evaluated as true, queues *action* to be executed;
- *action* is the ID of a business object operation, execution environment or workflow engine operation to be executed;
- *postConditions* defines what will be turned into fact after the successful execution of *action*.

Facts may be *global* (only one instance per workflow instance), may refer to individuals or to roles. A global fact is expressed, for instance, as *end_program*. A condition for every *student* would be, for instance, *student.paa_done*, to be evaluated for the *students* executing *paa_done*. For an individual it would be, for instance, *lpereira.paa_done*.

As an example, the following sentence

$activity(EscolhaModuloSinteseImagens,$

"$Escolha\ do\ modulo\ a\ ser\ cursado$",

$aluno,$

$aluno.avaliacao\_fundamentos\_done,$

$aluno.chosen\_module =$

$workflow.chooseOne(aluno,$

$SI\_ModuloA, SI\_ModuloB),$

$aluno.chosen\_module == 1?$

$aluno.SI\_ModuloA\_chosen :$

$aluno.SI\_ModuloB\_chosen);$

specifies that the activity identified by $EscolhaModuloSinteseImagens$, which label is "$Escolha\ do\ modulo\ a\ ser\ cursado$", to be executed by *alunos*, requiring that $avaliacao\_fundamentos\_done$ is a fact for *aluno* to execute a choice between two objects (aliased as $SI\_ModuloA$ and $SI\_ModuloB$). If the first option is chosen, condition $SI\_ModuloA\_chosen$ becomes a fact, otherwise, $SI\_ModuloB\_chosen$ becomes a fact.

One activity must be assigned to just one executor. This limitation can be overcome by splitting the activity in two sub activities.

3.3 The temporal layer:

Information of the temporal layer correspond to the time restrictions associated to the execution of activities of a process. These restrictions are defined relative to the initial date or absolutely. The initial date is specified by the clause **beginInstance**(*aaaammddhhmm*) that are the date, hour and minute after which pre-conditions will start to be evaluated in order execute activities.

After the evaluation of the pre-conditions, the workflow engine associated to the workflow instance in a peer checks for time restrictions, starting or blocking activities.

The clause **intervalConstraint**($idActivity, notBefore, notAfter$); specifies that the activity identified by *idActivity* will start not before $notBefore$ and will end not after $notAfter$. One of these two parameters may be omitted (it would be a nonsense to omit both), but all commas are required. Times are expressed as *[+]yyyymmddhhmm* (*yyyy* for the year, *mm* for the minute, *dd* for the day and *hhmm* for the hour and minute). if a "+" precedes one of these parameters, it means that it is to be relative to the date/hour specified by **beginInstance**(*aaaammddhhmm*).

A restriction expressed in the form **durationConstraint** ($idActivity, minDuration, maxDuration$);, imposes a minimum and/or maximum duration for the activity *idActivity*. Durations are expressed in the form *yyyymmddhhmm* and one of these two parameters may be omitted. Clauses $intervalConstraint$ and $durationConstraint$ can be specified for the same *idActivity*.

4. MESSAGES, DATA MANAGEMENT, QUERY PROCESSING AND TEAM NETWORKS MANAGEMENT

Peers communicate exchanging messages. These messages are classified according to the objective, according to the range and according to the dissemination mechanism.

Messages are:

- to announce a (voluntary) entry or exit of a peer. In this case, the peer that is entering or leaving the network sends the message, which needs to reach all the other peers. The message broadcast is done in a collaborative way (as we will see in details bellow). When a peer involuntarily leaves the network (due to an exception, for instance), the message announcing that a peer left the network is broadcast (also collaboratively) by the peer that notes its absence, probably its *father* in the network;
- query propagation. Queries may be placed in order to find business objects or to query the status of a process. They are also broadcast.
- to announce a new workflow instance. In this case, messages contain the complete specification (model) of the new workflow instance and are originated in the peer where the workflow instance is published. They are addressed to the peers where the workflow instance will be executed.
- to inform a new fact, such as the conclusion of a task. These messages are sent by the peers where the facts occurred and are addressed to the peers that run any part of the same workflow instance;

- to inform the conclusion of a workflow instance section. Every peer that controls a portion of a workflow, at its end, sends this fact to the peer that controls the workflow instance (the same that published the instance).
- to inform a workflow instance cancellation. The peer that published the workflow instance may inform its end, be due to the end of all portions or be due to a cancellation by the publisher.

A new query instantiates a new query execution thread on the peer that it is originated. The query is replicated *collaboratively* throughout possibly all peers in the network, i.e., every peer that receives a query, besides executing it against its own data repository, replicates the query to other n peers (we use $n = 2$) of its own list of connected peers, as we will see below.

The query execution thread generates the query command, gives to it an unique network ID (a sequential number concatenated to the peer ID), initializes a response control vector (which is used to check if the responses from all other peers were received – a *timeout* may occur), passes the query command to the n peers it points to, at the same time it executes que query against its own data repository. The query replication process continues until all peers receive and process the query. The results are sent directly to the peer that originated the query. Queries already processed are discarded.

The list of active peers must be kept updated in every peer of the network. The messages dissemination mechanism also requires that they must be structured in the same manner in each peer.

The list of peers is structured as an *n-ary* tree, which is implemented as a vector of peer IDs sorted ascending. Figure 2-a illustrates the case where $n = 2$ – a binary tree – omitting, for simplicity, the suffix of the IDs. M is the number of peers in the network, being equal to 10 in the example.

A peer that receives a query to be processed or a message to be forwarded knows its position – suppose i – in this vector. The peer determines the beginning of the list of n peers (initial position $p(i)$ in the vector), to which it must forward the query or message. $p(i)$ is evaluated by the following simple expression:

$$p(i) = 2 + (i - 1)n \quad (1)$$

As an example, still referring to figure 2 and applying eq. 1, peer 1144 (position $i = 4$ in the list) needs to propagate a query it receives to the $n = 2$ peers with IDs initiating at position $p(4) = 2 + (4 - 1)2 = 8$, i.e., to peers 1313 and 1318. If it is the case that $p(i)$ or any of its $n - 1$ successors is greater than M, peer position 1 (root) will be the peer to which the query or message must be sent to.

As we said, queries or messages already received (and so already forwarded) by a peer are discarded, requiring that each peer must keep a list of message IDs (recently) received.

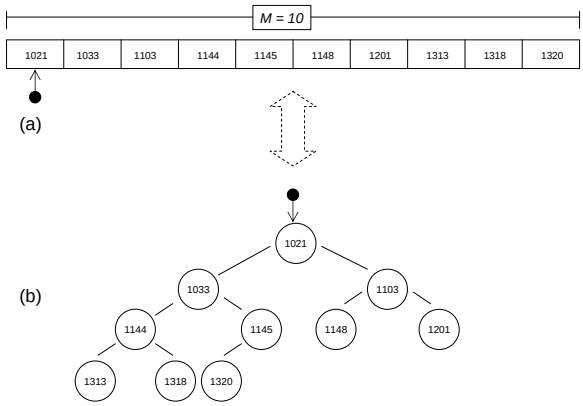

Figure 2: Connected peers list structure and implementation in TEAM. (a) is the vector of connected peers IDs as stored in every peers and (b) is the tree structure it represents.

With this, the messages/queries broadcasting burden is distributed to all peers in a TEAM network.

Despite TEAM networks are not *ad hoc* networks, as all partners are required to participate playing specific roles in a process, it is important to consider the cases that the peer lists are not synchronized. This may happen right after a peer leaves the network, leaving *orphans* peers, and so peers that will receive no messages or queries to be processed and (re)forwarded.

Generally speaking, two possible consequences may arise from this temporary non synchronism: (a)one or more peers receive more than one copy of a message/query and (b) one or more peers receive no messages/queries.

As peers only process the first copy of a query/message, case (a) is trivially handled. Case (b), although, needs a more complex analysis, which depends on the reason for the non synchronism.

The non synchronism of the peers list may be due to:

1. The entry of a new peer in the network, while its registry has not been completely processed by all existing peers yet. In this case, if there are already n peers in the network – the new one being the $(n + 1)th$ –, $n - 1$ peers would have the original list while the peer to which the new peer has introduced itself would have $n + 1$ peers in its list. This case unfolds into other two (sub)cases:
   (a) if the new peer, after having its ID inserted in the list of the peer to which it has introduced itself, is to become one of its sons, this one processes the message, forwards it to its previous and to the new sons and then updates its peers list. During this process, the peers that receive the messages also update their lists;

(b) if the new peer, after having its ID inserted in the list of the peer to which it has introduced itself, is not to become one of its sons, this one processes the message, forwards it to its current sons and to the future father of the new peer, also informing them the entry of the new peer. In this case, it is also important that the peer to which the new peer introduces itself, verifies that, with the entry of this new pair – even not becoming one of its sons – the list of its sons would be affected. If this is the case, messages must be sent also to the previous and new sons, as in the previous case.

2. A peer announced its exit from the network, while the message is not completely processed by all other peers. In this case, the peer that leaves the network may ask its father to temporarily "adopt" its sons or simply leave the network, reducing this case to case below;

3. A peer involuntarily left the network. The peer that notices the missing of a peer adopts temporarily its sons, if it is its father, or tells the father of the missing peer that one of its sons has left the network.

5. DISTRIBUTED WORKFLOW MANAGEMENT

Workflows specifications is implemented in XML for storage and distribution. As we already discussed, the list of executors (with their respective home peers) is given in the model. This list defines which peers will need a copy of the workflow specification, where portions of the workflow will be executed, as execution context and the interface with the user are handled by executors' home peers.

When a new workflow instance is published, the workflow enactment service of the peer that publishes it instantiates a workflow engine to handle the distribution of the model to the other peers that will execute any part of the workflow. The workflow enactment service of each contacted peer will then instantiate a new workflow engine to control its portion of the workflow instance. This engine will remain active until the peer that published the instance signals its end.

Workflow engines keep waiting until the time specified by clause **beginInstance**(*aaaammddhhmm*) comes, to start the first pre conditions evaluation loops in order to activate the activities that depend on them. These activities are immediately started or placed in the work lists (cf. [10]) of the executors assigned to them. Tasks that are accomplished will possibly generate new facts and, consequently, new pre conditions will be evaluated in the next pre conditions evaluation loop. The workflow engines keep waiting until new activities are accomplished and/or new facts are received from other peers, so they can start other pre conditions evaluation loop.

This sequence repeats until all activities of all peers finish. The peer that published the workflow then reports to all execution peers that the workflow instance has finished so they can kill their corresponding workflow engine threads. In other words, a workflow engine is finished only by the peer that publishes the instance. This handles properly cancellations and modifications of the model during runtime.

Differently than with queries (which are broadcast), facts are selectively broadcast, i.e., are only sent by each peer that generates a fact to the peer(s) executing portions of the same workflow instance.

6. EVALUATING TEAM

TEAM grants interoperability and autonomy, two requisites for a workflow specification language mentioned in [3], which we extend to the operating environment as a whole.

Flexibility is another important requisite that TEAM complies. Independent partners tend to be subjected to many dynamic market forces. Consequently, the *contract*, corresponding to the public portion of a (macro) process mentioned in [16, 18], which is supposed to be structured, tend to be greatly influenced by these forces. By using a workflow specification language that is based on pre and post-conditions, we are able to specify processes from non structured to structured in TEAM. The flexibilities by selection and by adaption (cf. [9]), with correction steps *a priori* and in runtime, may be achieved using the mechanism for selection on alternatives provided by the language or by addition or redefinition of facts during runtime.

We used a language processor prototype (see [6]) to verify the expressiveness of the language. We implemented sixteen out of twenty workflow patterns listed by van der Aalst in [20]. The patterns Multiple Merge, Discriminator, Interleaved Parallel Routing and Cancel Activity could not be concisely implemented using the constructs currently provided by the language. We believe that this could be accomplished by reinforcing boolean expressions formulations, using the same C language syntax, for instance. We left this issue for future work.

The declarative process specification facilitates the reuse of parts of the model on one hand, but turns the specification a bit verbose. For the structured parts of the processes it is possible to use graphical tools to automate the generation of the ordering information (pre and post-conditions).

7. CONCLUSION

At the Database Technology Research Laboratory – TecBD/PUC-Rio – we are starting the development of an environment for research in distance learning, using TEAM to coordinate the distributed and collaborative learning content execution. This environment will be using the workflow enactment service prototype that we developed to demonstrate part of our architecture.

In this paper we presented the characteristics of the components and connectors of the architecture named TEAM, conceived to provide data integration and process coordination of collaborative execution in distributed and heterogeneous environments composed of loosely coupled execution nodes. These issues are present in interorganizational workflows.

This work presented an informal description of the components of the architecture, the mechanisms for managing workflow states at the execution nodes of the network, the mechanisms for managing the network itself as also for collaboratively broadcast control messages and queries.

We also described a declarative workflow specification language that captures the factual, functional and temporal dimensions of the model using predicates.

TEAM may be applied both in the public and in the private portions of interorganizational workflows, being able to interface with other WfMSs.

8. ACKNOWLEDGEMENT

We would like to thank Banco Central do Brasil for the partial support given to this work.

9. REFERENCES

[1] W. Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[2] R. M. Bastos and D. D. A. Ruiz. Extending UML activity diagram for workflow modeling in production systems. In *35th Annual Hawaii International Conference on System Sciences (HICSS'02)*, volume 9, 2002.

[3] M. Bernauer, G. Kappel, G. Kramler, and W. Retschitzegger. Specification of interorganizational workflows - a comparison of approaches. *Proceedings of the 7th World Multiconference on Systemics, Cybernetics, and Informatics*, pages 30–36, 2003.

[4] L. Böszörmenyi, H. Groiss, and R. Eisner. Adding distribution to a workflow management system. *10th International Workshop on Database and Expert Systems Applications (DEXA)*, 1999.

[5] A. E. M. Ciarlini. *Geração Interativa de Enredos*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, 1999.

[6] L. A. de Moraes Pereira. *TEAM: Uma arquitetura para gerência de e-workflows*. PhD thesis, Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio, 2004.

[7] M. Dumas and A. H. M. ter Hofstede. UML activity diagrams as a workflow specification language. In M. Gogolla and C. Kobryn, editors, *UML 2001 - The Unified Modeling Language. Modeling Languages, Concepts, and Tools. 4th International Conference, Toronto, Canada, October 2001, Proceedings*, volume 2185 of *LNCS*, pages 76–90. Springer, 2001.

[8] H. Eshuis. *Semantics and Verification of UML Activity Diagrams for Workflow Modelling*. PhD thesis, Universiteit Twente, 2002.

[9] P. Heinl, S. Horn, S. Jablonski, J. Neeb, K. Stein, and M. Teschke. A comprehensive approach to flexibility in workflow management systems. In *Proceedings of the international joint conference on Work activities coordination and collaboration*, pages 79–88. ACM Press, 1999.

[10] D. Hollingsworth. The workflow reference model. Technical Report TC00-1003, Workflow Management Coalition, 1995.

[11] P. Mangan and S. Sadiq. On building workflow models for flexible processes. In X. Zhou, editor, *Thirteenth Australasian Database Conference (ADC2002)*, Melbourne, Australia, 2002. ACS.

[12] M. Mecella, B. Pernici, M. Rossi, and A. Testi. A repository of workflow components for cooperative e-applications. *Proceedings of the 1st IFIP TC8 Working Conference on E-Commerce/E-Business*, pages 73–92, 2001.

[13] L. A. M. Pereira, R. N. Melo, F. A. M. Porto, and B. Schulze. TEAM: Using the grid to enhance e-learning environments. *II Workshop de Grade Computacional e Aplicações – WGCA'2004*, fevereiro 2004.

[14] L. A. M. Pereira, R. N. Melo, F. A. M. Porto, and B. Schulze. A workflow-based architecture for e-learning in the grid. *The First International Workshop on Collaborative Learning Applications of Grid Technology – CLAG'2004*, April 2004.

[15] S. Sadiq, W. Sadiq, and M. Orlowska. Pockets of flexibility in workflow specification. In *20th International Conference on Conceptual Modeling (ER-2001)*, number 2224 in Lecture Notes in Computer Science. Springer-Verlag Heidelberg, 2001.

[16] W. van der Aalst. Loosely coupled interorganizational workflows: Modeling and analyzing workflows crossing organizational boundaries. *Information & Management*, 37(2):67–75, March 2000.

[17] W. van der Aalst and A. Hofstede. YAWL: Yet another workflow language (revised version). QUT Technical Report FIT-TR-2003-04, Queensland University of Technology, Brisbane, 2003.

[18] W. M. van der Aalst and M. Weske. The p2p approach to interorganizational workflows. *Proceedings of the 13th Conference on Advanced Information Systems Engineering (CAiSE'01)*, pages 140–156, 2001.

[19] W. M. P. van der Aalst, L. Aldred, M. Dumas, and A. ter Hofstede. Design and implementation of the YAWL system. Qut technical report, Queensland University of Technology, Brisbane, 2003.

[20] W. M. P. van der Aalst, A. H. M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14:5–51, July 2003.

[21] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, March 1992.

[22] G. Wirtz, M. Weske, and H. Giese. Extending UML with workflow modeling capabilities. *7th International Conference on Cooperative Information Systems (CoopIS'2000)*, pages 30–41, 2000.

[23] M. T. Özsu and P. Valduriez. *Princípios de Sistemas de Bancos de Dados Distribuídos*. Editora Campus, 2 edition, 2001.