The Architecture of a Mobile Emergency Plan Deployment System

FÁBIO MEIRA DE OLIVEIRA DIAS
MARCELO TÍLIO MONTEIRO DE CARVALHO
Grupo de Tecnologia em Computação Gráfica / TeCGraf, Pontifícia Universidade Católica do Rio de Janeiro,
Rua Marquês de São Vicente 225, Rio de Janeiro, RJ, Brasil, CEP 22453-900
{fmdias, tilio}@tecgraf.puc-rio.br

Abstract. This paper describes the implementation of a workflow management system to support mobile GIS workgroup applications and highlights the design of the workflow definition language the system offers. The system features a hierarchical execution model in which each instance is able to delegate the execution of sub-workflows to other active instances. It incorporates instance discovery and event handling services, that enhance the proper treatment of disconnections, and adopts an optimist replication strategy to handle workflow context variables. The workflow definition language features special constructors that improve the legibility of large workflow definitions. The results reported here build on the experience accumulated with the implementation and deployment of an emergency management application that offers action plans, easy access to geographic data and tight control over the resources allocated to face an emergency.

1. Introduction

Workflow management systems are frequently used for modeling, monitoring and controlling the coordinated execution of activities performed by workgroups in a variety of contexts. Combined with geographic data management features, they provide an environment to implement GIS workgroup applications, such as emergency management applications. Going one step further, the growing computational power of portable computers makes it possible to create mobile versions of such applications. However, many existing solutions do not work satisfactorily in the presence of frequent disconnections and in heterogeneous environments. In fact, one can argue that several limitations are not due to the state of current technology, but rather they are intrinsic to any system subject to a degree of mobility [23].

We describe in this paper the major decisions that guided the design of the next version of InfoPAE [7,8], an emergency management application that offers sophisticated action plans, easy access to vital geographic data and tight control over the resources allocated to face an emergency. The sytem is built upon a workflow management system that supports mobile GIS workgroup applications, subjected to disconnections. We also highlight the major features of XPAE, the workflow definition language the system offers.

Several strategies are seamlessly integrated to address the challenges of mobile GIS workgroup applications. The system features a hierarchical execution model in which each instance is able to delegate the execution of sub-workflows to other active instances. The model offers flexible consensus mechanisms and adequately deals with disconnections. The system also incorporates instance discovery and event handling services, that enhance the proper treatment of disconnections, and adopts an optimist replication strategy to handle workflow context variables.

The workflow definition language, XPAE, has special constructors that improve the legibility of large workflow definitions. Although not limited to, its major application is the definition of emergency plans. Indeed, its design builds on the experience accumulated during the last three years with the implementation of over fifty emergency plans for large industrial facilities.

The first significant project regarding distributed execution of workflows was Exotica/FMQM (FlowMark on Message Queue Manager) [3]. The system was later modified to take into account disconnected clients (Exotica/FMDC) [2]. Domingos [9] proposed a hierarchical structure with a core system connected to a high-speed network, which maintains the official information about the execution of workflows and data updates executed by the participants. The CoAct (Cooperative Activity Model) transaction model [17] was also extended to support mobile users. Grigori [11] proposed combining a cooperative protocol with a traditional workflow model.

The efforts described above adopted different strategies to deal with workflow execution. In some cases, the emphasis was on the distribution of the process over several machines, without taking into account disconnection. In other cases, disconnection had to be programmed, which is not a viable alternative in mobile environments and which may even permanently stop the execution, when coupled with pessimistic strategies. Also, process structuring is frequently lacking, which limits the applicability of the mechanisms developed. Indeed, most of the solutions

found in the literature for such an environment were not fully implemented, or were adapted from solutions originally designed for less complex environments or addressed only part of the problem. By contrast, the solution proposed here seamlessly integrates several strategies and has a fully operational, concise implementation.

As for emergency management systems, FRIEND [5], INCA [15] and MokSAF [18] are representative examples. In particular, the MokSAF system supports route planning by combining AI techniques with GIS. Other examples of multi-agent systems with internal planning components are RETSINA [20] and HIPaP [21]. A survey of cooperative multi-agent systems appears in [19].

The XPAE workflow definition language shares some of the concepts of XRL [1], a language that supports the exchange of workflow specifications across companies. XPAE is also consistent with the Workflow Management Coalition [27] proposal. We preferred not to adopt any of the Web services flow languages [4,24] to express emergency plans mostly to retain flexibility and to maintain compatibility with legacy emergency plans.

The paper is organized as follows. Section 2 gives the motivation for the work described in the next sections. Section 3 covers the workflow definition language. Section 4 describes the major decisions that guided the design of the system. Finally, Section 5 contains the conclusions.

2. Motivation

The major motivation for the work reported in this paper is the implementation of mobile emergency management applications, a class of mobile GIS applications designed to improve the response to emergency situations. Such systems must handle emergency plans, as well as conventional and geo-referenced data, in a mobile environment. An emergency plan is a predefined workflow procedure that helps guiding the actions of emergency teams during an event. Therefore, an emergency plan describes instructions to human beings and consists of actions that are meant to be manually executed, most of the time.

Throughout the paper, we will use an example based on an overly simplified emergency situation. First, we suppose that the emergency team is composed of an officer-in-charge, an officer-on-duty and several sub-teams with specific responsibilities.

The emergency situation, broadly speaking, is:

- when the emergency is reported, the officer-on-duty starts the emergency plan from his post and characterizes the scenario of the emergency;
- he may rapidly close the case and file a report using the system, if the emergency was a minor one;
- otherwise he summons the officer-in-charge, who then takes control of the emergency plan, perhaps moving the execution to an emergency room facility;
- the officer-in-charge may access geo-referenced data, such as a map of the region where the emergency occurred;
- the officer-in-charge may display on the map the anticipated or observed scenarios, which act as anchors to specific emergency sub-plans;
- the officer-in-charge may click on a scenario and select a specific emergency sub-plan;
- he may assign the sub-plan to the appropriate operational sub-team, passing control of the sub-plan to the mobile device of the sub-team;
- the sub-teams and the officer-in-charge then exchange data and messages using the mobile devices and the central facility.

Therefore, the system must have a GIS module that offers a geo-referenced visual space, where scenarios can be displayed and used as anchors to access emergency sub-plans, and a workflow execution module to display and control plan execution. Moreover, these modules must operate in a mobile environment so that the sub-teams and the officer-in-charge may exchange data and messages. This requires a balanced integration of the modules since emergency control, to some extent, shifts from the geo-referenced visual space to the workflow control interface.

The workflow definition language must, in turn, treat scenarios as first class objects and must provide specific constructs to help create emergency plans. Indeed, emergency plans are highly structured and follow specific standards. For example, one of the InfoPAE customers structures a plan into well determined phases, which are: (1) emergency identification and immediate actions; (2) mobilization of the emergency teams; (3) characterization of the emergency scenario; (4) combat procedures; and (5) closing procedure.

In general, an emergency scenario or simply, a scenario is characterized by well-defined attributes, usually indicating the nature of the emergency, the product involved (if applicable), the facility where the emergency occurred and the environmental conditions under which the emergency took place. An example of a scenario would be “spill of oil type II at the pier, with high tide and south wind”.

Each scenario is associated with one or more methods, specified as workflows describing the actions the emergency teams must follow and defined according to a specific combat logic. Each action, or group of actions, may be associated with people or institutions to be contacted, types of resources required to perform the

action, emergency teams to be mobilized and general documentation to be used.

There is also considerable similarity, if not redundancy, between methods for different scenarios of the same plan. In fact, they frequently differ only on the associated information.

Users also strongly suggested that it should be possible to invoke methods by clicking on the location of the accident, visualized on the user interface. This can be achieved indirectly by geo-referencing the scenarios and creating a *scenario information layer*. Since methods are always linked to scenarios, they become implicitly geo-referenced objects that can be activated from the scenario information layer.

Finally, any sub-plan should also be explicitly geo-referenced by accessing a system variable that gives the location of the mobile device where the sub-plan is being executed (assuming that the mobile device has a GPS).

3. The design of XPAE

The workflow definition language, XPAE, includes elements to:

- define sub-plans
- define classes of (geo-referenced) scenarios
- associate methods to scenarios
- structure elementary actions
- hyperlink actions, and other parts of a plan, to objects and documents in a database
- help reduce plan redundancy

Table 1 enumerates the major elements of XPAE, some of which we discuss in more detail in what follows.

Element Group	Elements
Related to plans	“plan”, “call”
Related to scenarios	“type_of_scenario”, “scenario”, “class_of_scenario”, “method”
Related to variables	“variable” , “value”, “domain”
Related to action control and execution	“do”, “ask”, “group”, “test”, “repeat”, “classify”, “associate”
Related to the repository	“user”, “permission”, “resource”, “style”, “include”

Table 1 XPAE Elements

The element “plan” defines a collection of actions, structured as a workflow. To facilitate its definition, a plan may be decomposed into sub-plans (or sub-workflows) and it may declare classes of scenarios equipped with methods. A plan may be executed as an *independent* process and it may be executed by more than one process. If executed by a separate process, it may be *interruptible* or *cancelable.* A sub-plan behaves exactly as a subroutine, callable from the plan where it was defined. Parameter passing is always by value. Without going into details, in general, scope rules are fairly rigid to avoid side-effects.

Inter-process communication is based on the event handling service described in section 4.3 and is not discussed here, for brevity.

The element “type_of_scenario” corresponds to the declaration of a type of scenario. It includes a list of attributes and, optionally, indicates a geo-referenced attribute of the list. An attribute has a name and a type, which in the present implementation is just “scalar”, “geo-referenced” or a type of scenario defined in the plan. The element “scenario” describes a *scenario* of a given type. The element “class_of_scenario” declares a *class of scenarios* and indicates the class type, the class variable and one or more a *methods*, defined as plans.

The element “variable” represents a variable declaration to be used in a plan. In the current implementation, the variable type can be “*scalar*”, “*geo-referenced*” or one of types of scenario defined in the plan. The element “value” defines a constant and the element “domain”, a set of constants.

There is an internal pseudo-variable, `HERE`, whose value is always the location of the mobile device where the plan is being executed (assuming that the mobile device has a GPS).

The element “do” defines an elementary action the emergency team must perform. The element “ask” requests information from the user and sets internal control variables. The element “group” (in all its variations) captures the usual workflow action structuring constructs. Finally, the element “test” is the usual if-then-else construct and the element “repeat”, also defined in the XRL language, represents the usual iteration construct.

The element “classify” defines classifications, whereas the element “associate” links a syntactical element of the plan with an object in the database, such as a document, report, etc. Both are applicable to a variety of syntactical elements.

The element “repository” represents, in XML, one or more plans and objects in the database. This is the initial element of the XML document.

Finally, the semantics of the XPAE language is defined with the help of Petri nets, as suggested in [1]. Furthermore, the implementation of the language, using the facilities and services described in Section 4, is straightforward and will not be discussed for brevity.

4. System design

4.1. Execution model

The system internally models a workflow definition as a graph, much in the same way that the semantics of the language is specified using Petri net concepts. Omitting the details, elements of the graph can be of two types: `Place` or `Transition`. A `Transition`, in turn, can be a `Workflow` or a `Task`. A `Place` works as a container for tokens, which determine the control flow during execution of the workflow. At the same time, each `Workflow` can be a sub-graph corresponding to the definition of a sub-workflow. Finally, an `AutomaticTask` represents an automatic activity whose execution is performed by means of a call to a method called `execute` defined in the `Action` interface.

To increase flexibility, along the lines of *adaptable workflows* [6,11,14], a hierarchical execution model is supported, in which each instance is able to delegate the execution of sub-workflows to other active instances. As each sub-workflow can be composed by other sub-workflows, it is possible to delegate the execution of each one of them to different instances of the system, recursively. Therefore, the delegated instances have the autonomy to manage the execution of any workflow under its responsibility, be it based upon a complete definition or just part of a larger definition.

The execution model may be abstracted as a tree in which the edges represent a delegation. Intermediary nodes act simultaneously as servers and clients, with their parent nodes being the servers and their child nodes being the clients. When the execution of each delegated workflow terminates, the event is informed to the instance from which the delegation originated. Eventually, the execution is completed in the node corresponding to the root of the delegation tree.

This hierarchical execution model indeed offers increased flexibility. However, we must analyze its behavior in the presence of disconnections, specifically, what happens when: (1) the execution of a sub-workflow ends and it is not possible to contact the originating instance; and (2) the disconnection periods last too long. The first problem is dealt with by postponing the actual termination to the moment when communication becomes possible and with the second problem by implementing a leasing mechanism according to which each delegation is subject to an expiration time. When this period is over, the sub-workflow is made available once again. Therefore, if there is any equipment failure or an excessive delay in the execution of a critical activity, the execution of the workflow as a whole does not risk being permanently hindered.

The execution model also allows an activity or a sub-workflow to be delegated to more than one instance, simultaneously. The user decides when to adopt multiple delegations either at workflow design time or during workflow execution. In the current implementation, execution of the (parent) workflow proceeds when the first instance of the sub-workflow successfully terminates. Different strategies might be adopted as well, such as waiting for the termination of all sub-workflow instances or requiring a given quorum (using the service described in Section 4.4).

The design decisions described above might have delicate consequences when used inappropriately. If a sub-workflow is delegated to more than one instance and each one of them further delegates other nested sub-workflows, a large number of cancellations might result, for example, when the first successful termination reported automatically invalidates the other delegations. In order to avoid this kind of situation, the system does not permit the occurrence of recursive delegations, which does not solve the problem completely, but simplifies it considerably. Naturally, other strategies could have been adopted.

Finally, recall from the definition of the XPAE language in Section 3, that a sub-workflow that represents a method for a given scenario is implicitly geo-referenced, if scenario is geo-referenced. In such cases, the system supports a geo-referenced partial representation of the global workflow execution, based on scenarios, that is quite useful to help visualize the current state of the execution, the location of the teams and other resources associated with the sub-workflows. In general, using the internal pseudo-variable `HERE`, the workflow designer may geo-reference any plan. For example, he may decide to geo-reference rescue sub-plans of the global emergency plan to automatically control the location of the rescue teams, that is, the teams that are executing the rescue plans.

4.2. Instance Discovery Service

Instance discovery refers to a (network) service that allows each instance of the system to rapidly identify (on the network) other instances that are simultaneously operational at a given time (on fixed or mobile devices).

To implement this service, the system features a protocol that incorporates concepts from service discovery architectures, such as Jini [26] and SLP (Service Location Protocol) [12]. The instance discovery protocol first broadcasts identification requests to which accessible instances reply informing of their existence. Each reply message contains a reference that permits the use of its interface for communication. Upon receiving a reply, the protocol collects the corresponding reference and adds it to its list of known instances. After that, it sends an acknowledgement with the reference of its associated instance, allowing further communication between the parties involved.

However, in the presence of disconnections, a reference to a remote instance does not guarantee that it can be successfully used. The correct behavior of the system would be to inform its known peer instances prior to disconnection, which might not happen due to failures. Therefore, each known instance is periodically *pinged* to check that it is still up and running. If an instance cannot be contacted, its reference is removed from the list of known instances.

Once the (network) instance discovery service is equated, it is a simple matter to implement, for example, a geographic location service that permits an instance of the system to identify the geographic location of mobile instances that are simultaneously operational. Indeed, this service can be implemented by a simple query on the value of the `HERE` pseudo-variable of the mobile instances, after running the (network) instance discovery service.

4.3. Event Handling Service

An inter-process communication mechanism built into the workflow management system can be very useful, for example, to notify abnormal occurrences during the execution of a workflow. In general, Hagen [13] argues that the use of events can be beneficial to the integration of heterogeneous tools with no common API, such as conferencing systems and other groupware tools.

However, in a distributed environment with disconnections, the task of transmitting notifications becomes non trivial. It is not possible, for example, to determine the time that it takes for a notification to reach its destiny, to detect if the notification was lost or to guarantee the order in which the notifications reach the destination.

An event handling service is implemented that allows the registration of events of interest, such as starting and ending the execution of activities or workflows, updating context variables, (dis)connecting other instances or communicating the geographical location of events, resources, etc. to peer systems. The service offers a generic registration and notification interface that allows its use with relatively heterogeneous components.

For example, the event handling service permits implementing a more sophisticated geographic location service that allows an instance of the system to identify the *last known* geographic location of mobile instances. Indeed, it suffice to use this service to periodically register the value of the `HERE` pseudo-variable of the mobile instances.

In the current implementation, notifications are retained when immediate delivery is not possible and a configurable number of attempts are made until the message is finally discarded. In certain cases, however, it might be necessary to provide increased delivery guarantees and more sophisticated strategies. In particular, the service indeed permits registering intermediaries to relay the notification generated by the originator of the event to its final destination.

The service implements the concept of *lease* [10], similar to that of Jini [26] and other similar systems, coupled with a renewal mechanism that permits to extend the period during which the registrations are valid. Together, these features enable the development of sophisticated strategies that take into account the fact that some machines might remain disconnected for a period longer than expected or even indefinitely.

4.4. Data Replication Service

Each workflow has a set of context variables, which determine the control flow during execution, among other purposes. Therefore, to achieve the desired level of autonomy of each delegation, the system must create and maintain replicas of the context variables.

In the presence of failures or intermittent communication, optimist replication schemes offer a number of advantages over pessimist schemes. In particular, they improve user autonomy, as they allow reading and writing of any data at any time. On the other hand, they possibly lead to inconsistent local states, thereby requiring consensus mechanisms to guarantee convergence to a common state.

In view of these observations, an optimist data replication service is implemented to handle replication of workflow context variables. A replica is created whenever a delegation occurs and is destroyed when the delegation ends upon notification of the originating instance. When a delegation is about to complete, a consistency check is triggered. Conflict resolution can be either manual or automatic, using one of the simple algorithms the system implements. In both cases, the delegated instance must ensure that conflict resolution does not compromise the execution of the sub-workflow involved. In some cases, however, it might be necessary to perform compensating steps, which must be included in the workflow definition. This type of strategy is frequently used to handle exceptions in workflow execution [16,25].

The resolution methods developed are similar to those found in many systems. For example, the method `EARLIEST` chooses the earliest between the current values in the originating and delegated instances and the method `OVERWRITE` discards the update to the originating instance and retains the current value in the delegated instance.

The user may also define application-specific resolution methods, perhaps using geographic operators to suit his needs. For example, consider an emergency plan to recover oil from an oil spill at seashore. Suppose that the team controlling the execution of this plan (the

originating instance) delegates to several sub-teams the cleaning procedures. Suppose also that all these instances share a context variable defining the area cleaned. However, due to environmental conditions, the exact area each instance cleans may actually vary from the original specification. Then, an application-specific resolution method would replace the area at the originating instance with the overlay of all areas the delegated instances report, which is the total area cleaned.

5. Conclusions

We described in this paper the major design decisions of a workflow management system that supports mobile GIS workgroup applications, subjected to disconnections.

The workflow definition language is carefully designed to help specify large emergency plans, among other applications. The system offers an execution model, based on the concepts of sub-workflow and delegation, which is sufficiently general to accommodate, equally well, other workflow definition languages. The model permits adjusting the degree of flexibility to the desired level, while supporting workflow execution in the presence of disconnections. The system features an integrated infrastructure to deal with the major problems of instance discovery, event handling and data replication.

With the help of simple examples, we indicated how the execution model and the infrastructure can be combined with GIS features to address the specific needs of a sample mobile GIS application.

Our future work includes the execution of extensive testing in real mobile devices. Tha availability of our chosen environment (Java) in such devices is constantly increasing at the same time that the required technologies such as GPS become more commonplace. We expect to enhance the system towards being more spatial-aware, which will certainly make it more relevant in the context of GIS applications.

Finally, we stress that the design of the system and of the workflow definition language profited from the experience accumulated during the last three years with the implementation of the InfoPAE system and the specification of emergency plans for over fifty large industrial facilities.

References

[1] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management", *The Journal of Circuits, Systems and Computers* (1998) 21–66.

[2] G. Alonso, R. Günthör, M. Kamath, A. Agrawal, A. Abbadi, C. Mohan, "Exotica/FMDC: Handling Disconnected Clients in a Workflow Management System", *Proc. 3rd Int'l Conf. on Cooperative Information Systems*. Vienna (1995).

[3] G. Alonso, A. Agrawal, A. Abbadi, C. Mohan, R. Günthör, M. Kamath, "Exotica/FMDC: A Persistent Message-Based Architecture for Distributed Workflow Management", *Proc. of the IFIP WG8.1 Working Conf. on Information Systems Development for Decentralized Organizations*. Trondheim, Norway (1995).

[4] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, et. al., *Business Process Execution Language for Web Services* (Version 1.1). BEA, IBM, Microsoft, SAP and Siebel (2003).

[5] B. Bruegge, K. O'Toole, D. Rothenberger, "Design Considerations for an Accident Management System". *Proc. of the Conference on Cooperative Information Systems* (1994) 90–100.

[6] C. Bussler, "Adaptation in Workflow Management". *Proc. of the 5th Int'l Conf. on the Software Process (ICSP5)*, Computer Supported Organizational Work. Illinois, USA (1998).

[7] M. T. Carvalho, M. A. Casanova, F. Torres, A. Santos, "INFOPAE - an Emergency Plan Deployment System". *Proc. International Pipeline Conference*. Calgary, Alberta, Canada (2002).

[8] M. A. Casanova, T. Coelho, M. T. M. Carvalho, E. T. L. Corseuil, H. Nóbrega, F. M. Dias, C. H. Levy, "The Design of XPAE – An Emergency Plan Definition Language". *Proc. IV Workshop Brasileiro de Geoinformática (GEOINFO)*. Caxambu, Minas Gerais, Brasil (2002) 25–32.

[9] H. J. Domingos, J. L. Martins, N. Preguiça, S. Duarte, "A Workflow Architecture to Manage Mobile Collaborative Work". *Proc. First Workshop on Mobile Computing* (1999).

[10] J. Gray, D. Cheriton, "Leases: an efficient fault-tolerant mechanism for distributed file cache consistency". *Proc. of the 12th ACM Symp. on Operating Systems Principles*. ACM Press. (1989) 202–210.

[11] D. Grigori, H. Skaf-Molli, F. Charoy, "Adding Flexibility in a Cooperative Workflow Execution Engine". *Proc. High Performance Computer and Networking (HPCN Europe 2000)*. Amsterdam, The Netherlands (2000) 227–236.

[12] E. Guttman, "Service Location Protocol: Automatic Discovery of IP Network Services". *IEEE Internet Computing* 3(4) (1999) 71–80.

[13] C. Hagen, G. Alonso, "Beyond the Black Box: Event-based Inter-Process Communication in Process Support Systems". *Proc. of the 19th IEEE Int'l Conf. on Distributed Computing Systems*. Austin, Texas (1999).

[14] Y. Han, A. Sheth, C. Bussler, "A Taxonomy of Adaptive Workflow Management". *Proc. CSCW Workshop on Adaptive Workflow System*. Seattle, USA (1998).

[15] W. Iba, M. Gervasio, “Adapting to User Preferences in Crisis Response”. *Proc. of Intelligent User Interfaces* (1999) 87–90.

[16] S. Jablonski, C. Bussler, *Workflow Management: Modeling Concepts, Architecture, and Implementation*. International Thomson Computer Press (1996).

[17] J. Klingemann, T. Tesch, J. Wäsch, “Enabling Cooperation among Disconnected Mobile Users”. *Proc. Conf. on Cooperative Information Systems* (1997) 36–46.

[18] T. Lenox, T. Payne, S. Hahn, M. Lewis, K. Sycara, *MokSAF: How should we support teamwork in human-agent teams?* Technical Report CMU-RI-TR-99-32, Robotics Institute, CMU (1999).

[19] V. Lesser, “Cooperative Multiagent Systems: A Personal View of the State of the Art”. *Knowledge and Data Engineering* 11(1) (1999) 133–142.

[20] M. Paolucci, D. Kalp, A. Pannu, O. Shehory, K. Sycara, “A Planning Component for RETSINA Agents”. *Agent Theories, Architectures, and Languages* (1999) 147–161.

[21] M. Paolucci, O. Shehory, and K. Sycara, “Interleaving planning and execution in a multiagent teamplanning environment”. Technical Report CMU-RI-TR-00-01, Robotics Institute, CMU (2000).

[22] E. Pitoura, G. Samaras, *Data Management for Mobile Computing*. Kluwer Academic Publishers (1998).

[23] M. Satyanarayanan, “Fundamental Challenges in Mobile Computing”. *Proc. of the 15th Annual ACM Symp.on Principles of Dist. Computing* (1996) 1–7.

[24] W3C Note: *XML Pipeline Definition Language (Version 1.0)* (2002).

[25] H. Waechter, A. Reuter, “The ConTract Model”. In: Elmagarmid, A. (ed.): *Database Transaction Models for Advanced Applications*. Morgan Kaufmann Publishers, San Mateo (1992) 219–263.

[26] J. Waldo, “The Jini architecture for network-centric computing”. *Comm. of the ACM* 42(7). ACM Press (1999) 76–82.

[27] *Workflow Management Coalition (WfMC): Workflow Management Coalition Terminology and Glossary*. Technical Report WFMC-TC-1011 3.0. Brussels (1999).