Report Number: WUCSE-2006-8

2006-01-01

Supporting Collaborative Behavior in MANETs using Workflows

Rohan Sen, Gregory Hackmann, Mart Haitjema, Gruia-Catalin Roman, and Gill

Groupware activities provide a powerful representation for many collaborative tasks. Today, the technologies that support typical groupware applications often assume a stable wired network infrastructure. The potential for collaboration in scenarios that lack this fixed infrastructure remains largely untapped. Such scenarios include activities on construction sites, wilderness exploration, disaster recovery, and rapid intervention teams. Communication in these scenarios can be supported using wireless ad hoc networks, an emerging technology whose full potential is yet to be understood and realized. In this paper, we consider the fundamental technical issues that need to be addressed in order to introduce groupware concepts... Read complete abstract on page 2.

Supporting Collaborative Behavior in MANETs using Workflows

Rohan Sen, Gregory Hackmann, Mart Haitjema, Gruia-Catalin Roman, and Gill

Complete Abstract:

Groupware activities provide a powerful representation for many collaborative tasks. Today, the technologies that support typical groupware applications often assume a stable wired network infrastructure. The potential for collaboration in scenarios that lack this fixed infrastructure remains largely untapped. Such scenarios include activities on construction sites, wilderness exploration, disaster recovery, and rapid intervention teams. Communication in these scenarios can be supported using wireless ad hoc networks, an emerging technology whose full potential is yet to be understood and realized. In this paper, we consider the fundamental technical issues that need to be addressed in order to introduce groupware concepts into mobile ad hoc networks. Starting with a simple workflow model, we examine the process of allocating its actions to physically-mobile agents in a manner that accommodates transient communication and runtime errors.

2006-8

Supporting Collaborative Behavior in MANETs using Workflows

Authors: Rohan Sen, Gregory Hackmann, Mart Haitjema, Gruia-Catalin Roman, Christopher Gill

Corresponding Author: rohan.sen@wustl.edu

Abstract: Groupware activities provide a powerful representation for many collaborative tasks. Today, the technologies that support typical groupware applications often assume a stable wired network infrastructure. The potential for collaboration in scenarios that lack this fixed infrastructure remains largely untapped. Such scenarios include activities on construction sites, wilderness exploration, disaster recovery, and rapid intervention teams. Communication in these scenarios can be supported using wireless ad hoc networks, an emerging technology whose full potential is yet to be understood and realized. In this paper, we consider the fundamental technical issues that need to be addressed in order to introduce groupware concepts into mobile ad hoc networks. Starting with a simple workflow model, we examine the process of allocating its actions to physically-mobile agents in a manner that accommodates transient communication and runtime errors.

Type of Report: Other

Supporting Collaborative Behavior in MANETs using Workflows

Rohan Sen, Gregory Hackmann, Mart Haitjema, Gruia-Catalin Roman, and Christopher Gill
Department of Computer Science and Engineering
Washington University in St. Louis
Campus Box 1045, One Brookings Drive, St. Louis, MO 63130, U.S.A.
{rohan.sen, ghackmann, mart.haitjema, roman, cdgill}@wustl.edu

Abstract

Groupware activities provide a powerful representation for many collaborative tasks. Today, the technologies that support typical groupware applications often assume a stable wired network infrastructure. The potential for collaboration in scenarios that lack this fixed infrastructure remains largely untapped. Such scenarios include activities on construction sites, wilderness exploration, disaster recovery, and rapid intervention teams. Communication in these scenarios can be supported using wireless ad hoc networks, an emerging technology whose full potential is yet to be understood and realized. In this paper, we consider the fundamental technical issues that need to be addressed in order to introduce groupware concepts into mobile ad hoc networks. Starting with a simple workflow model, we examine the process of allocating its actions to physically-mobile agents in a manner that accommodates transient communication and runtime errors.

1 Introduction

Workflow Management Systems (WfMS)s constitute a class of groupware that has been shown to be effective for coordinating the activities of groups of individuals that are working toward common goals. This effectiveness is evidenced by the formulation of standard workflow languages, e.g., WS-BPEL [4] and Wf-XML [8], and commercial support for these standards from vendors like IBM [7] and Oracle [5]. The academic community has also developed numerous workflow systems and models, e.g., YAWL [9], ADAPT$_{\text{flex}}$ [19], and OPENflow [13].

Most WfMSs today are targeted either for stable wired networks, or for nomadic wireless networks where interruptions are rare and the network topology remains stable for extended periods of time. However, there are many dynamic scenarios where such network models are either inappropriate or too restrictive. Consider for example a large scale disaster area where several agencies such as the police, fire department, paramedics, etc. must coordinate their activities to gain control of the situation and save as many people as possible. In such situations, a response plan encoded as a dynamic workflow can help coordinate the activities of potentially hundreds of first responders equipped with mobile devices. In such situations, cellular networks may be unavailable due to destruction of transmitters or overloading of the network capacity and there is no place where a centralized server can be placed. Hence, we must assume that any network that is available to the responders is a mobile ad hoc network (MANET) that is formed directly by the devices of personnel in the area.

A MANET is a dynamic network that is formed on-demand and relies on no external infrastructure. Since most devices making up a MANET are physically mobile, the network topology is very volatile and disconnections between hosts are commonplace. This introduces fresh challenges for the development of WfMSs that can survive and execute in such dynamic environments. The lack of a central server means that planning, allocation, and execution of workflow tasks must occur in a completely distributed fashion. Frequent disconnections may result in half-completed tasks or in completed results not being passed on. Error handling also becomes more complex, as nodes may not be able to communicate failure information to others in the group.

This paper is an initial investigation into the implications of ad hoc mobility for WfMSs. We have taken the essential features of the workflow model and have used them as the basis for our study. The contributions of this paper are: (1) an algorithm that allocates workflow tasks to individuals while taking into account their *mobility* as well as their individual capabilities and (2) an error handling model that uses a *local re-planning strategy* to recover from failures in workflow execution in a distributed manner. We have kept our presentation in this paper intentionally abstract rather than committing to a specific workflow language, so that the concepts and techniques presented in this paper are applicable to a broader set of WfMSs and are unaffected by idiosyncrasies of particular workflow languages and system implementations.

In Section 2 we describe other groupware approaches that have been realized in wired and nomadic networks, and identify limitations of these approaches in MANETS. In Section 3 we discuss fundamental concepts of mobile collaboration and identify *plans* as a suitable representation of workflows. Section 4 describes a graph model for plans and introduces the important concept of *allocation*. Section 5 presents our algorithm for allocating plans, and Section 6 describes how run-time errors can be handled. Finally, Section 7 summarizes the approach presented in this paper and describes additional areas of investigation that are needed to extend the results of this study.

2 Related Work

Previous work in groupware systems has focused on specific application domains. This is especially true of the systems historically used in enterprise settings. Traditional groupware systems like Lotus Notes [3] and Groove [2] are often deployed to support specific interactions, e.g., through shared whiteboards and file sharing, and are rarely flexible enough to accommodate entirely new kinds of applications. For example, though construction workers could use Lotus Notes to share a list of tasks that need to be completed on a job site, absent additional programming, they cannot use it to assign these tasks to qualified workers automatically or to enforce the order in which jobs are performed.

More recently, there has been a push towards general-purpose workflow systems, especially within the Web services community. WS-BPEL [4], Wf-XML [8], and WS-CDL [14] extend workflow execution to the Web services domain. These languages define industry-standard XML schemas that model business processes as workflows. WS-BPEL in particular has been widely adopted by commercial vendors such as IBM [7] and Oracle [5], as well as by open-source workflow engines such as Twister [6] and ActiveBPEL [1]. These efforts build on existing Web service technologies like WSDL which are designed for stable networks where temporal disconnections are not a concern. Hence, they are best-suited for the coordination of legacy Web services in wired networks.

Other workflow systems, such as Exotica/FMDC [10], DOORS [18], and ToxicFarm [12], focus on supporting workflow execution in the face of network disconnections. Clients automatically hoard copies of needed data from a centralized server before they disconnect from the network. When the clients reconnect to the network, the server merges any changes to this data. These systems often assume that any disconnections from the network are temporary: clients must eventually reconnect to the server with the results of their computations. Moreover, they do not leverage the potential for collaboration among clients which are disconnected from the central server but which are in communication with each other.

3 Fundamentals of Mobile Collaboration

A MANET, which is the target environment for our work, consists of physically mobile devices (*hosts*) that support one or more *agents*, the basic units of modularity and execution. The key challenges in developing software for MANETs stems from the fact that motions of hosts results in frequent disconnections, which can occur at inopportune moments. In addition, one cannot rely on any fixed, centralized resources and the decentralization of the resources makes resource allocation more complex. These challenges are especially relevant to any WfMS targeted to MANETs. Even a simple application such as a shared whiteboard must be re-engineered because there is no central server that can host the whiteboard. Similarly, mobility can result in a disconnection between collaborators or cause resources to become unavailable.

One option for overcoming these challenges involves the exploitation of knowledge of the mobility patterns of hosts. Under this scheme, hosts report their *motion profiles*, which give each host's location as a function of time. These profiles can be used to deduce intervals of time when communication is guaranteed between a particular pair of hosts (by virtue of them being in communication range of each other). Although this approach would not be suitable if hosts primarily moved in a random pattern, given that our work is targeted to scenarios where groups of hosts collaborate to achieve a single goal, it is plausible that in such situations, mobility of the hosts follow well-defined patterns, which can be captured, e.g., by examining information in personal schedules, popular routes of travel, etc.

We assume a setting where a team of individuals meets at the start of the day to work collaboratively on a complex task. The individual activities associated with the task, as well as the order in which they must be completed, is defined in a *plan*. In simplest terms, a plan is a workflow consisting of set of *actions* and edges that connect the actions, thereby imposing an ordering among the actions (a more precise description appears in Section 4).

Each individual on the team carries a mobile device upon which execute one or more agents. Each agent provides some functionality which it advertises in terms of *capabilities* and *performance attributes*. To use the terms of the service-oriented computing paradigm, an agent can be thought of as providing a service that could be used to fulfill the requirements of some activity within the plan. We assume that each agent has the motion profile of the host on which it is executing, which it exchanges freely with other agents using a gossiping protocol. The motion profiles collected by an agent are stored in a local *knowledge base*. A closed collection of agents is called a *group*. Once a group is defined, additional agents are not allowed to join, though agents in the group may leave (of their own volition or due to failure).

The remaining issue is that of *allocating* the actions in the workflow to suitable agents in the group. A pre-selected *group leader* runs a special agent that performs this allocation. Recall that all members of the group are initially co-located. Thus, the group leader has access to the capability and motion information of all agents in its group. The group leader runs an allocation algorithm which allocates each action in the plan to a particular agent keeping in mind not only the agents capabilities but also its motion pattern. The agents then disperse and begin execution. The next section describes the structure of the plan in more detail and subsequent sections describe details of the allocation process.

4 Plan Structure

The plan is collectively the set of actions and an ordering structure which dictates the sequence in which the actions must be carried out in order to achieve the objectives of the group. As was stated in the introduction, we intentionally use a simple and abstract notation which captures only the essential features of a workflow model so that we may initially explore the implications of mobility on this model, without reference to more complex features.

The basic component of a plan is an *action*. Every action has an *identifier* that is unique within any given plan. In addition to the unique identifier, each action has a *name*, which is a short English phrase used to describe the action, and is mainly used to enhance human readability of the plan. For example, a name for an action could be "Print Paper". The action specifies an *input vector* consisting of *input ports*. Each input port corresponds to a single incoming edge to the action and specifies the type of data that is transmitted over that edge. Similarly, an *output vector* consists of multiple *output ports* that specify the type of data for each outgoing edge from an action. The *action description* specifies the characteristics of an agent that can perform the action. In our model, this description is split into two parts (1) capability requirements which describe what capabilities an agent must have and (2) performance attributes which specify how well an agent is able to deliver its capabilities. The capabilities and attributes are specified as elements that obey an ontology, e.g., OWL-S [17]. It should be observed that in addition to the explicitly specified restrictions, there is an implicit one which requires that the agent chosen to perform the action must be able to take in the inputs to the action as its own input and produce outputs corresponding to the action's outgoing edges.

The *ordering structure* is responsible for specifying the order in which actions must be completed and the flow of data and notifications between them. The ordering structure is specified in the form of a directed graph, with each node in the graph corresponding to an action. In keeping with the current state of the art in workflow systems, the ordering structure is required to be a sequence or a lattice. Multiple possible final outcomes of the workflow lead to a tree structure (e.g., an injured person can be taken to one of many hospitals), but this can be transformed into a lattice by adding a dummy end node and adding edges from the original end nodes to the dummy end node.

The ordering structure supports multiple edges to and from a single action. Any action in the graph that has multiple edges emerging from it must specify the semantics of traversing those edges. AND semantics requires that the action must place data or notifications along all the edges emerging from it. OR semantics requires that data or notifications be placed on only one of the edges emerging from the action. Correspondingly, any action that has multiple incoming edges must also specify AND or OR semantics. AND semantics in this context means that the action must wait for all its inputs before proceeding with execution. OR semantics means that only one of the inputs is required for the action to proceed, though more than one available input is also acceptable. Multiple edges are not allowed between adjacent actions. The edges in the ordering structure must mirror the inputs and outputs of the actions they connect, i.e., there should be at least the same number of incoming and outgoing edges to an action as there are inputs and outputs for that action.

In addition to actions and the ordering structure, we introduce the notion of *allocation constraints*. These constraints must be obeyed by the allocation algorithm when assigning actions to specific agents. An example of an allocation constraints is that a group of actions must be allocated to the same agent, such as in the case of electing a supervisor from a group of workers, who must then provide periodic progress reports. The agent that performs the supervisor action must be the one that performs the report writing action. Another example of a constraint is when actions should not be allocated to the same agent such as in the case where three reviewers are needed for reviewing a paper. In this case, the same agent should not be allocated to do more than one review.

5 Workflow Allocation & Execution

The allocation process maps each action in the plan to an agent that is well-suited to carrying it out. There are several important properties that must be satisfied while making these allocations. First, actions must be allocated to agents that are capable of carrying them out. Second, agents cannot be allocated to carry out two actions simultaneously. Third,

Connection Profile	
a – b	1 . . . 10
a – c	0 . . . 5
a – d	0 . . . 10
a – g	5 . . . 10
b – e	0 . . . 5
b – f	0 . . . 5
e – g	3 . . . 10
f – g	4 . . . 5

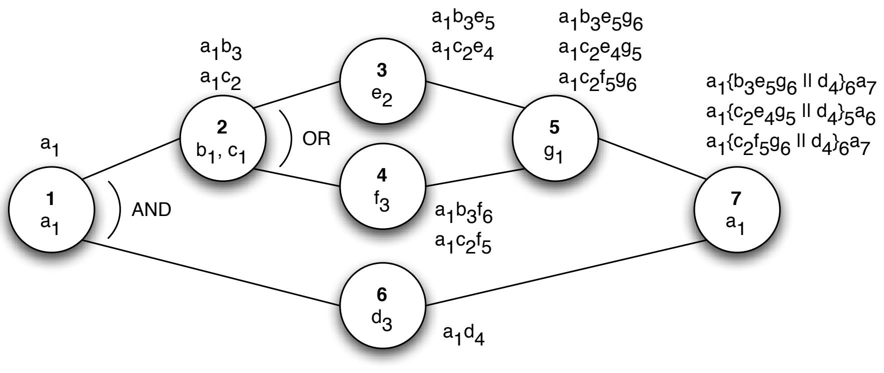

Figure 1. Sample Connection Profile and Annotated Plan Graph

the allocations must preserve the agent constraints specified within the workflow. Finally, agents must be connected to their predecessors and successors at the appropriate times, so that data may flow from one agent to another as needed.

The first issue is unaffected by mobility, so it can be solved using existing agent matchmaking schemes. For example, if agents advertise their capabilities and actions express their requirements using a uniform ontology, such as OWL-S [17], then matching actions to capable agents is straightforward [16]. Likewise, the second and third issues are fairly straightforward resource allocation problems which are unaffected by mobility.

However, the issue of agent connectivity is critical in adhoc environments, since it is greatly affected by agent mobility. We must not assign two sequential actions to agents which will never come into contact after the first action completes. Otherwise, the first agent will be unable to pass the result of its computation to the second agent, and hence the second agent will never be able to begin its own computation. If this scenario occurs at the wrong place in the execution (i.e., in a path that is not part of an OR lattice), then the entire workflow may fail.

We address this issue by leveraging the motion profiles described in Section 4, in order to derive the physical distance between any two agents at a given point in time, and hence determine when any two agents will be in communication range. We call this derivation the "connection profile". For example, according to the connection profile shown in Figure 1, agents a and b will be connected from times 1 to 10. For the purposes of this paper, we assume that each agent's motion profile is accurate, and that they are defined at least until the end of the plan's execution. In future work, we will accommodate inaccuracies in motion profiles, as well as situations where the motion profile may not be known far enough into the future.

Once the connection profile has been established, we can begin the allocation process. This process will annotate the workflow graph with all potential executions and their latest possible ending times. We begin the annotation process by marking the set of capable agents for each action, along with the time each agent will need to complete it. These annotations are shown in the center of each action in Figure 1.

We can now determine which paths may traverse each node in the graph. We represent these paths as ordered sequences that list the participating agents and their completion times. For example, the sequence "a_1b_3" defines a path containing two actions, where the first action is performed by agent a from times 0 to 1 and the second is performed by agent b from times 1 to 3. We can compute these sequences recursively as follows. If the node is not joining multiple branches of an AND lattice (e.g., any node in Figure 1 besides Node 7), then:

1. Collect the paths from all the predecessor actions. Select one agent (B) that can perform the current action.
2. For each path, compute its new ending time as follows:
   (a) Extract the last agent (A) from the sequence, along with the corresponding ending time (t).
   (b) Consult the connection profile to find the earliest time after t that A and B will be connected. If they will never be connected after time t, then discard this path.
   (c) Take the time of connection and add the time that B will need to perform the current action; call this time t'. Concatenate $B_{t'}$ to the end of the sequence.
3. Repeat this process for all collected paths and all agents.

For example, consider Node 5 in Figure 1. One of its predecessor's sequences is "$a_1c_2f_5$". Agents f and g will be connected immediately after f completes its computations at time 5. Since it will take 1 unit of time for agent g to perform the action, we end up with the sequence "$a_1c_2f_5g_6$". However, we discard the path "$a_1b_3f_6$", since f and g will never connect after time 6.

For nodes that join multiple branches on an AND lattice, we follow the same procedure with a few modifications. Instead of collecting each incoming sequence, we collect *combinations* of incoming sequences, using one sequence from each incoming edge on the graph. For each combination, we first extract the common prefix, i.e., the part corresponding to all the actions before the lattice split. We then collect the remaining portions of the sequences inside a pair of brackets, with parallel bars separating each parallel path through the lattice. We verify that B will connect to *all* of its predecessors after they complete, and let t be the latest time that any one input will arrive at B. Finally, we append this bracketed expression and t to the prefix, and treat the bracketed expression as a single path ending at t.

For example, at Node 7 in Figure 1, we collect the sequence $a_1 b_3 e_5 g_6$ from Node 5 and $a_1 d_4$ from Node 6. We extract the prefix a_1 and express the parallel paths as $\{b_3 e_5 g_6 \ || \ d_4\}$. a will be connected to d and g when they finish their computations, at times 4 and 6 respectively. Execution will begin after g's input arrives at time 6, and therefore end at time 7. So, we express the path as $a_1 \{b_3 e_5 g_6 \ || \ d_4\}_6 a_7$.

When this procedure has completed, we will have gathered at the last node a list of all feasible allocations and end times for the workflow's actions; we can also trivially compute their start times. Using this information, we can discard paths which allocate two actions to the same agent simultaneously, as well as paths which violate any agent constraints. Since this procedure is not affected by mobility, we can adopt existing mechanisms for this, such as those described in [11] and [15]. Any paths that are not discarded represent acceptable allocations; we can select one to execute non-deterministically, or using some prescribed policy.

Once the allocation algorithm has run to completion, the hosts that form the group are free to disband. The agent that is assigned to the first action in the plan begins executing while the agents assigned other actions do their own work while waiting for inputs from their predecessors. When an action is completed, the agent executing it notifies the agents performing the subsequent actions, passing along any necessary data. In the absence of errors, execution continues in this manner until the end of the plan is reached. Recall that since the plan is required to be a lattice, there is only one end state. The agent assigned to that end state is responsible for collecting the results and notifying other appropriate agents accordingly. In the next section, we discuss how we handle situations in which the execution does not proceed as planned.

6 Error Handling

Error handing is an integral part of most workflow languages. In mobile settings, error handling assumes added significance since the likelihood of errors in a MANET is much higher than in more stable wired or nomadic settings. The use of motion profiles helps to remove errors due to hosts moving out of communication range. However, other sources for errors still exist. For example, an agent may crash unexpectedly or resources may become permanently unavailable. In this section, we define the two types of errors we handle and discuss their impact on the plan definition and allocation strategy.

We divide errors into two broad categories: (1) *aborts* - errors from which it is possible to recover by re-allocating agents and (2) *failures* - errors which require a change in the plan structure. In the case of an abort, we simply backtrack to an appropriate point, re-allocate actions to alternate agents, and continue execution. In the case of failure, we backtrack to an appropriate point, and choose another branch in the plan, pruning the branch of the plan that caused the failure. In both cases, it is crucial to establish the point in the plan to which we must backtrack. To support this, we introduce the concept of a *failure zone*.

A failure zone is the set of actions within the plan that are affected by a particular error. Thus, if there is an error in one of the actions in the zone, the entire zone is affected and must be re-executed. Formally, a failure zone is defined as the smallest lattice structure in the graph that includes (recursively) the dependencies of the node that failed. Failure zones are parameterized by a label that is associated with a source of failure. When an error occurs, the affected area of the graph is determined by the failure zone parameterized with the same label as the factor that caused the error. For example, consider a scenario where two actions must be executed by the same agent. While executing one of those actions, the agent unexpectedly crashed. Now, both actions must be reassigned to another single agent. The bounds of the failure zone can be calculated using simple rules. If the dependencies are part of a lattice substructure within the graph, the delimiters are placed at the greatest lower bound and the least upper bound of the lattice substructure. If the dependencies are within a sequential structure, then the delimiters are placed at the first and last occurrence of a dependent node, respectively. Graphs which are a combination of these two basic structures, must first apply the rule for the lattice, and then the rule for sequential structure.

When an error occurs, the error type (abort or failure) and the cause of the error are propagated in both directions along the plan. Intermediate nodes forward the error if any child or parent of the node is within the failure zone and has not yet received the error notification. We designate the greatest lower bound preceding the relevant failure zone as the *re-planning node*. We cannot assign the lower bound of the failure zone as the re-planning node because it might itself be the node that caused the failure within the zone. The agent assigned to the re-planning node must perform a local re-allocation depending on the type of the error.

In the case of an abort, the agent at the re-planning node runs the same allocation algorithm that was run by the group leader in the beginning. However, since all agents are not likely to be in range during this re-allocation process, the re-planning agent uses the knowledge in its local knowledge base, which is typically a subset of the global knowledge. Thus, re-planning is likely to produce a sub-optimal allocation, but does not result in a complete failure if alternate suitable resources are available. In the case of failure, no re-planning is required. This is because a failure indicates that a particular path of workflow execution is infeasible. Delimiters for failures must be placed at branches with OR semantics so that alternate valid path for execution is available. The re-planning node simply resumes execution, except that this time it takes a different branch. The original branch is removed from the plan.

Any agents that are drafted to replace failed agents or that are part of the alternate path are notified of their new role *just-in-time* using the same notification system that is used to inform agents responsible for subsequent actions to begin execution or to pass data along, (i.e., just before an agent receives the inputs for its action, it receives a notice informing it of its new role within the overall plan execution.) This local re-planning can create a resource allocation problem if multiple errors exist simultaneously in the plan since two re-planners not in communication range can independently exploit an agent without knowledge of each others' actions. Due to constraints of space, we defer discussion of this problem to future work.

7 Conclusion

Collaboration frameworks for ad hoc mobile environments represent an exciting and relevant area of study, as evidenced by the many possible applications of such technology, e.g., disaster response, wilderness exploration, and construction management. In this paper, we adopted a simplified workflow model which has been very successful in stable wired networks and tried to understand how the model would need to evolve if ad hoc mobility were introduced. As a result of our study, we present an new allocation algorithm, that takes into account the motion pattern of agents as well as their capabilities when allocating actions to be executed by them. We also present an error handling model which uses a de-centralized local re-planning strategy to recover from errors during execution. The work presented here is a preliminary investigation in this area. Much work needs to be done to develop more optimized execution engines, error handling mechanisms, and potentially completely new collaboration models that are specifically geared towards mobile environments.

References

[1] ActiveBPEL engine. `http://www.activebpel.org/`.

[2] Groove virtual office. `http://www.groove.net/`.

[3] IBM Lotus Notes. `http://www.lotus.com/products/product4.nsf/wdocs/noteshomepage`.

[4] OASIS web services business process execution language (WSBPEL) TC. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel`.

[5] Oracle BPEL process manager. `http://www.oracle.com/technology/products/ias/bpel/index.html`.

[6] Twister. `http://www.smartcomps.org/twister/`.

[7] WebSphere process server. `http://www-306.ibm.com/software/integration/wps/`.

[8] Wf-XML 2.0. `http://www.wfmc.org/standards/wfxml_demo.htm`.

[9] W. M. P. v. d. Aalst and A. H. M. t. Hofstede. YAWL: Yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[10] G. Alonso, R. Gunthor, M. Kamath, D. Agrawal, A. E. Abbadi, and C. Mohan. Exotica/FMDC: Handling disconnected clients in a workflow management system. In *Proc. 3rd International Conference on Cooperative Information Systems (CoopIS)*, pages 99–110, Vienna, May 1995.

[11] J. Eder, E. Panagos, H. Pozewaunig, and M. Rabinovich. Time management in workflow systems. In *3rd International Conference on Business Information Systems*, pages 265–280. Springer Verlag, 1999.

[12] C. Godart, P. Molli, G. Oster, O. Perrin, H. Skaf-Molli, P. Ray, and F. Rabhi. The toxicfarm integrated cooperation framework for virtual teams. *Distributed and Parallel Databases*, 15(1):67–88, 2004.

[13] J. J. Halliday, S. K. Shrivastava, and S. M. Wheater. Flexible workflow management in the openflow system. In *EDOC '01: Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*, page 82, Washington, DC, USA, 2001. IEEE Computer Society.

[14] N. Kavantzas, D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto. Web services choreography description language version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, November 2005.

[15] H. Li, Y. Yang, and T. Y. Chen. Resource constraints analysis of workflow specifications. *J. Syst. Softw.*, 73(2):271–285, 2004.

[16] L. Li and I. Horrocks. A software framework for matchmaking based on semantic web technology. In *WWW '03: Proceedings of the 12th international conference on World Wide Web*, pages 331–339. ACM Press, 2003.

[17] D. Martin, M. Burstein, J. Hobbs, O. Lassila, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, B. Parsia, T. Payne, E. Sirin, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web services. `http://www.w3.org/Submission/OWL-S/`, November 2004.

[18] N. Preguiça, J. L. Martins, H. Domingos, and S. Duarte. Integrating synchronous and asynchronous interactions in groupware applications. *Lecture Notes in Computer Science*, 3706:89–104, 2005.

[19] M. Reichert and P. Dadam. A framework for dynamic changes in workflow management systems. In *Proc. of the 8th International Workshop on Database and Expert Systems Applications*, page 42. IEEE Computer Society, 1997.