Resource Disconnection Management in MANET Driven by Process Time Plan

Massimiliano de Leoni
Sapienza-Università di Roma,
Dipartimento di Informatica e Sistemistica
via Salaria 113 (2^{nd} floor)
I-00198 Roma, Italy
deleoni@dis.uniroma1.it

Fabio De Rosa
Sapienza-Università di Roma,
Dipartimento di Informatica e Sistemistica
via Salaria 113 (2^{nd} floor)
I-00198 Roma, Italy
derosa@dis.uniroma1.it

Schahram Dustdar
Vienna University of Technology, Distributed Systems Group (DSG),
Information Systems Institute
Argentinierstrasse 8/184-1
A-1040 Wien, Austria
dustdar@infosys.tuwien.ac.at

Massimo Mecella
Sapienza-Università di Roma,
Dipartimento di Informatica e Sistemistica
via Salaria 113 (2^{nd} floor)
I-00198 Roma, Italy
mecella@dis.uniroma1.it

ABSTRACT

The use of mobile ad hoc networks (MANETs) and of adaptive process management systems, able *(i)* to enact cooperative processes among the on-the-field operators and *(ii)* to cope with disconnection anomalies, is an effective tool in emergency management. In this paper, we present the architecture of MOBIDIS, a novel MANET-based process management system for emergencies, and we describe the novel techniques proposed for adaptiveness of the process enactment. A prototype implementation and experiments are presented.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; H.5 [**Software Engineering**]: Miscellaneous

Keywords

adaptive process management, mobile ad hoc network, emergency response, cooperative work, disaster analysis and management, homeland security-related application

1. INTRODUCTION

The widespread availability of network-enabled hand-held devices (e.g., PDAs with WiFi - the 802.11x-based standard - capabilities) has made the development of pervasive computing environments an emerging reality. Pervasive computing allows to build adaptive peer-to-peer software infrastructures for supporting coordination and management of processes in mobile and dynamic scenarios such as we have in emergency/disaster situations. Generally, in such scenarios, different teams belonging to different organizations need to collaborate for making disaster analysis. Each team member is equipped with hand-held devices (PDAs) and communication technologies, and should carry on specific tasks. In such a way, it is possible to see the whole team as carrying on a process. As an example, let us consider the following disaster analysis scenario. After an earthquake, a team (e.g., belonging to the Homeland Security Department) equipped with mobile devices (laptops and PDAs) is sent to the affected area to evaluate the state of specific sites. Their goal is to document the damage directly on a situation map, and to schedule following activities (e.g., evacuation for probable collapses, rebuilding jobs, and so on). Before recovery phase starts, the team leader stores all area details, including site map, list of the most important objects at the site, and some previous reports and materials on his/her personal assistant. All such details are provided by some back-end center of the Homeland Security Department, which assembled them by integrating information, knowledge and content stored by many other peer organizations (e.g., the Ministry of Internal Affairs, some basic spatial data provided by different public and private organizations, etc.).

The team constitutes a MANET in which the team leader's device coordinates the other team members devices[1] by providing appropriate information (for example, maps, important objects, and so on) and assigning activities. Mobile (or Multi-hop) Ad hoc NETworks (MANETs, [1]) are networks of mobile devices that communicate with one another via wireless links without relying on a wired infrastructure.

[1]Devices can be laptops (for specific requirements) or mostly PDAs

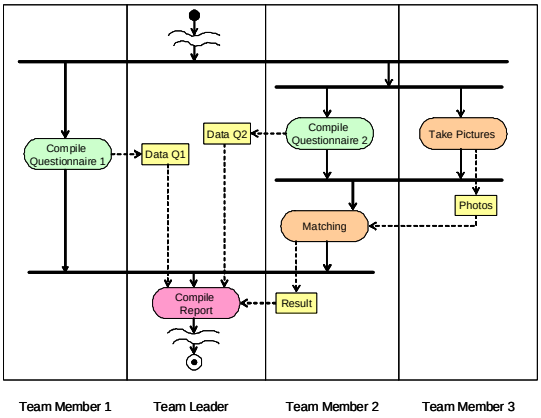


Figure 1: Process to be enacted.

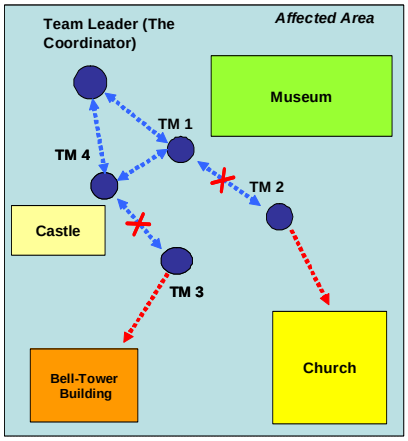


Figure 2: Resource disconnection.

To establish communication in a MANET, each device acts as an endpoint and as a router forwarding messages to devices within radio range. MANETs are a sound alternative to infrastructure-based networks whenever an infrastructure is no longer available, or cannot be used, as in emergency scenarios.

In Figure 1 is depicted a possible process we can have in emergency scenarios: after visual analysis of a building, supported by some map-based application, team member 1 (using her/his device, i.e., a PDA) fills out a report and enters attributes and data related to the damage. The team leader will analyze these reports and spatial data, with the help of specific software, to schedule the next activities. Team member 3 takes pictures of precarious buildings, whereas the team member 2 is in charge of image processing of older and recent photos of the site. The execution of this task should be done as soon as possible, as identification of architectural anomalies might highlight the danger of some collapse. In that case the time response of all team work is crucial. Nevertheless, the device/PDA with the high-resolution camera and the device/PDA with the older stored pictures must be connected in order that the image processing task can be carried out.

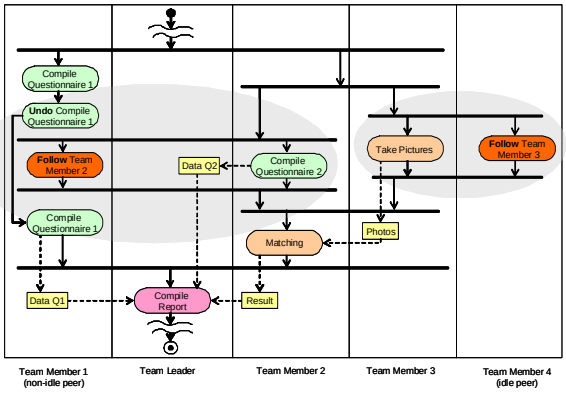


Figure 3: Modified process.

But in a scenario such as the one in Figure 2, the camera-equipped device/PDA's movement might result in its disconnection from other devices, giving situation of stalling during process execution. That could be unacceptable overall in situations where temporal constrains for task or process execution are needed, such as one above described.

Therefore, a pervasive architecture supporting process management in MANET contexts should be able to predict such disconnection situations in order to alert the coordination layer. The coordination layer, in turn, should direct a "bridge" device (team member 4's PDA) to follow the actor/PDA that is going out of range, maintaining the connection and ensuring a path between devices. In this way, the coordination layer, on the basis of a disconnection prediction, schedules the execution of new and not previously scheduled activities, as shown in Figure 3 (note the new activity for team member 4).

In this paper we focus on process management issues deriving from probable peer (resource) disconnections we have in MANET contexts, and we present a pervasive architecture suitable in emergency scenarios for process management in MANET, constituted by *(i)* a predictive layer for disconnection anomalies and *(ii)* an adaptive coordination layer able to carry out processes and change them when peer disconnections are raised. In particular this work proposes a *process model* for adaptive process management in cases of peer disconnections, and an algorithm for choosing a bridge based on a priority concept between tasks. Differently to [8], the proposed process model takes into account also time information such as expected duration for each task and statistically weighted values for each conditional branch. This information allows us to define time plans calculated starting from a *probabilistic timed graph* associated to process schema [4].

More in details the paper is organized as follows: in Section 2 some considerations on emergency management together with requirements and issues for process management on MANETs are discussed; starting from that, two transformation rules for adaptive process management are given. In following Section 3 we present our approach, named MOBIDIS[2], for adaptive process management on MANETs, whereas

[2]MOBIDIS: MOBile at Dipartimento di Informatica e Sis-

Section 4 presents the proposed collaborative process model and the bridge algorithm. In Section 5 related works are considered, and Section 6 presents some preliminary experimental results. Finally Section 7 concludes the paper by discussing future work.

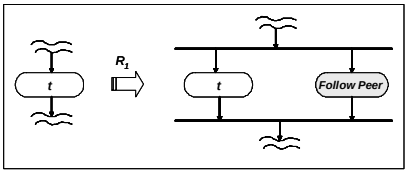
2. RESOURCE DISCONNECTION MANAGEMENT

In MANET scenarios, actors (peers) have to exchange information (i.e., *objects*) needed to carry out assigned tasks. Such an exchange may occur only if peers are connected to network. For instance, in the presented collaborative scenario a path has to exist between the peer having the high-resolution camera and the one containing the stored pictures in order that the former can send the latter new images. In these cases, if the two nodes are going to disconnect, the process could be not terminated (by lacking input data for some task) leading to situation of stalling in the system; that could be unacceptable overall when temporal constraints on task and process execution are required. "Bridging" actions may be used in process management to avoid those situations, and to support time plans for process execution [4]. They can be seen as adding new tasks in process instances to support other ones which, otherwise, could not be carried out. We term new added tasks as "supporting tasks" and process tasks defined by designer as "primary tasks", e.g.: the task "follow the Team Member 3", in the Figure 3, is a supporting task for the primary one "Send Photos".

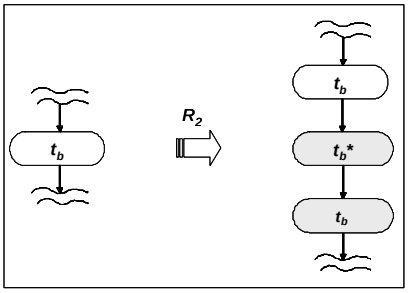
Supporting tasks may be added in parallel or in sequence with primary ones during the process execution, leading to *ad-hoc changes*. The particular arrangement adopted for the process restructuring depends strongly on the bridge choice. Let X be the peer who is going to disconnect, and let t be the task that peer X is going to carry out. If the choice is an idle peer (i.e., actually it is not executing task) then the process manager assigns to it supporting task such as "Follow X". Such an operation requires a change in the process instance by adding one task "Follow peer" (in our case peer X) in parallel with the task t, as depicted in Figure 4.(a). We term this rule R_1. In cases, instead, where the chosen bridge is carrying out a task t_b, the process management system could apply additional restructuring operations in the process instance besides those ones required by rule R_1. Indeed, if the task t_b can be compensated then the process manager introduces in sequence the corresponding recovery task t_b^* (making the undo of t_b) and postpones the task t_b in the process. That transformation rule is depicted in Figure 4.(b), and we term it R_2.

In Figure 3 is reported the modified process of Figure 1 obtained by applying a combination of the two proposed rules: in the first case (disconnection of team member 2) both the rules R_1 and R_2 are applied, since the chosen bridge (team member 1) is performing a task ("Compile Questionnaire 1"). In the second case (disconnection of team member 3) only rule R_1 is applied, since the chosen bridge (team member 4) is an idle peer. Note that the two changes of the process are independent each other since the involved tasks ("Compile Questionnaire 1", "Compile Questionnaire 2", and "Take Pictures") are in parallel.

temistica (DIS), of the University of Rome "La Sapienza", Italy.



(a) Rule R_1 used in cases of idle peer.



(b) Rule R_2 used in cases of non-idle peer.

Figure 4: Transformation rules for process adaptation.

In choosing a bridge, several aspects have to be taken into account, such as: assignment of tasks to peers at any time, task execution time, dependencies between tasks[3], and network topology. Task assignment allows to distinguish between idle and non-idle peers as well as to know which tasks are executed by whom. Execution time, in particular task time plan (time intervals for possible task execution [4]), together with dependencies between tasks (expressed in quantitative manner through a positive number) are needed for knowing which tasks can be delayed (postponed) without (or at least to have lower probability of[4]) violating process time constraints. That information is necessary when all possible candidate bridges are executing a task t_i. Indeed, in those cases if $P(t_i)$ represents the execution time of process P that we obtain by postponing the performing of task t_i and δ is the process execution deadline, then the task t_k, such that $P(t_k) - \delta \leq P(t_i) - \delta$ for each considered task t_i, will be stopped (undone) and the assigned peer is chosen as bridge. Finally, topology network information – specifically, number of neighbours and distance between neighbours – is used to choose between more candidate bridges (peers with the lowest number of neighbours are preferred,

[3]In this work we focus principally on dependencies between tasks deriving from process control flow.

[4]A tuple (p, s, e) in the time plan associated to a task defines the probability p that no time-constrained is violated when the task starts and ends in the time-interval $[s..e]$. In some cases it might be necessary to execute the task in time interval with probability $p \neq 1.0$, which means that the task execution can produce constraint-violation with probability $1 - p$.

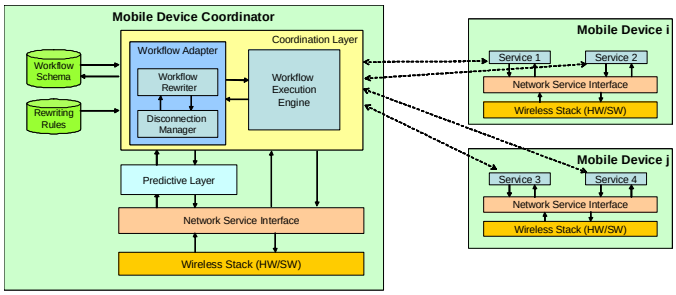


Figure 5: The MOBIDIS Architecture.

since the probability of new disconnections due to bridge movements is minimized) as well as to know the number of bridges needed to manage the disconnections. In Section 4 it is presented in more detail the algorithm for the bridge choice. It takes into account criteria above discussed.

3. MOBIDIS FRAMEWORK & APPROACH

In this section we give an overview on our approach, named MOBIDIS, for adaptive process management systems (PMSs) on MANETs. For more details, the reader is referred to [9].

With respect to resource disconnection management, the MOBIDIS approach combines local connection management among devices with global management of both network topology and task assignment. Local connection management consists of monitoring and checking one-hop communications between a device and its neighbours. It is realized as special services running on hand-held devices that implement techniques for estimating and calculating distances and relative positions (angle and direction of arrival) between a specific device and its direct neighbours. Global management maintains a consistent state of the network and of each peer in the network. It manages the network topology (and its predicted next states) and tasks each peer is in charge of, as well as services that peers offer (that is, it provides a service registry). On the basis of that information, the coordinator applies algorithms for choosing a bridge and/or executes process task reassignment when needed.

As regard process adaptation, MOBIDIS uses an ECA (Event / Condition / Action [2]) based approach to specify which events (peer disconnections) can produce probable process restructuring, and a set of transformation rules stating which control action have to be performed for process adaptation (i.e., which tasks have to be added, postponed, undone, etc.). Furthermore, a predictive approach is used to catch probable peer disconnection events and possible future service unavailability. Such a rule-based approach is highly flexible as rules are able to react on events at any time during process execution without making assumptions about when these events occur. This is in contrast to an approach based on adding conditional branches to process definitions [28], that cannot be proposed for very frequently changing environments such as ones of MANETs.

Figure 5 shows the proposed architecture: each device has a *wireless stack* consisting of a wireless network interface (the *wireless channel* with hardware for calculating distances from neighbours. On top, a *network service interface* [5, 6] offers to upper layers the basic services for sending and receiving messages (through multihop paths) to and from other devices, by abstracting over the specific routing protocols. Offered services (i.e., specific applications supporting tasks of human users) are accessible to other devices and can be coordinated and composed cooperatively. Some of these services are applications that do not require human intervention (for example, an image-processing utility). Others act as proxies for humans (for example, the service for instructing human users to follow a peer is a simple GUI that alerts the user by displaying a pop-up window on his or her device and emitting a signal).

In contrast, the coordinator device presents the *predictive layer* on top of the network service interface, signalling any probable disconnection to the upper *coordination layer*. The predictive layer implements a probabilistic technique [7] which can predict if all devices will still be connected in the next instant. The coordination layer manages situations when a peer is going to disconnect, by applying algorithms for choosing a bridge, and by executing process instance restructuring and process task reassignment when needed (e.g., it assigns the activity "Follow X" to the selected bridge). Specifically, the *Workflow Adapter* module is in charge of catching disconnection events incoming from the Predictive Layer and, on the basis of the current process execution state (taken from the *Workflow Execution Engine* module, which also is in charge of managing activity assignments), applies transformation rules, modifying the process instance of the cooperative work. In the next Section we describe the algorithms used in these phases.

4. COORDINATION LAYER

The coordination layer is in charge of enacting process instances taking into account peers jointed[5] in every instant to MANET and relative services offered by them. Specifically, the coordinator manages the current execution state of running process instance, in particular the state of every task constituting it; when a task is ready to be undertaken by some peer (i.e., the task preconditions are fulfilled), it performs a specific algorithm to select a peer of the network and assigns to it the task. When a disconnection event taking place, the coordinator executes the algorithm to select the corresponding bridge. Concurrently, it changes both the states of involved tasks and the assignment of tasks to peers; in addition, it modifies the process instance applying the relative restructuring rules.

The section is organized as follows: in subsection 4.1 we describe the process model and relative data structures used in the coordination layer. We then present in subsection 4.2 and subsection 4.3 the bridge algorithm and relative associated techniques and data structures, respectively, used to choose the bridge when disconnection events take place. Finally, in subsection 4.4 we report the algorithm used by coordinator to assign "primary tasks" to peers.

[5]When a peer wants join to team he/she contacts the coordinator and begins a joining procedure.

4.1 Process Model

In this work we have used directed graph as process schema language in modeling processes, and we have supposed that processes are structured [29] i.e.: each OR-split has a corresponding OR-join and each AND-split has a corresponding AND-join. In addition, the model contains the expected duration for each task and statistically weighted values for each conditional branch; that information can be established throughout process mining techniques and tools such as Caramba and TeamLog [10]. Therefore, each task can have multiple start-times and end-times due to conditional branches depending on the execution path. To represent probabilistic values of possible start-times and end-times of a task we use time plans calculated starting from a *probabilistic timed graph* associated to process schema [4]. A time plan TA on a task A is a set of tuples (p, s, e); each tuple (p, s, e) defines the probability p that no time-constrained is violated (deadline) when the task starts and ends in the time-interval $[s..e]$. Finally, in our model, each graph node represents a particular process construct, specifically: task, parallel routing, selective routing and N-join[6].

As far as **loops** within process schema, we refer to only ones named *structured cycles* [3], that is cycles can have only one entry point to the loop and one exit point from the loop and they cannot be interleaved.

Two special tasks, named *Start* and *End Task*, exist which represent, respectively, initial and final task of the process.

Beside to design-time information, deriving principally from the process structure and causal relationship among process tasks, the adaptive coordination layer needs additional information, taken at run-time, to enact processes. That information concerns overall: *(i)* the knowledge of the set of peers joined in every instant to the team (MANET), together with the set of roles each peer overlays, so to establish which peer can carry out which tasks; *(ii)* the state of every task constituting the process instance. A task can be in one of the following possible states [11]: *Not enabled*, *Enabled*, *Running* (or fired), and *Completed*.

The "bridging" actions needed to maintain the topology constraints can be seen as *supporting* tasks associated to primary ones, added or deleted during the execution of the process instance. Therefore, at run-time, it is possible to establish a relation between primary task and one or more tasks which are supporting it. That relation is modified by considering information available at design and run time, such as: the graph process; process execution state; assignment of task to resources, and peer disconnection events incoming from the Predictive layer. The support relation and its modifications together with task states represent the process instance restructuring. In [9] a more formal definition of the support relation is reported.

4.2 Bridge Choice Algorithm

The algorithm for choosing a bridge is used by coordinator when a peer is going to disconnect. It is based on the following criteria:

- Idle neighbours are preferred.
- If each neighbour is carrying out a task, then the one associated with the highest priority is chosen.
- With same priority values, the preferred bridge is one having the smallest number of neighbours. The bridge role likely leads to movement of the node and this might cause new disconnections. By selecting a node with the lowest number of neighbours, the probability of new disconnections is minimized.
- With same number of neighbours, it is preferred the nearest one.

The algorithm is as follows: let A be the set of peers constituting the MANET, and let t_a be the task assigned to peer $a \in A$. For each idle peer it is supposed to be assigned a null task (i.e., a task doing nothing) with priority value $\delta+1$ (i.e., $P(t_a) = \delta + 1$, see Section 4.3). Further, let $n(a)$ be the neighbours' set of peer a (i.e., $n(a) = \{b | b \in A \wedge b$ is in the radio range of $a\}$), and let d be the peer is going to disconnect. The procedure for computing the bridge is the following:

- Step 1: Compute $B = \{b | b \in n(d) \wedge P(t_b) = max\{P(t_a) | \forall a \in n(d)\}\}$;
- Step 2: Compute $B^{'} = \{b | b \in B \wedge |n(b)| = min\{|n(a)| : a \in B\}\}$;
- Step 3: Compute $B^{''} = \{b | b \in B^{'} \wedge r_d(b) = min\{r_d(a) : a \in B^{'}\}\}$; [7]
- Step 4: Select one peer $b \in B^{''}$.

4.3 Priority Algorithm

The goal of the priority algorithm is to assign a weight (priority) to each process task in order to know which tasks can be delayed (postponed) without violating process time constraints. It is used by coordinator when all candidate peers for bridge action are executing a task.

The proposed algorithm is based on both task time plan, obtained by applying the algorithm reported in [4], and the number of tasks depending by execution of involved task. Specifically: let $[\alpha_i..\beta_i]$ be a time interval in the time plan associated to task t_i with the maximum probability. Let $N(P)$ be the number of tasks constituting the process, and let $N(t_i)$ the number of depending tasks of task t_i. The priority of task t_i is calculated as follows:

$$P(t_i) \;=\; \beta_i(1 - \frac{N(t_i)}{N(P)}) \qquad (1)$$

It is possible to observe as:

[6] A N-join node is the corresponding for both OR-join and AND-join control flow constructs; it is a point within the process where two or more alternative process branches reconverge to a single common task

[7] $r_d(a)$ is the distance between the peers d and a.

- $0 \leq P(t_i) \leq \delta$, with δ the process deadline which can be calculated either as structural deadline [4] or in experimental manner.
- If a task t_i comes before a task t_j in the process control flow then the priority assigned to t_j is grater than one assigned to t_i, that is $P(t_i) < P(t_j)$. That directly derives from time plan and weight constructions (see *Note 3* in Appendix).
- When $N(t_i) = N(t_j)$ the priority value depends by β_i and β_j values as well as when $\beta_i = \beta_j$ the priority value depends by number of depending tasks.
- When $P(t_i) = P(t_j)$, we consider $P^{'}(t_i) = p \times P(t_i)$ and $P^{'}(t_j) = q \times P(t_j)$, with p the probability associated to time interval $[\alpha_i..\beta_i]$ and q the probability associated to time interval $[\alpha_j..\beta_j]$.

Algorithm for calculating the number of depending tasks.

The algorithm for computing the number of the depending tasks of a specific task is based on the property that a structured process can be considered as a composition of several sub-processes, in turn decomposable in other smaller ones (with respect to number of tasks making them up), and so on up to elementary processes, that is tasks. In addition, it is assumed that AND/OR split nodes have the same complexity than any task and thus they will be considered in the computation. This hypothesis is justified because more often a split operation (differently to join one) requires a non-zero computation time (i.e., it can require the computation of some operations).

The algorithm is constituted of two phases. In the first one, it is built a n-ary tree, named *process tree*, starting from the graph model representing the structured process. Each tree node is a process (elementary or not) whose children are nodes representing the sub-processes in which it can be decomposed in. More in detail:

- if (sub)process P is an elementary one (i.e., task), then the corresponding tree contains only one node (root) labelled with P (P corresponds to the task name).
- if (sub)process P can be divided in a sequence of two subprocess P_1 and P_2, then corresponding tree has a "sequence"-type node as root (labelled with P) and two children nodes labelled with P_1 and P_2, respectively.
- if the (sub)process P can be seen as parallel of many sub-processes P_1, P_2, ..., P_n, the corresponding tree has a "parallel"-type node as root (labelled with P), and n children nodes labelled with P_1, P_2, ..., P_n.
- if the (sub)process P is composed by a selection of several subprocess P_1, P_2, ..., P_n, then the corresponding tree has a "selective"-type node as root (labelled with P), and n children nodes labelled with P_1, P_2, ..., P_n.
- If the (sub)process P is a type-A loop (sub)process and n is the number of its iterations, then the corresponding tree has an "A loop"-type node as root (labelled with P) and n "sequence"-type node as children labelled with P_{1_1}, P_{1_2}, ..., P_{1_n}.
- If the (sub)process P is a type-B loop (sub)process and n is the number of its iterations, then the corresponding tree has an "B loop"-type node as root (labelled with P) and n "sequence"-type node as children labelled with P_{1_1}, P_{2_2}, P_{1_3}, P_{2_4}, ..., $P_{2_{n-1}}$, P_{1_n}.

Referring to the last two (sub)process cases, if it is not known the number of loop iterations but only the probability p to exit from one cycle and its next iteration, then it is possible to approximate n as $n = round(\frac{1}{p})$[8] (see [9]).

The algorithm for building the process tree starts from the whole process P by analyzing how P can be decomposed in P_1, P_2, ..., P_n sub-processes. According to possible decomposition, the corresponding initial process tree is built: a root node P and its children P_1, P_2, ..., P_n. For each subprocess P_i, if P_i is an elementary process, then no additional decomposition is done. Otherwise, the algorithm analyses how the sub-process P_i can be further decomposed. According to corresponding translation case, the node P_i is replaced by relative sub-tree. The algorithm stops when every leaf node is an elementary process (i.e., task).

In the second phase, the structure built in the first phase is used as basic information to compute the number of the depending tasks of a task. More in detail: following the decomposable characteristic of a structured process, we can recursively define the number of tasks ($P.Weight$) associated to a (sub)process P as follows:

- The number of tasks of a elementary process (task) is 1;
- The number of tasks of a sequence (sub)process composed by (sub)processes P_1 and P_2 is $P_1.Weight + P_2.Weight$;
- The number of tasks of a Parallel routing (sub)process composed by n branches is $1 + \sum_{i=1}^{n} P_i.Weight$, with P_i the (sub)process of the branch i;
- The number of tasks of a Selective routing (sub)process composed by n branches is $1+Max_{i=1,...,n}\{P_i.Weight\}$, with P_i the (sub)process of the branch i;

Starting from the process tree it is possible to assign a weight $N(\cdot)$ to each tree node P_i as follows:

$$N(P_i) = \begin{cases} 1 & \text{leaf} \\ N(P_{i_1}) + N(P_{i_2}) & \text{sequence} \\ 1 + \sum_{j=1}^{n} N(P_{i_j}) & \text{parallel} \\ 1 + Max_{j=1,...,n}\{N(P_{i_j})\} & \text{selective} \end{cases} \quad (2)$$

Finally, for each sequence-type node P_i if $N(P_{i_{left}}) < N(P_i)$ (with $P_{i_{left}}$ the left child node of P_i) then $N(P_{i_{left}}) =$

[8] p may be estimated by examining the execution of many process cases.

$N(P_{i_{left}}) + (N(P_i) - N(P_{i_{left}})$, and $N(P_k) = N(P_k) + (N(P_i) - N(P_{i_{left}}))$ for all node P_k belonging to the sub-tree whose root is $P_{i_{left}}$.

The leaf node weights so far obtained are the numbers of depending tasks of the task associated to leaf node. Note that if a task t_i comes before a task t_j in the process control flow then $N(t_i) > N(t_j)$.

4.4 Algorithm for Task Assignment

In this section we report the algorithm used by coordinator to assign "primary tasks" to MANET peers. "Supporting task" such as "Follow peer" are assigned to bridges in push manner (i.e., unconditionally) by coordination layer.

When a primary task changes its state from "not enabled" to "enabled", it is added to the *worklist* so to be undertaken by peers. Each task in the worklist (*work-item*) is marked by a timestamp representing the instant when it has been added to the list.

The assignment approach used in MOBIDIS is based on a Role-Based Allocation pattern[9] [12] for defining the range of resources that may execute the work item, and a combination of PUSH and PULL approaches [12] for offering and allocating tasks to peers. Specifically, when a peer a is ready to carry out a task, it looks up the worklist stored in the coordinator device. Let $R(a)$ be the set of roles overlayed by the peer a and W the current worklist upon the a's query. The coordinator returns a work-item set $SW = \{w \in W | R(w) \subseteq R(a)\}$ where $R(w)$ is the set of roles a peer has to overlay in order to carry out the work-item w. After the SW set is obtained, the peer can choose any task belonging to it (PULL procedure).

A work-item w cannot be stored in the worklist for an unlimited time. If a task w is stored in the worklist for a period greater than an α time, then the coordinator decides to allocate (push) it to a peer. That is done by using a protocol derived from the Contract Network Protocol [42] for task assignment in Multi-Agent System (MAS) [43]. The idea is to send a Call For Proposal (CFP) message to offer the task w to each free peer able to carry out it. A set of acknowledges is collected in a given time interval after which the coordinator chooses the peer accordingly to some execution metrics. When a peer sends a positive acknowledge, it stays on waiting for a positive (PUSH) or negative (NOPUSH) reply from coordinator and may pick up no task. That is, by positively acknowledging, peers subscribe a sort of contract imposing not to accept any task until they receive a positive or negative reply. Negative replies are sent not only to acknowledged peers but also to each peer in $A^{'}$ because there might be a peer which sent an acknowledge arriving to coordinator after timeout expiring. In that case, the peer is not in $A^{''}$ but, anyway, it is waiting for a reply.

5. RELATED WORK

The adaptation of process to possible exceptional cases or to changes in management policies has been soon recognised as a necessity for practical uses of process management systems [3, 16, 20]. Solutions to the related problem of dynamic change i.e., how to transform the process without suspending all its instances or waiting for all instances to have come to conclusion, of the possibility of creating inconsistent states of process instances have been studied in formal frameworks, typically defined by Petri nets. Examples of them are: WF-nets [13, 15], Flow Nets [16, 17], and MILANO nets [18, 19] (that is, marked, acyclic Free-Choice Petri Nets).

Moreover, adaptations of single process instances become necessary when exceptional situations occur or the structure of a workflow process dynamically evolves. More often, necessary changes and their scope for process evolution are known at design time. Examples of systems supporting such kinds of changes are: DYNAMITE [23], EPOS [24] and AgenWork [25]. Consequently, respective adaptations can be *pre-planned* and *automated*. In contrast *ad-hoc* changes have to be applied as response to unforeseen exceptions [26]. Relevant work on ad-hoc change in process management are ADEPT [26], Breeze [27] and $WASA_2$ [30] approaches. In them, authors address issues of manually modifying process instances in order to insure that none of the guarantees which have been achieved by formal checks at build time are violated. A detailed discussion of all these approaches can be found in [9, 22].

The MOBIDIS framework uses an ECA (Event / Condition / Action [2]) based approach to specify which events (peer disconnections) can produce probable process restructuring, and a set of transformation rules stating which control action have to be performed for process adaptation (i.e., which tasks have to be added, postponed, and undone). Differently to ad-hoc approaches present in literature, MOBIDIS supports an automatic and non-preplanned process adaptation due to the unforeseen and very frequently disconnection events we can have in MANET contexts.

In recent years, research in the MANET area has mainly focused on the development of appropriate routing protocols, security and reliability of the communications, methods for energy preservation, and other issues on the lower four ISO/OSI layers [35–39, 41]. Effective routing in ad hoc networks is still an actively-addressed open problem [33, 35, 37], with some interesting proposals presented in the literature (e.g., Dynamic Source Routing - DSR, Ad hoc On demand Distance Vector - AODV routing, Zone Routing Protocol - Z-RP, etc.). Researchers in this area assert that a sound technical basis for MANETs exists and it is thus time to start thinking about how to support applications based on MANETs. In order to enable the development of application layer software (and thus of any information system for MANET), abstractions on the specific characteristics of the routing algorithms and, more generally, on the services and data provided by the lower network layers, are required. [5] proposes a network service interface to be used as the basic layer on which to build application software, starting from the analysis and abstraction of current routing protocols.

6. EXPERIMENTAL RESULTS

This section illustrates the preliminary set of experiments we have considered to evaluate the MOBIDIS approach and framework. The goal has been that to determinate and ver-

[9] At design time it is specified that a task can only be executed by resources which correspond to a given role.

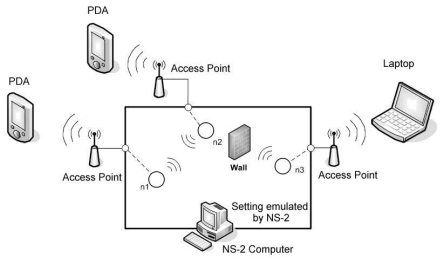


Figure 6: System Architecture for experiments.

ify the accuracy of the system in managing disconnections; note that a peer disconnection to the MANET could lead to resource unavailability which might generate stalling or deadlock situations in the system (e.g, the disconnection to the MANET of an unique peer equipped with photo-camera can yield stalling situation as no other MANET peer can execute the task "Take Pictures"). Therefore, the focus of these preliminary experiments has been that to evaluate the goodness of the bridge choice algorithm under simplified network circumstances. That is, in this phase, we have not considered network parameters such as control overhead and alike. These metrics will be taken into account in future experiments in order to better evaluate the proposed algorithm under more realistic situations.

The section is structured as follows: firstly, we give an overview on the system architecture we have set up for our experiments; secondly, we report on the preliminary experiments with relative results[10]

6.1 System Architecture

For our experiments we have set up a particular environment able to emulate moving of MANET peers according to a specific movement model. The emulator is deployed on a specific PC, and each peer device is connected to it. Each network peer has no knowledge about the existence of the emulator. When the software running on the peer device sends packets to another peer, it really sends data to emulator. If nodes are in the same radio-range then emulator forwards data to destination. Otherwise, packets are dropped. Peers in the emulated area can move towards a destination (a tower, a street and so on) according to a specific movement models implemented within the emulator. In Figure 6 is depicted the system architecture.

In realising our network emulator (named *Octopus Server*) [8], we have used a modified version of the network simulator NS-2 [44] with the *Magdeburg* patch [45] needed for establishing wireless emulations. Since NS-2 does not support any integration with external software, we have written a TCL[11] TCP/IP server. This server enables client software to be able to contact NS-2 via standard socket. As model of movement we have implemented the Voronoi mobility [40].

6.2 Tuning

The purpose of the first part of experiments has been that to tune some parameters of the algorithms realised in MOBIDIS framework in realistic situations. In particular, these parameters allow us to tune the reactivity of the system to disconnections. We recall that, doing that, we do not consider information about network overhead and alike. These will be taken into account in future experiments.

The first tuning metric is the **Polling Time**, i.e. the spent time in monitoring possible changes in network topology; lover value means more reactivity in doing corrective actions[12]. The second parameter is β, i.e. the fraction of the radio-range within the predictive technique [7]. As an example, in IEEE 802.11 with 100 meters radio-range, β equal to 0.3 means after 70 meters the prediction algorithm signals a probable disconnection.

Table 1: Experimental results.

β	**0.3**	**0.5**	**0.7**
polling time 3 sec	1%	0.09%	0.02%
polling time 5 sec	32%	4%	0,88%

The result of experiments is depicted in Table 6.2. In making that, we chose polling time between 3 and 5 seconds and β between 0.3 and 0.7. The polling time values have been chosen on the basis of the mean speed of a human walker, as well as the β values have been chosen to have stronger (0.7) or weaker (0.3) connections.

Reported results are the ratio (in percent) between the stalling time and total time (stalling time + execution time). Smaller values of β or greater polling time means lower, respectively, in number and in frequency of corrective actions, that is more freedom in moving. In the experiments, ratio decays with the decrease of the freedom. Little freedom brings to a greater number of rollback actions and then to inefficiency in process progress. The choice of β and polling time equal, respectively, 0.3 and 5 sec shows that stalling time is unacceptably around a third of total time. The results suggest that less often corrective actions are, more the stalling time is. A good tuning trade-off is polling time equal 3 seconds and *beta* between 0.3 and 0.5. In the following section we show some preliminary results obtained by considering such values.

6.3 Preliminary results

Our preliminary experiments concern how many predicted disconnections have been resolved. Specifically doing that, we wanted to evaluate the effectiveness of the bridge choice algorithm. The result is depicted in Figure 7; it shows the total number of disconnections in all experiments (the vertical axis). We can note that the number of disconnections resolved by bridging or task rolling-back (blue colour) is invariant with respect to the chosen values for beta (0.3 and 0.5), and closely to half of disconnections are correctly han-

[10]For more details about this section, the reader is referred to [8], in particular to the appendix of the paper which is available at the site `http://www.dis.uniroma1.it/~deleoni/documents/AppendixDMC2006.pdf`.

[11]TCL is the scripting language which NS-2 uses to configure emulations and simulations

[12]A corrective action is the application of one transformation rule, i.e.: R_1 or R_1.

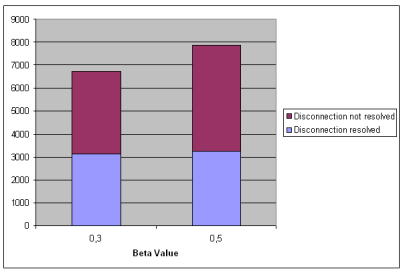


Figure 7: The total number of managed and non-managed disconnections.

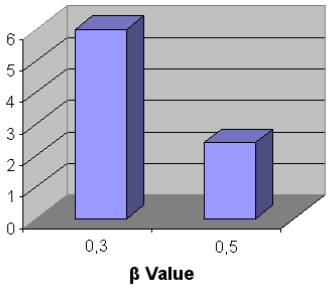


Figure 8: The mean value of the obtained connected components.

dled (the other half has not been managed because of disconnection events have not been captured).

The goodness of combination of both prediction and bridge algorithm is confirmed by another set of experiments, summarized in Figure 8. This set analyses the number of average connected components created during experiments. Note that the best situation is one connected component (all resource are connected in the network). For $\beta = 0.5$ the mean value of created connected components is just over 2; that implies that less than 10 nodes goes out of MANET range in a whole process performance.

6.4 Future Experiments

For a more accurate evaluation of our approach, we are going to make another set of experiments. Firstly, we will consider network parameters such as control overhead and alike in order to better evaluate the effectiveness of the proposed bridge algorithm; secondly, more effort will be devoted to the resource management: in this direction, we will use a resource distribution r over network peers in order to model situations in which some peers can carry out more process tasks. For instance, if 100 is the number of process tasks, and p is a MANET peer then with $r(p) = 20$ we indicate that p can carry out the 20% of process tasks. Thus, the experiment plan will be the following: for each fixed number of peers and considered resource distributions, we will execute 100 cases for each defined process schema (we will use 10 process schemas having both basic and complex constructs i.e., AND/OR split, and loops): the first time without any supporting actions; the second time with supporting actions. From the obtained results, it will be interesting to analyse the number of terminated processes and the time differences between process deadlines and the real time when processes are terminated. Finally, during our experiments, we will consider scenarios in which some points of the hit area are insecure. That will allow us to know how the system reacts to situations in which peers are unavailable (i.e., impossibility in moving) due to the around circumstances.

7. CONCLUSION AND FUTURE WORK

In this paper we have presented a pervasive architecture suitable in emergency scenarios for workflow management on MANETs, that through *(i)* a basic predictive layer for disconnection anomalies and *(ii)* an adaptive coordination layer able to change the process schema when disconnection anomalies are raised. We discussed the basic techniques developed and some preliminary experimental results validating our approach.

We also plan to evolve the coordination layer from a centralized to a distributed one (i.e., having a subset of devices act as coordinators). At the moment, the centralized architecture might be a bottleneck, but the current dimensions of a typical MANET for the considered scenarios (tens of devices) don't pose critical scalability issues.

Nevertheless, our results are based on synthetic data, and thus are only a preliminary validation of our approach. Future work will be devoted to enhance our techniques and to validate our approach in the context of some research projects we are currently involved[13].

8. REFERENCES

[1] D.P. Agrawal, and Q.A. Zeng. Introduction to Wireless and Mobile Systems. *Thomson Brooks/Cole*, 2003.

[2] N. Paton (Ed.). Active Rules in Database Systems. *Springer*, Berlin, 1999.

[3] W.M.P. van der Aalst and K. van Hee. Workflow Management: Models, Methods, and Systems. *MIT Press*, 2001.

[4] J. Eder, H. Pichler, W. Gruber, and M. Ninaus. Personal Schedules for Workflow Systems. Proc. *BPM, LNCS 2678*, Springer-Verlag Berlin Heidelberg, 2003.

[5] F. De Rosa, V. Di Martino, L. Paglione, and M. Mecella. Mobile Adaptive Information Systems on MANET: What We Need as Basic Layer?. Proc. *IEEE MMIS'03*, 2003.

[6] F. De Rosa, and M. Mecella. Designing and Implementing a MANET Network Service Interface with Compact .NET on Pocket PC. Proc. *International Conference on .NET Technologies*, UNION Agency - Science Press, 2005.

[7] F. De Rosa, A. Malizia, and M. Mecella. Disconnection Prediction in Mobile Ad hoc Networks for Supporting Cooperative Work. *Pervasive Computing*, IEEE, Vol.4, N. 3, 2005.

[8] M. de Leoni, F. De Rosa, and M. Mecella. MOBIDIS: A Pervasive Architecture for Emergency Management. Proc. *DMC*, 2006.

[13] *MAIS* - `http://www.mais-project.it`, and the IST FP6 *WORKPAD* - `http://www.workpad-project.eu`

[9] F. De Rosa. *Adaptive process management in mobile and dynamic scenarios*. PhD thesis, SAPIENZA - Università di Roma, Department of Computer Science, Italy, 2007.

[10] S. Dustdar, T. Hoffmann, and W.M.P. van der Aalst. Mining of ad-hoc business processes with TeamLog. *DKE*, Elsevier, Vol. 55, 2005.

[11] Workflow Management Coalition. Terminology & Glossary. Document Number WFMC-TC-1011, Document Status - Issue 3.0, December 2005. www.wfmc.org

[12] N. Russell, W.M.P. van der Aalst, A.H.M. ter Hofstede, and D. Edmond. Workflow Resource Patterns: Identification, Representation and Tool Support. Proc. *CAiSE 2005*, LNCS 3520, Springer-Verlag Berlin Heidelberg, 2005.

[13] W.M.P. van der Aalst, M. Weske, and G. Wirtz. Advanced topics in workflow management: Issues, requirements, and solutions. *IJIDP*, Vol. 7(3), 2003.

[14] W.M.P. van der Aalst, A.H.M. ter Hofstede, and B. Kiepuszewski Fundamentals of control flow in workflows. *AI*, Springer-Verlag, Vol. 39, 2003.

[15] W.M.P. van der Aalst, and T. Basten Inheritance of Workflows: an approach to tackling problems related to change. *TCS*, Elsevier Science Publishers Ltd., Vol. 270, 2002.

[16] C. Ellis, K. Keddara, and G. Rozenberg Dynamic Change Within Workflow Systems. Proc. *COOCS*, ACM Press, 1995.

[17] C.A. Ellis, and K. Keddara. A workflow change is a workflow. Proc. *BPM*, LNCS, vol. 1806, 2000.

[18] A. Agostini, and G. De Michelis. Improving flexibility of workflow management systems. Proc.*BPM*, LNCS, vol. 1806, 2000.

[19] A. Agostini, and G. De Michelis. A light workflow management system using simple process models. Proc. *Int. J. Collab. Comp.*, M. Klein, C. Dellarocas, A. Bernstein (Eds.), Vol. 9 (34), 2000 (Special issue on adaptive workflow systems).

[20] L. Baresi, F. Casati, S. Castano, I. Mirbel, and B. Pernici. WIDE Workflow Development Methodology. Proc. *WACC*, 1999.

[21] F. Casati and M.C. Shan. Dynamic and Adaptive Composition of *e*-Services. *IS*, Vol 3(6), 2001.

[22] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems a survey. *DKE*, Elsevier, Vol. 50, 2004.

[23] P. Heimann, G. Joeris, C. Krapp, and B. Westfechtel. DYNAMITE: dynamic task nets for software process management. Proc. *ICSE*, Berlin, 1996.

[24] C. Liu, and R. Conradi. Automatic replanning of task networks for process model evolution. Proc. *ESEC*, 1993.

[25] R. Müller, U. Greiner, and E. Rahm. AGENTWORK: A Workflow-System Supporting Rule-Based Workflow Adaptation. *DKE*, Vol. 51(2), 2004.

[26] M. Reichert, and P. Dadam. $ADEPT_{flex}$ supporting dynamic changes of workflows without losing control. *JIIS*, Vol. 10, 1998.

[27] S. Sadiq, O. Marjanovic, and M.E. Orlowska. Managing change and time in dynamic workflow processes. *IJCIS*, Vol. 9 (12), 2000.

[28] S.W. Sadiq, W. Sadiq, and M.E. Orlowska. Pockets of flexibility in workflow specification. Proc. *ER*, Lecture Notes in Computer Science, vol. 2224, Springer, Berlin, 2001.

[29] B. Kiepuszewski, A. H. M. ter Hofstede, and C. J. Bussler. On Structured Workflow Modelling. Proc. *CAiSE 2000*, Springer-Verlag Berlin Heidelberg, LNCS 1789, 2000.

[30] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. Proc. *HICSS-34*, 2001.

[31] D.B. Johnson and D.A. Maltz. Dynamic Source Routing in Ad-hoc Wireless Networks. *MC*, Imielinski and Korth eds., Vol. 353, Kluwer Academic Publishers, 1996.

[32] N.B. Priyantha, A. Miu, H. Balakrishnan, and S. Teller. The Cricket Compass for Context-aware Mobile Applications. Proc. *ACM MOBICOM* Conference, 2001.

[33] R. Beraldi, and R. Baldoni. Unicast Routing Techniques for Mobile Ad Hoc Networks. in *The Handbook of Mobile Ad Hoc Networks*, CRC Press, 2002.

[34] D. Niculescu, and B. Nath. Error Characteristics of Ad hoc Positioning Systems (APS). Proc. *ACM MobiHoc* Conference, 2004.

[35] Nitin H. Vaidya. Mobile Ad Hoc Networks: Routing, MAC and Transport Issues. *Tutorial on Mobile Ad Hoc Networks*, http://www.crhc.uiuc.edu/ nhv, University of Illinois at Urbana-Champaign, USA, July 2006.

[36] T. Salonidis, and L. Tassiulas. Distributed Dynamic Scheduling For End-to-end Rate Guarantees In Wireless Ad Hoc Networks. Proc. *ACM MobiHoc*, 2005.

[37] J. Kong, X. Hong, Y. Yi, J. S. Park, J. Liu., and M. Gerla. A Secure Ad-hoc Routing Approach using Localized Self-healing Communities. Proc. *ACM MobiHoc*, 2005.

[38] K. Nahm, A. Helmy, and C. C .J. Kuo. TCP over Multihop 802.11 Networks: Issues and Performance Enhancement. Proc. *ACM MobiHoc*, 2005.

[39] W. Su, S.J. Lee, and M. Gerla. Mobility Prediction and Routing in Ad hoc Wireless Networks. *IJNM*, Vol. 11(1), 2001.

[40] A. Jardosh, E.M. BeldingRoyer, K.C. Almeroth, and S. Suri. Towards Realistic Mobility Models For Mobile Ad hoc Networks. Proc. *MobiCom*, 2003.

[41] L. Zhow, and Z. J. Haas Securing Ad Hoc Network. *IEEE Network*, 1999.

[42] R. G. Smith. The contract net protocol. *IEEE transaction on computers*, Vol. 29(12), 1980.

[43] J. Ferber. Multi-Agent Systems. *Addison-Wesley*, 1999.

[44] The network simulator NS-2, http://www.isi.edu/nsnam/ns

[45] D. Mahrenholz, and S. Ivanov. Real-Time Network Emulation with ns-2". University of Magdeburg, Germany.