DIPARTIMENTO DI INFORMATICA
E SISTEMISTICA ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

A Web Service-based Process-aware Information System for Smart Devices

Daniele Battista
Daniele Graziano
Valerio Franchi
Alessandro Russo
Massimiliano de Leoni
Massimo Mecella

Technical Report n. 5, 2009

A Web Service-based Process-aware Information System for Smart Devices

Daniele Battista, Daniele Graziano, Valerio Franchi, Alessandro Russo
Faculty of Computer Engineering
SAPIENZA - Università di Roma, Rome, Italy

Massimiliano de Leoni, Massimo Mecella
Dipartimento di Informatica e Sistemistica
SAPIENZA - Università di Roma
{deleoni,mecella}@dis.uniroma1.it

Abstract

Nowadays, process-aware information systems (PAISs) are widely used for the management of "administrative" processes characterized by clear and well-defined structures. Besides those scenarios, PAISs can be used also in mobile and pervasive scenarios, where process participants can be only equipped with smart devices, such as PDAs. None of existing PAISs can be entirely deployed on smart devices, making unfeasible its usage in highly mobile scenarios. The use of smart devices poses interesting issues, such as reduced power, small screen size and battery consumption. This paper illustrates a full-fledged operationalisation of a PAIS, namely ROME4EU, which relies on a mobile Web service middleware and a BPEL orchestration engine, where both client applications and server-side components are running on Windows Mobile PDAs.

Keywords: Process Management Systems, MANET, Web-Services, Services, Adaptability, Human-Computer Interaction

1 Introduction

Over the last decade there has been increasing interest in process-aware information systems (PAIS), also known as Process Management System or Workflow Management System. A PAIS is, according to the definition in [5], "a software that manages and executes operational processes involving people, applications, and information sources on the basis of process models". The elementary pieces of work are called *tasks*, e.g., "Approve travel request XYZ1234".

A PAIS is driven by some process model which defines the tasks comprised in the processes, their pre- and post-conditions. The process model defines the control flow, which defines what tasks have to be executed beforehand and afterwards. Indeed, tasks cannot be performed in any order; certain tasks can be executed only when other tasks have been already completed. Moreover, typically processes defines some variables which somehow routes the process execution. Indeed, according to the values of such variables (i.e., the state), some tasks may need to be executed several times, whereas others may be skipped because they are not required any longer.

At the heart of PAISs there exists an engine that manages the process routing and decides which tasks are enabled for execution by taking into account the control flow, the value of variables and other aspects. Once a task can be assigned, PAISs are also in charge of assigning it to proper participants; this step is performed by considering the participant "skills" required by single tasks. Indeed, a task will be assigned to those participants that provide all of the skills required.

Participants are provided with a client application, often named *task list handler*. It is aimed to receive notifications of task assignments. Participants can, then, use this application to pick from the list of assigned tasks which one to work on as next.

Nowadays, process-aware information systems (PAISs) are widely used for the management of "administrative" processes characterized by clear and well-defined structures. The usual processes in pervasive and mobile scenarios (such as emergency management, healthcare, etc.) are characterized for being as complex as typical business processes of banks and insurances and for involving teams of tenths of members. Therefore, the exploitation of PAISs to support the process enactment seems to be very helpful. For instance, let us consider a typical pervasive scenario: emergency management. Rescue operators are arranged in teams, which are sent to the affected area to assess it and to provide first-aid assistance to the involved people. Teams are headed by a leader coordinating the activity of the other members.

To our knowledge, most of current PAISs comes with engines and task-list handlers working only on desktop or laptop machines. As widely motivated in Section 2, team members cannot be equipped with laptops. That would reduce their possibility of moving in the surrounding area to execute the assigned tasks, and the movement is a key requirement in mobile and pervasive scenarios. Therefore, task-list handlers must be working on smart devices of all members and the engine must reside a certain device on the spot.

In the light of the above, we have developed a PAIS, namely ROME4EU (The Roman Orchestration Mobile Engine for Emergency Units), whose engine may reside on a MS-Windows PDA. The engine and Task Handlers are completely decoupled. Task Handlers are supposed to be installed on the devices of all team members and the engine can be hosted in any of these devices or even in an extra one. From this moment one we assume the most realistic case in which the team leader deploys both of components, being equipped with the most powerful devices. Modern PDAs are becoming increasingly powerful and, hence, able to execute complex applications.

As better detailed in Section 3, ROME4EU had to overtake interesting challenges to be actually working on smart devices in pervasive environments. Firstly, it takes into account that mobile networks provide reduced communication bandwidth and low reliability. Secondly, smart devices are battery operating and, thus, the engine has to deal with the issue of minimizing the power consumption in order to guarantee its continuous functioning for certain amount of hours. It is worthy mentioning that reduced screen sizes limit the amount of information which can be visualized at the same time; therefore, we had to carefully study how to position enough information all together on the screen.

Finally, pervasive and dynamic scenarios are characterized by being very instable. In such scenarios, unexpected events may happen, which break the initial condition and makes the executing processes unable to be carried on and terminate successfully. These unforeseen events are quite frequent and, hence, the process can often be invalidated. Therefore, ROME4EU is flexible enough to adapt the processes when the condition of the environment where processes change. ROME4EU is equipped with some sensors which enable to monitor the status of the surround environments, such as network coverage, devices' location, speed, distance or battery level. On the basis of such monitoring, ROME4EU learns when some exogenous events happen or are about to happen and adapts the process schema accordingly.

The paper is structured as follows. Section 2 is aimed to give a quick insight of the rationale why current available PAISs are failing when working in mobile and pervasive environments and, consequently, ROME4EU has been devised. Section 3 delineates the critical issues we had to consider during the design and developed, whereas Section 4 describes the ROME4EU's architecture. Section 5 aims at describing how the whole architecture has been tested. Since we are willing to use the ROME4EU Process-aware Information System on Mobile Ad-hoc Networks (MANETs), we had to devise a MANET layer and test it to learn the actual Quality-of-Service which can be provided. Indeed, the MANET layer QoS constrains somehow some design ROME4EU choices. Section 7 describes the methodology used to make sure that ROME4EU is really usable for final users and meets their requirements. Section 8 shows the use of ROME4EU for emergency management, whereas Section 9 discusses related works. Finally, Section 10 concludes the paper, sketching future work directions.

2 An Overall Insight

The use of Process-Aware Information Systems in pervasive scenarios is valuable and can be used to improve coordination and communication during pervasive process executions.

Firstly, these processes are mobile and pervasive, meaning that devices and operators are located in the area and need to move in the surrounding area for carrying out the process. If team members were equipped with laptops, their possibility of moving to perform tasks would be seriously reduced. We may neither assume to deploy on the spot only Task Handlers and leave the engine in team headquarters. Indeed, the communication between team members and the headquarter are either too slow (e.g., satellite) or may become unavailable during critical situations (e.g., UMTS antennas overused or fallen down after an earthquake). Consequently, all components, both at server and client side, must be available on the spot and connected through mobile networks, which are partially unreliable.

Secondly, these processes are highly critical and time demanding as well as they often need to be carried out within strictly specified deadlines; for instance, in emergency management scenarios, saving minutes could result in saving people's life. Therefore, it is unapplicable to use a pull mechanism for task assignment where PAIS assigns every task to all process participants qualified for it, letting them decide autonomously what task to execute as next. Consequently, the ROME4EU engine aims at improving the overall effectiveness of the process execution by assigning tasks to just one member and, vice versa, by assigning at most one task to members.

Finally, these processes are created in an ad-hoc manner upon the occurrence of certain events. These processes are designed starting from provided templates or simple textual guidelines on demand. In the light of that, these processes are used only once for the specific setting for which they were created; later, they will not be used anymore. Moreover, teams are sent to the area only in order to face one situation and, hence, they take part in only one process simultaneously.

We have developed ROME4EU, taking into account the aforementioned considerations. In ROME4EU, process schemas are defined in the form of Activity Diagrams enriched for describing all the different aspects: definition of tasks in term of pre- and post-conditions, the control and data flow, as well as the assignment of tasks to appropriate members. Every task gets associated to a set of conditions to hold in order that it can be assigned; conditions are defined on control and data flow (e.g., a previous task has to be finished, a variable needs to have values in a specific range, etc.). Of course, not every member is able to execute every task. Every task needs to be assigned to a certain member that provides certain capabilities. We model that by binding each and every task to a set of capabilities; in addition, every member declares to furnish certain capabilities. Considering the control and data flow, the ROME4EU engine assigns every task to a certain member providing all required capabilities.

Every member device (including the leader) deploys a Task Handler, which allows to join to the team and specify the capability that she can provide. Then, it stays waiting for notifications of task assignments. The next task to work on is, then, visualized on the screen; when the member is ready to start it, she picks it and, possibly, appropriate applications are started to support the task execution.

In sum, taking into account the considerations above, the ROME4EU task cycle, depicted in Figure 1, is specialized with respect to those of other PAIS [18]:

1. When all pre-conditions over data and control flow holds, the ROME4EU engine assigns the task to the team member that guarantees the highest effectiveness. The task moves to the *Assigned* state.

2. The Task Handler notifies to ROME4EU, when the corresponding member is willing to begin executing. The task moves to the *Running* state.

3. The member begins executing it, possibly invoking external applications.

4. When the task is completed, the Task Handler notifies to ROME4EU. The task move to the final state *Completed*.

3 Design Issues and ROME4EU Solutions

ROME4EU is aimed at being working on smart devices in very dynamic and mobile scenarios over a network partially unreliable. Therefore, it copes with some challenging issues, which can be divided in two groups. The first group is concerned with grasping mental attentions onto the system as little as possible since pervasive processes are really mentally stressing for participants. On the other hand, there exist some technological challenges which deal with reducing the resource use and consumptions. Indeed, smart devices are low profile and, hence, are characterized to get very limited computation powerful.

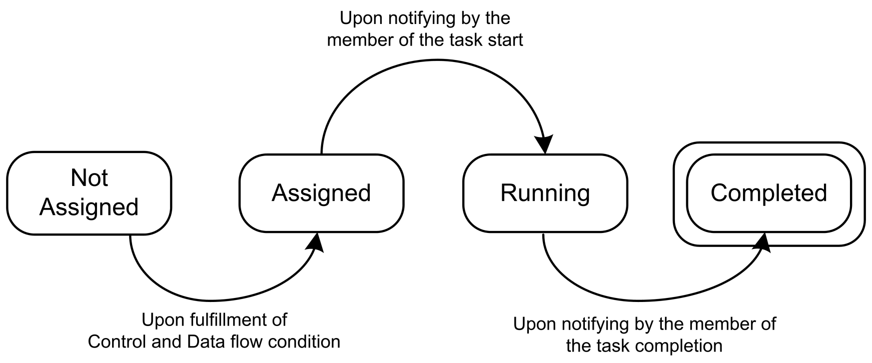

Upon notifying by the member of the task start
Not Assigned
Assigned
Running
Completed
Upon fulfillment of Control and Data flow condition
Upon notifying by the member of the task completion

Figure 1: The task cycle in ROME4EU.

3.1 Human-Computer Interaction Issues of Task Handler

The human being receives continually a huge quantity of stimuli from the environment. Book [20] defines attention as the totality of information cognitively manipulated by a person. The attention allows human being to consider stimuli in a judicious way, prioritizing them and taking into account only the most important ones. This judiciousness is used to increase the probability of a rapid and accurate answer. Activities in critical and pervasive scenarios are highly-stressing situations for users, who generally give more priority on the physical stimuli concerning the activities to execute than on those coming from software applications.

Therefore, we designed the Task Handler interface in order to grasp user attention only when strictly required. For instance, we have significantly made use of popups and sonorous alarms to achieve these results.

An aspect worthy to consider is accessibility and ergonomics when using PDAs in critical pervasive scenarios. Indeed, we have taken into account the fact that these devices may be used in extreme conditions. So, particular precautions must be taken when designing the UI. In particular the choice of colors should be effective and easy-to-read; they should be highly contrasting in order to be clearly visible in particular light conditions (e.g., in night missions). Moreover, the interaction with the interface takes mostly place through fingers, instead of the stylus. Therefore, the user interface elements should be sized and spaced out in order to avoid users to press on wrong elements because of the proximity of that users were willing to push.

3.2 Technical Issues

When devising the system we kept in mind to reduce as much as possible the use of three kinds of resources: the computational power, the communication bandwidth and the battery. Such resources are quite limited for smart devices in highly mobile and pervasive scenarios.

Smart devices, such as PDAs, are typically equipped with 400 Mhz CPUs, which are definitely slower than usual desktop machines. Therefore, ROME4EU avoids as much as possible busy-waiting where devices are continually listening for the occurrence of certain events. Asynchronous techniques and communication patterns (e.g., event-based or one-way SOAP-based invocations) are preferable in order to save CPU usage and battery consumption.

As already stated, in highly mobile and pervasive scenarios, team members are connected through mobile networks. Since such networks cannot rely on underlying infrastructures (because overused or unavailable), a good choice could be to make communicate members through MANETs, as also stated in [13]. A MANET is an IEEE 802.11x Wi-Fi technology where nodes are communicating with each other directly without any fixed access points: pairs of neighboring nodes (i.e., at direct communication range) can communicate directly. Not neighboring nodes can anyhow communicate, provided that there exists a path of nodes that can act as relays forwarding data packets towards the final destination. We performed some non-emulated experiments through NRLOLSR[1], a specific MANET implementation developed by U.S. Naval Research Lab (NRL). NRLOLSR is a research oriented OLSR protocol implementation, evolved from OLSR draft version 3. It is written in C++ according to an object oriented paradigm, and built on top of the NRL ProtoLib library [2], which guarantees system portability and cross-platform support.

ProtoLib supports a variety of platforms, including Linux, Windows, WinCE/PocketPC, MacOS, OpenZaurus, as well as the NS2 and Opnet simulation environments; it can also work with IPv6. It provides a system independent interface: NRLOLSR does not make any direct system calls to the device operating system. Timers, socket calls, route table management, address handling are all managed through ProtoLib calls. To work with WinCE, Protolib uses the RawEther component to handle at low level raw messages and get access to the network interface cards. The OLSR core code is used for all supported systems. Porting NRLOLSR to a new system only requires re-defining existing ProtoLib function calls.

As detailed in Section 6, we obtained a throughput of 70-80 Kbyte/s that is compatible with a lightweight Web service middleware for enabling the communication between the ROME4EU engine and the various Task Handlers.

As far as concerning the communication bandwidth, it is important to keep Web service invocations as light as possible. Indeed, it is anyhow limited as regards to modern Internet connections. Therefore, we designed

[1]`http://cs.itd.nrl.navy.mil/work/olsr/`

[2]`http://cs.itd.nrl.navy.mil/work/protolib/`

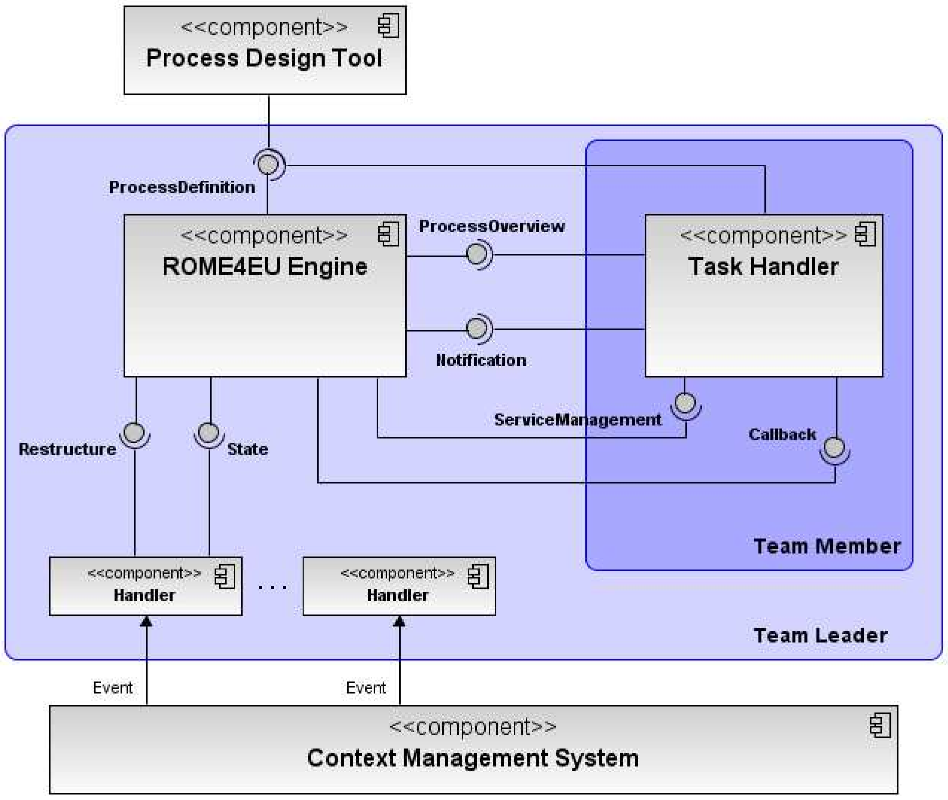


Figure 2: The overall ROME4EU's architecture.

the interfaces between Task Handlers and the engine in order to include only the input and output parameters strictly required. Moreover, whenever possible, we preferred simple types (e.g, integer, string, etc) rather than complex data transfer objects.

4 Software Architecture

The architecture of ROME4EU is depicted in Figure 2 and is basically composed by two main components: the ROME4EU *Engine*, and *Task Handler*. Team leader's device deploys both of components whereas generic members install only the *Task Handler*.

The *Engine* manages and coordinates the execution of complex processes representing missions to carry out. It performs task assignment and it manages and temporary stores information about team members involved in process execution, tasks to be completed and variables produced or modified during tasks execution.

The *Process Design Tool* is an external component used by the team leader during the initial planning stage. It allows the team leader to design the process to be executed (i.e. tasks and required capabilities) and to load it into the *Engine*.

As already stated, emergency management scenarios are highly dynamic and the environment is very instable and changing. In order to deal with unforeseen events, which may invalidate process execution, ROME4EU adopts an adaptability approach driven by context- and geo-awareness. Context data (like the GPS position or the battery status of team members' devices) are provided by the *Context Management System*, an external distributed application that, by means of some hardware/software sensors, monitors and retrieves information from the environment and makes them available through publish/subscribe techniques. In order to use context data to adapt process execution, ROME4EU is equipped with some specific event handlers, which are registered as subscribers to the *Context Management System.* An *Handler* is a software component that implements the logic required to manage a specific event. When an *Handler* is notified about the occurrence of an event (e.g. battery drops below 20%, device disconnects etc.) it performs a set of activities in order to adapt the process to the unexpected event. Each *Handler* is able to query the *Engine* to get information about process execution status. On the basis of such information and context data an *Handler* is able to restructure the process. Process restructuring is performed applying to the original process schema a set of adaptation patterns, which mainly require tasks insertions and/or deletions.

Process adaptation is thus achieved through a plug-in-based architectural approach. Managing a class of exogenous events requires a developer to implement the adaptation logic in a specific *Handler*, which receives notications from the *Context Management System* and interacts with the *Engine* in order to restrucure the process.

ROME4EU has been designed and developed according to Service Oriented Architecture (SOA) principles. Communication relies on an underlying middleware which hides behind Web service calls every critical issue and aspect concerning the communication.

The *Engine* provides five interfaces, which are developed as Web service endpoints and are accessed by *Task Handlers*, event *Handlers*, and by the external *Process Design Tool*:

ProcessDefinition allows to upload the process schema definition in XML format and to enact the process. Specifically, only the *Task Handler* of the team leader is authorized to start the process.

ProcessOverview is used to retrieve the process information, such as goals, members involved in the process and the status of the task that each member has got assigned (i.e., assigned, running, completed). Specifically, it will be used by *Task Handler* of the team leader which is allowed to show information about the status of team members (e.g., if they are running a certain task or if they have been assigned a task). The same interface is also used by every member to join to the team and specify the capabilities he provides.

Notification allows *Task Handlers* to notify the execution start of a given task as well as to signal the completion. Upon notification of the start of a certain task, the engine replies to the *Task Handler* of the member executing that task by returning the value and name of the variables that the member executing the task is authorized to read. Upon a task completion, the *Task Handler* of the executor returns to the engine the variables whose value has been updated by the task itself.

Status is used by the event *Handlers* to retrieve current process execution status, i.e. process schema, running tasks, team members' status and capabilities. Such information and data are used to determine wich adaptation patterns should be applied to restructure the process.

Restructure allows event *Handlers* to modify process schema according to specific adaptation patterns. Specifically, it allows to temporary stop process execution, add and/or remove tasks, rollback running tasks and restart process execution.

Task Handler provides two interfaces, which are also developed as Web services and accessed by the ROME4EU *Engine*:

Service Management is in charge of receiving notifications from the *Engine* about task assignments and the process conclusion. This interface is also used to notify the PAIS requests to roll back a task execution.

Callback is used as callback end point to respond to asynchronous one-way requests initiated by the *Task Handler*.

The internal modules of the *Task Handler* and *Engine* are communicating with each other only through further Web service endpoint interfaces in order to increase the flexibility and modularity of the whole system. For instance, it is possible to divide the modules of the PAIS *Engine* in two groups and to deploy each group on a different PDA in order to balance the *Engine* load on different devices.

Implementation details. The ROME4EU *Engine* and *Task Handler* have been developed in Microsoft Visual C# on the .NET Compact Framework for PDAs based on MS Windows Mobile, which is the de-facto standard for smart devices. The choice of the .NET Compact Framework rather than Java is motivated by the fact that .NET allows to control and tune more deeply the devices based on MS Windows Mobile. Moreover, as far as mobile devices, the .NET Compact Framework guarantees better performances than Java.

In order to develop a Web services middleware for hosting Web services on mobile devices, we started from a preexisting solution in Microsoft .NET C# [16], which we extended firstly to handle asynchronous requests/responses (i.e. one-way Web services invocations) and secondly to support complex data types. Indeed, allowing asynchronous communication is quite important in critical scenarios characterized by high mobility, where it is difficult and battery consuming to keep alive TCP connections (and, hence, SOAP ones) for long times. Complex data types simplify the notification to *Task Handlers* of task assignments and the task completion. Moreover, we improved the original XML and SOAP parser module, reducing memory consuption in order to manage big data sets (e.g. possible photos taken on the field) as input or output parameters in Web service calls.

On top of the Web service middleware we built a lightweight BPEL execution engine, which represents PAIS' core component. It is able to parse, execute and manage BPEL-compliant processes, hiding coordination overhead.

5 Emulation vs simulation and on-the-field test

In order to develop and test mobile applications, a complete development environment is required. As part of the development process, it is needed to study alternatives for design and implementation of software modules, analyze possible trade-offs and verify whether specific protocols, algorithms and applications actually work. There exist three way to perform analysis and tests: *(i)* simulation, *(ii)* emulation and *(iii)* on-the-field drills.

Simulation and emulation allow to perform several experiments in a cheaper and more manageable fashion than field tests. Simulator and emulator (i.e., hardware and/or software components enabling simulation or emulation) do not exclude each other. Simulation can be used at an earlier stage: it enables to test algorithms and evaluate their performance before starting actually implementing on real devices. Simulators allow for several kinds of hardware, through appropriate software modules (such as different device types, like PDAs or smart phones, or networks, like Ethernet or WLAN 802.11). Even if the application code written on top of simulators can be quickly written and performances easily evaluated, such a code must be throw out and rewritten when developers want to migrate on real devices.

The emulators' approach is quite different: during emulation, some software or hardware pieces are not real whereas others are exactly the ones on actual systems. All emulators (for instance, MS Virtual PC or device emulator in MS Visual Studio) share the same idea: software systems are not aware about working on an emulated layer (at all or partially). On the other hand, performance levels can be worse: operating systems running on

Microsoft Virtual PC work slower than on a real PC with the same characteristics. Anyway, software running on emulators can be deployed on actual systems with very few or no changes.

On the basis of such considerations, in order to develop a complete research environment for MANETs in emergency scenarios, we have designed and developed an emulator, named OCTOPUS [3]. Our emulator is intended to emulate small scale MANETs (10-20 nodes). Instead of making the whole MANET stack virtual, which would require duplication of a large amount of code, we decided to emulate only the physical MAC layer, leaving the rest of the stack untouched. OCTOPUS keeps a map of virtual areas that users can show and design by a GUI. Such a GUI enables users to put in that map virtual nodes and bind each one to a different real device. Further, users can add possible existing obstacles in a real scenario: ruins, walls, buildings.

The result is that real devices are unaware of OCTOPUS: they believe to send packets to destinations. Actually, all broadcasted packets are captured by OCTOPUS, playing the role of a gateway. The emulator analyzes the sender and the receiver and takes into account the distances of corresponding virtual nodes, the probability of losses as well as obstacles screening direct view[4]. On the basis of such information, it decides whether to deliver the packet to the receiver.

The virtual map, which OCTOPUS holds, allows users to insert obstacles representing walls, ruins and buildings. Virtual nodes are able to move into the map by passing around without going over such obstacles. Moreover OCTOPUS supports events which were not scheduled at design-time. Indeed, destinations of nodes are required to be defined at run-time, according to the behavior of client applications. Essentially, movements cannot be defined in a batch way; conversely, during emulations, nodes have to *interactively* inform the emulator about the movement towards given destinations.

The advantage of OCTOPUS is that, in any moment, developers can remove it and perform field MANET tests without any kind of change. As of our knowledge, OCTOPUS is the first MANET emulator enabling clients to *interactively* influence changes in the topology, upon firing of events which were not defined before the beginning of the emulation. Other emulators require the specification in batch mode, i.e., when the emulation is not yet started, of which and when events fire. In addition, OCTOPUS allows to include whichever kind of device, even PDAs or smartphones, and applications, whereas other approaches support only some platforms and applications coded in specific languages. Finally, OCTOPUS supports packet loss models (such as RayLeigh) and enhanced movement models, like Voronoi [9].

[3]Downloadable at: `http://www.dis.uniroma1.it/~deleoni/Octopus`. At the URL, the reader can also download a user manual.

[4]We assume whenever two nodes are not directly visible, every packet sent from one node to the other one is dropped.

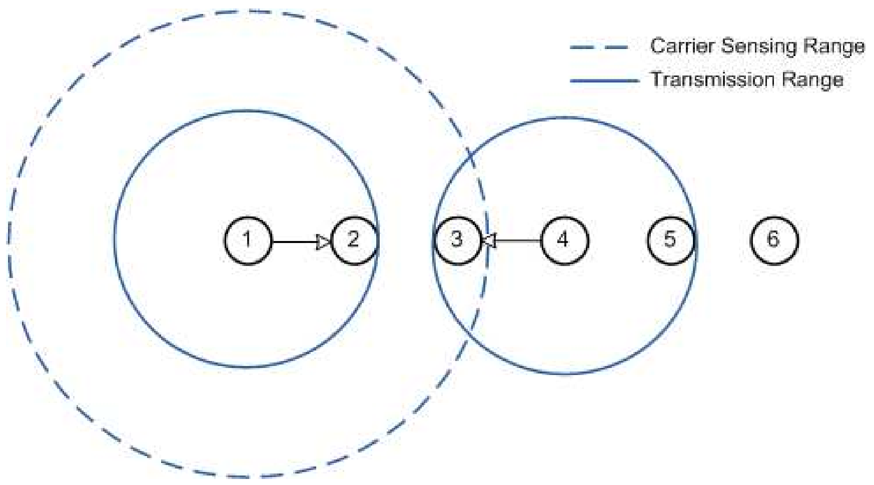


Figure 3: MAC interference among a chain of nodes. The solid-line circle denotes a node's valid transmission range. The dotted-line circle denotes a node's interference range. Node 1's transmission will corrupt the node 4's transmissions to node 3

6 QoS testing of the MANET layer

The actual feasibility of ROME4EU, the PAIS for smart devices, relies significantly on the QoS (e.g., the throughput) provided by the underlying MANET layer. In the light of that, we made some performance test of our MANET layer before actually starting devising the PAIS application meant to be used on top of it. Indeed, the MANET test results have definitely driven some choices in the design of the ROME4EU Process-aware Information System. It is worthy saying that basing QoS measurements only on simulated/theoretically-calculated values may lead up to build a system unable to work in real scenarios. We did not test through simulations, as many of other approaches do, since it would not return actual results. Conversely we performed emulation, by letting PDAs really exchange packets. Clearly on-field tests would be the better solution, but they require many people moving around in large areas and repeatability of the experiments would be compromised. Therefore emulation is considered an acceptable trade-off, in which the mobility is "synthetic" but the devices (and whatever running onto) are real (to be compared vs. simulation, in which both mobility and devices are synthetic). Nevertheless, we discovered and proved a relationship between laboratory and on-the-spot results, thus being able to derive on-the-spot performance levels from those got in the laboratory.

6.1 Deriving on-the-field tests from laboratory's ones

One of the most significant performed tests concerns the throughput in a chain of n links, when the first node is willing to communicate with the last.

In this chain, every node is placed at a maximum coverage distance from the previous and the next node in the chain, such as in Figure 3. In the shared air medium, any 802.11x compliant device cannot receive and/or send data in presence of an interference caused by another device which is already transmitting. From other studies (e.g., [12]) we know that every node is able to communicate only with the previous and the next, whereas it can interfere also with any other node located at a distance less or equal to the double of the maximum coverage distance. Therefore, if many devices are in twice the radio range, only one of them will be able to transmit data at once.

In our tests for the chain throughput, all devices are in the same laboratory room, which means they are in a medium sharing context. The chain topology is just emulated by OCTOPUS. Of course, having all devices in the laboratory, the level of interference is much higher than on the field; hence, the throughput gets a significant decrease. We have achieved a way to compute a theoretical on-field throughput for a chain from the result obtained in the laboratory.

Let $Q_{field}(n)$ be the throughput in a real field for a chain of n links (i.e., $n+1$ nodes). We are willing to define a method in order to compute it starting from laboratory-measured throughput $Q_{lab}(n)$. Here, we aim at finding a function $Conv(n)$, such that:

$$Q_{field}(n) = Conv(n) \cdot Q_{lab}(n) \tag{1}$$

in order to derive on-field performance. We rely on the following assumptions:

1. The first node in the chain wishes to communicate with the last one (e.g, by sending a file). The message is split into several packets, which pass one by one through all intermediate nodes in the chain.

2. Time is divided in slots. In the beginning of each slot all nodes, but the last one, try to forward to the following in the chain a packet, which slot by slot arrives at the last node.

3. Communications happen on the TCP/IP stack. Hence, every node that has not delivered a packet has to transmit it again.

4. The laboratory throughput $Q_{lab}(n) = \frac{\alpha}{n^{\beta}}$, for some values of α and β. This assumption is realistic as it complies several theoretical works, such as [12, 6].

We have proved the following statement:[5]

Statement. *Let us consider a chain formed by* $(n+1)$ *nodes connected through* n *links. On the basis of assumptions above, it holds*[6]*:*

$$Conv(n) = \left(\lfloor \frac{n}{3} \rfloor + 1\right)^{\frac{\beta}{2}} \tag{2}$$

6.2 Tests results

We investigate on two main kinds of tests: the performance of chain topology and some tests with moving devices.

Performance of the chain topology. The aim of this test is to get the maximum transfer rate on a chain. To obtain the measurements an application for Windows CE was built (using the .NET Compact Framework 2.0), which transfers a file from the first to the last node on top of TCP/IP, reporting the elapsed time.

All the devices run the NRLOLSR protocol implementation and use the routing protocol with the default settings and HELLO_INTERVAL set to 0.5 seconds. OCTOPUS emulates the chain topology and grabs all broadcast packets. When a node wants to communicate to another node, it sends packets directly to it if this is in his neighborhood, otherwise it sends them following the routing path. Both real and emulated devices were used; each reported value is the mean value of five test runs.

Figure 4 shows the throughput outcomes. The blue curve tracks the laboratory results; as stated in Section 6.1, we found through interpolation that the curve follows the trend $Q_{lab}(n) = \frac{\alpha}{n^{\beta}}$ where $\alpha = 385$ and $\beta = 1.21$. The green curve is the maximum theoretical throughput computed by Equation 2. We believe the actual throughput we can trust when developing applications is between the green and the blue curve.

Tests with moving devices. This kind of test aims to determine whether or not the NRLOLSR implementation is suitable for a real environment where nodes are often moving. Indeed, in a real field it is important not to break the communication among movements of nodes. If a team member is transmitting information to another team member, and nodes topology changes, all data must be delivered successfully, provided that the sender and the receiver are connected at all times through a multi-hop path, maybe changing over the time.

In order to emulate a setting of moving devices, we investigate three topologies, as shown in Figure 5, where the dashed line shows the trajectory followed by a moving device. Such topologies are designed in order to have *(i)* the moving node always connected at least another node, and *(ii)* each

[5]The proof of this statement is available as appendix of this paper.

[6]$\lfloor \cdot \rfloor$ denotes the truncation to the closest lower integer

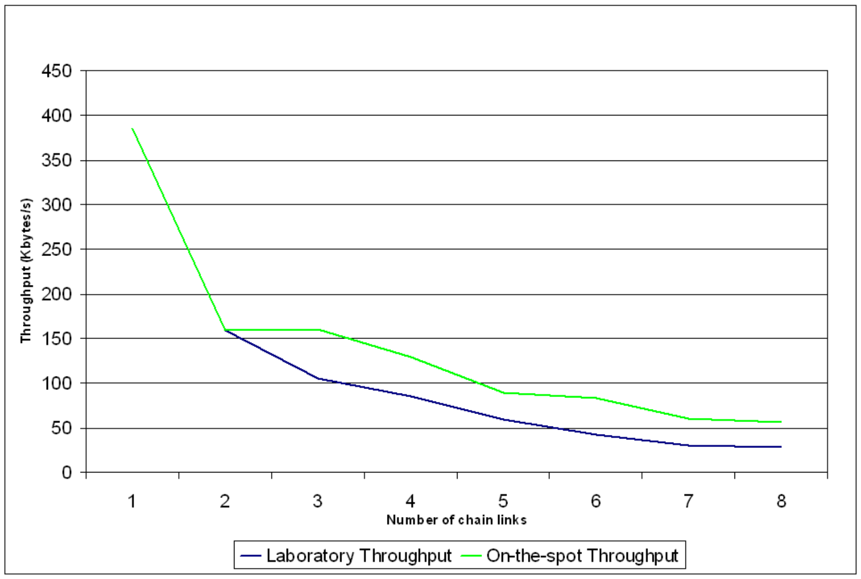


Figure 4: Test results for a chain-MANET, in the laboratory and estimated on-the-spot results

node is connected in some ways to at least another one, i.e., there are not disconnected node (no partitions in the MANET).

A WinCE application is used that continually sends 1000-byte longs TCP/IP packets between node S and node D. We tested every topology five times and every run was 300-seconds long.

Outcomes are demonstrated to be quite good, for every topology: during every run all data packets were correctly delivered to the destination. We experienced only some delays when the topologies were changing for a node movement. Indeed, while a new path is set up, data transmission incurs in 100% losses since the old path cannot be used for delivering. At application level, we are using reliable TCP and, hence, packets delivering is delayed since every single packet has to be transmitted again and again until the new path is built up.

TCP defines a timeout for retries; if a packet cannot be delivered by a certain time amount, an error is returned at application level and no attempts are going to be done anymore. In order not to incur in TCP timeouts, the node motion speed is crucial: if nodes are moving too fast, topologies are changing too frequently and, hence, the protocol is not reactive enough to keep routes updated. In the tested topologies, we have discovered that the maximum speed is around 18 m/s (65 km/h) such that TCP timers never expire.

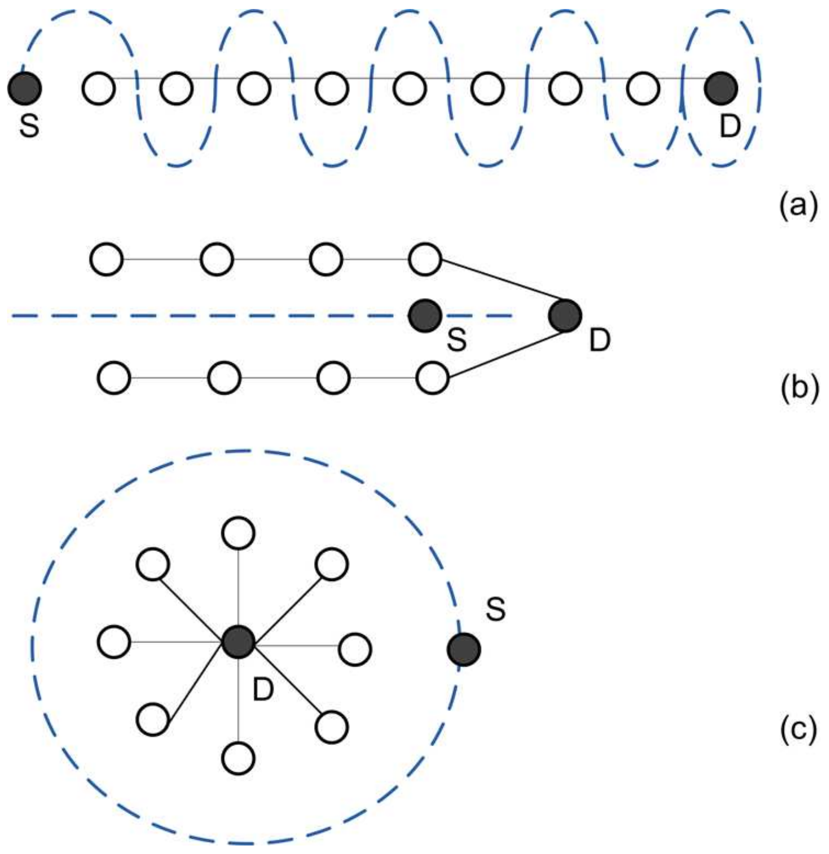


Figure 5: Dynamic topologies for testing TCP/IP disconnections

7 Evaluation and validation methodology

The ROME4EU system has been devised by keeping always in mind users and their requirements. Indeed, we use an evaluation and validation methodology which is named user-centered software development process [8]. At the heart of the evaluation there is always the user who is actively involved throughout the whole software development process.

We adopted a specific user-centered design (UCD) approach, in order to guarantee the usability of the software. Furthermore, requirements and following validation activities have been conducted by interviewing officers and generic actors from real emergency management organizations and applying ROME4EU software to Civil Protection of Calabria[7], Italy.

User requirements and any other relevant data related to the needs and expectations of users have been collected through interviews and questionnaires at the beginning of the design process. For the evaluation activities we mainly used qualitative usability evaluation methods like feature inspection, on-site observation of users while they perform different tasks, cooperative evaluation and questionnaires.

7.1 Usability tests and evaluations

Two usability tests with selected users from the Calabria Homeland Security Department have been performed. The results of the user tests were evaluated using qualitative evaluation methods and improvement recommendations were derived from the evaluation results. The tests took place with a selected number of users during the design phase of the software development process in order to get feedback from the users with regard to the following points:

- Test whether the requirements were understood correctly
- Test whether the main functionalities meet the users' demands
- Test whether the mock-up or prototype is easy understandable and usable
- Test whether specified tasks can be performed easily

In order to perform the user tests we used the following methods: qualitative online mock-ups and questionnaires, and cooperative evaluation accompanied by a semi-structured interview including a questionnaire.

The first usability test took place by means of mock-ups of the system's graphical components, i.e. Task Handlers. The test including the mock-ups was performed with an online questionnaire. The second usability test was

[7]http://www.protezionecivilecalabria.it/

performed with system's components prototypes. Cooperative evaluation test with users was performed with four users under the supervision of an evaluator guiding the users through the test. The user test was audio- and video-taped. After the user finished the tasks he was asked to watch the video and to comment it and the evaluator took notes of the user's comments. After looking through the video a semi-structured interview including a background questionnaire was performed with the users. The user test results were then analyzed in order to derive recommendation input for the system's development process.

7.2 Controlled experiments and on-field drills

Controlled experiments [21] test and compare user interfaces under controlled experimental conditions. In order to evaluate the system we are performing two identical on-field drills in a realistic setting at a simulated emergency site with emergency operators. In the first drill, which is going to take place at the end of 2008 in Calabria, operators will act as of today (i.e., without the support of the ROME4EU system). In the second drill, which will take place in April 2009, users (having the same user profile as the ones in the drill without ROME4EU) will be equipped with PDAs with the ROME4EU system and will perform the same tasks as the ones performed during the firs drill in order to guarantee comparability. The users' test activities will be videotaped so that a detailed analysis can be accomplished after the test. Test results of the drill with ROME4EU will then be compared to the ones obtained without the system. By this, conclusions can be drawn.

User-related evaluation metrics. We use the following evaluation metrics with regard to the two drills: efficiency, effectiveness, and user opinion.

Efficiency is defined as "doing the things right" [4]. The related measures are:

- The overall time needed to finish a task
- Mean time-on-task
- Ratio between completion rate and mean time-on-task
- Ratio between productive and unproductive time consumption (contains system response time)

Out of the video record of the usability testing session task time (i.e. how long it took the user to complete a task) and unproductive time (i.e. the time that it took the user to perform actions that did not contribute to the task output) can be measured and compared.

Effectiveness is defined as "doing the right things" [4]. The related measures are:

- Number of errors
- Completion rate
- Number of required assists
- Ratio between provided and required functions

Effectiveness can be measured also by counting and comparing the set implemented functions with the functions identified through user requirements analysis. The user should be provided with neither more nor less functions than the ones required.

User opinion is defined by measuring

- Satisfaction
- Usefulness
- Ease of use

from the user's point of view. We are going to measure user satisfaction with a questionnaire (based on a Likert scale) after the task performance and with an interview with open questions.

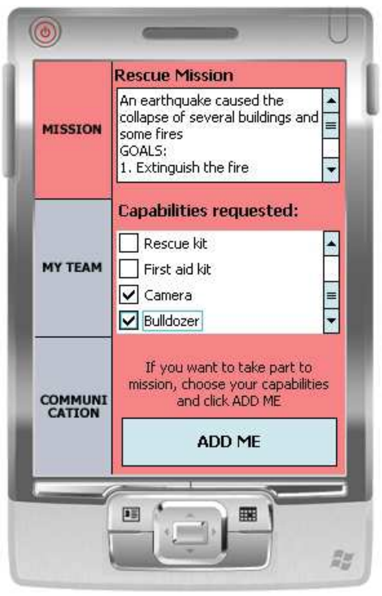
8 A Working Example: Emergency Management

In this section we are going to illustrate a number of features of ROME4EU by considering a potential scenario from emergency management. This scenario stems from a user requirement analysis conducted in the context of a European-funded project [1]. Teams are sent to an area to make an assessment of the aftermath of an earthquake. Team members are equipped with PDAs and their work is coordinated through ROME4EU.

For the scenario we have defined a process schema that can simulate a real emergency scenario, where several tasks have to be executed. The process schema foresees tasks which can be performed concurrently by different members.

In this example, we are envisioning four PDAs: the one of the team leader, namely Alessandro, and three generic members, Daniele, Alessio and Costantino.

The team leader device hosts the Task Handler and the engine where the generic members install only the former. Some capabilities are bound to specific external applications; these applications are meant to support the execution of the tasks requiring those capabilities. That means when a certain task is assigned to a certain member, since his Task Handler knows which capabilities are required for its performance, it automatically starts the proper applications associated to the required capabilities.



Rescue Mission

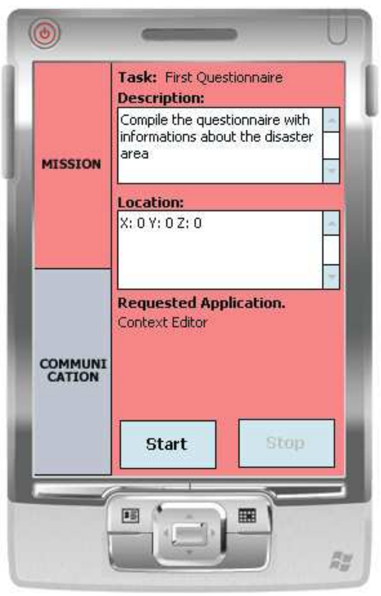
An earthquake caused the collapse of several buildings and some fires
GOALS:
1. Extinguish the fire

Capabilities requested:

- [ ] Rescue kit
- [ ] First aid kit
- [x] Camera
- [x] Bulldozer

If you want to take part to mission, choose your capabilities and click ADD ME

(a)



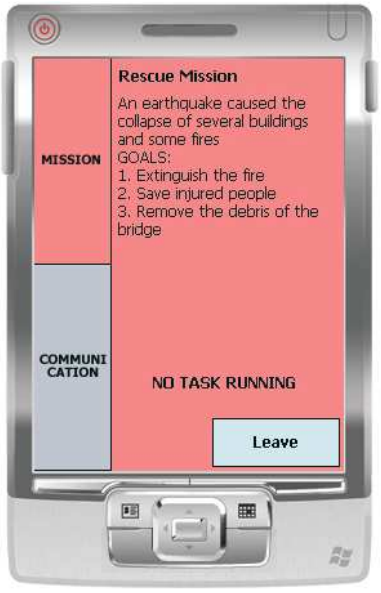
Task: First Questionnaire

Description:
Compile the questionnaire with informations about the disaster area

Location:
X: 0 Y: 0 Z: 0

Requested Application.
Context Editor

(b)

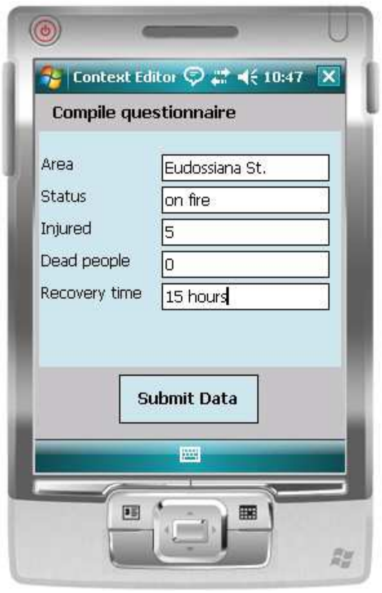


Rescue Mission

An earthquake caused the collapse of several buildings and some fires
GOALS:
1. Extinguish the fire
2. Save injured people
3. Remove the debris of the bridge

NO TASK RUNNING

(c)

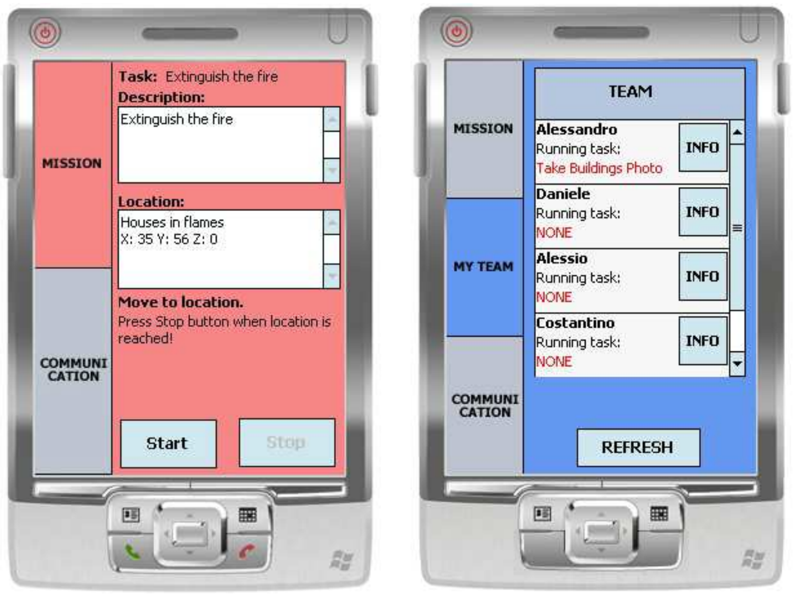


Compile questionnaire

Area	Eudossiana St.
Status	on fire
Injured	5
Dead people	0
Recovery time	15 hours

(d)

Figure 6: Some screen shots of Task Handler



Task: Extinguish the fire
Description:
Extinguish the fire
Location:
Houses in flames
X: 35 Y: 56 Z: 0
Move to location.
Press Stop button when location is reached!

Alessandro
Running task:
Take Buildings Photo
Daniele
Running task:
NONE
Alessio
Running task:
NONE
Costantino
Running task:
NONE

(a) (b)

Figure 7: Some screen shots of Task Handler (continued)

The process execution begins when the team leader loads the process inside the engine through his Task Handler; only the team leader is allowed to do that. At this stage, all members, generic and the leader, can join and select the capabilities they can cover. Figure 6(a) shows the screen shot of the Task Handler of the team leader while picking the capabilities she can cover from a check-box list. Indeed a member, in order to take part in a mission, has to cover some of the capabilities which are specifically defined.

The first process task is *Compile questionnaire about disaster area*, which should be assigned to a member providing capability *Text Editor*. Here, we assuming that only member *Daniele* provides this capability; therefore, the task is assigned to him (Figure 6(b)) and shown on his Task Handler.

When he is ready to start it, *Daniele* clicks on button *Start* and that causes firstly a notification message is sent to the Engine in background. Secondly, Task Handler starts the application that has been coupled to capability *Text Editor*. The *Text Editor* capability is associate to the application shown in Figure 6(d), which is meant to fill in a certain questionnaire for the assessment.

Generally speaking, when the Task Handler notifies the engine about the start of a certain task, the engine replies with the variables that can be read for the task execution. The definition of what variables can be read is given inside the process schema. After receiving the variables from the engine, Task Handler starts the applications associated to the task execution passing the received variable as input.

When *Daniele* completes the questionnaire, he closes the application and that is recognized as the completion of task *Compile Questionnaire*. The questionnaire data are sent back to the engine that stores them as a process variable (specifically as binary data).

While *Daniele* was executing that task, no more task has to be assigned to any member. Indeed, other tasks will be enabled when *Compile Questionnaire* is considered as completed. Therefore, the Task Handler of any other members is similar to 6(c).

After executing *Compile questionnaire*, the process splits in three concurrent branches. One is concerned with extinguishing a fire and the engine assigns it to member *Alessio*. When notified, *Alessio* may click on *Start* in order to begin the execution. This task is associated to no application, since it requires only a water pump to extinguish the fire occurred.

Figure 7(b) shows the actual implementation of the feature that allows the team leader to gain an insight of the status of all members. The status is shown in a table where each row refers to a certain member. Specifically, such rows show the (possible) task that every member is executing. By clicking on the INFO button associated to the row of a certain member, the team leader can obtain the history of tasks which that member executed in the past.

Space matters avoid us to place every screen shots more concerning the Task Handler.[8]

9 Related Work

Most of current Process-aware Information Systems cannot be completely deployed on smart devices, such as PDAs or smartphones. This holds both for open source products, e.g., jBPM[9] and Together Workflow[10], as for commercial systems, e.g., SAP Netweaver[11], Flower[12] or TIBCO's iProcess Suite[13]. In any case, a desktop or laptop machine is needed on which the engine has to be installed. Moreover, they do not consider other critical issues of mobile and pervasive scenarios, such as the battery consumption, the intrinsic slowness and unreliability of the mobile network as well as the reduced power of smart devices.

[8]A video of a system demonstration with 3 PDAs is available at `http://www.dis.uniroma1.it/~deleoni/video.wmv`

[9]jBPM web site - `http://www.jboss.com/products/jbpm`

[10]Together Workflow web site - `http://www.together.at/together/prod/tws/`

[11]Netweaver web site - `http://www.sap.com/usa/platform/netweaver`

[12]Flower web site - `http://global.pallas-athena.com/products/bpmflower_product/`

[13]iProcess Suite web site - `http://www.tibco.com/software/business_process_management/`

The recent literature provides some interesting developments of PAISs running on mobile devices. CiAN [19] is a language and middleware supporting collaboration in Mobile Ad-hoc Networks. The system is designed to run in a completely decentralized fashion with no need for a central coordinating entity. The CiAN middleware is responsible for executing processes specified using the CiAN language. It has been targeted to mobile networks and, hence, it addresses some of the typical issues of mobile environments. CiAN shows two important drawbacks. Firstly, the prototype has been implemented in Java using J2SE 5.0 and, consequently, it does not work on PDAs. Secondly, it does not provide enough flexibility to handle those exogenous events which bring the system in situations where processes cannot carried on and terminate successfully.

Another valuable implementation has been developed by the Nokia Group in Finland [17] and runs on smart devices. It claims future releases will be able to handle situations in which some resources disconnect from the mobile ad-hoc network and become unavailable. That is valuable but does not cover every possible situation caused by exogenous events. For instance, a device can get (or destined to become) unavailable for many other reasons, such as it runs out of battery.

WHAM [10] proposes a loosely couple PAIS which allows the off-line execution of tasks and the possibility of moving the engine among the available devices. But, conversely, it does not enable the run-time integration of arbitrary external services and applications. The process logic definition itself drives somehow how to integrate external services. Consequently, it is defined at design time how services can be orchestrated and what applications of process participants to invoke in order to perform tasks.

Moreover, there exist many systems for mobile networks and PDAs which aims at supporting the collaboration among different participants. The latest and most significant is Service-Oriented Mobile Unit (SOMU) [15]. This tool supports the data exchange and the collaboration to achieve information; it works also in a decentralized fashion with no central entities. But it does not provide the concept of processes to be carried out. As also the authors claim, it is meant as underlying infrastructure for collaboration on top of which other systems (like PAISs) provides the coordination. Indeed, coordination is something more than collaboration implies a plan (i.e., a process) to achieve such a collaboration. On the contrary, SOMU as well as other systems leaves freedom to let coordinate participants, and the freedom in highly dynamic and pervasive scenario is not a good way to proceed.

The BPEL4People and WSHumanTask standards [11] are currently under definition to extend BPEL processes to support activities performed by humans. As ROME4EU does, BPEL4People models every human as service covering some roles and the interaction with each service, including humans, is modeled as message exchanges using some communication protocols. Nevertheless BPEL4People is not yet standard and is still under debate [14]. Since its definition is still ongoing, the current implementations are only partially-fledged prototypes. Moreover, BPEL4People does not address explicitly the challenging issues of mobile Process-aware Information System. The most valuable implementation is VieBOP [7] and is meant to run only on desktop/laptop machines. BPEL4People has also got the drawback which does not describe how to assign tasks to process participants. Task assignment is a very important topic in many pervasive and mobile scenarios (such as Healthcare or emergency management) which should be carefully taken into account.

10 Conclusion

ROME4EU is a Process-aware Information System that can be completely deployed on smart devices, such as Windows-Mobile PDAs. The fact that it does not require laptops and desktops make feasible its use in pervasive and mobile scenarios, such as emergency management, domotics and healthcare. Indeed, as argued, every component has to be running on the spot and the possible use of laptops would limit the mobility of team members, which is a key point in aimed scenarios.

ROME4EU has been devised and developed taking always into account the user requirements. Indeed, we performed two interactive cycles so far during which we proposed our solution, they gave feedback on it by interviews and questionnaires and we adapted it accordingly. Specifically, in the context of a European-funded project, namely WORKPAD, we had the opportunity of making evaluate ROME4EU by typical operators, such as Civil Protection. In April 2009, we are going to have the final validation through a specific on-the-field drill where we will be emulating an earthquake and give the ROME4EU PAIS to such operators.

The current stable version of ROME4EU is just able to manage predefined sets of exogenous events. By default ROME4EU is equipped with some specific handlers to adapt the processes schema to the corresponding class of events. Specifically, we can deal with prediction of node disconnections, with running out of battery or space storage. But, basically, specific developers can plug into ROME4EU specific handlers for adapting the process schema to the corresponding class of exogenous events. On this concern, future work aims at being able to adapt process after the occurrence of any events, even those for which no specific handlers exist. The system itself should be able to understand what significant deviations occurred and what task sequence is needed to recover the process execution. The schema will be changed in the form that the recover tasks are put before the remaining part of process still to be executed. We have already proposed a general framework to enable adaptiveness [3] and developed a first prototype of the corresponding module[2]. Currently, we are working on integrating this module in the current ROME4EU version.

Acknowledgement. This work is supported by the European Commission through the FP6-2005-IST- 5-034749 project WORKPAD.

References

[1] T. Catarci, M. de Leoni, A. Marrella, M. Mecella, B. Salvatore, G. Vetere, S. Dustdar, L. Juszczyk, A. Manzoor, and H. Truong. Pervasive Software Environments for Supporting Disaster Responses. *IEEE Internet Computing*, 12:26–37, 2008.

[2] M. de Leoni, A. Marrella, M. Mecella, S. Valentini, and S. Sardina. Mobile ad hoc networks for collaborative and mission-critical mobile scenarios: a practical study. In *Proceedings of the 2nd IEEE International Workshop on Coordination Models and Applications: Knowledge in Pervasive Environments (CoMA)*, 2008.

[3] M. de Leoni, M. Mecella, and G. De Giacomo. Highly dynamic adaptation in process management systems through execution monitoring. In *The 5th Internation Conference in Business Process Management (BPM'07)*, pages 182–197, 2007.

[4] P. Drucker. *The Effective Executive.* HarperCollins Publishers, 1993.

[5] M. Dumas, W. van der Aalst, and A. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology.* Wiley, 2005.

[6] P. Gupta and P. R. Kumar. The capacity of wireless networks. *IEEE Transactions on Information Theory*, IT-46(2):388–404, March 2000.

[7] T. Holmes, M. Vasko, and S. Dustdar. VieBOP: Extending BPEL Engines with BPEL4People. In *Proceedings of the 16th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP 2008)*, pages 547–555, 2008.

[8] International Organization for Standardization. *Human-centered Design Processes for Interactive Systems.* International Standards Organization, ISO Standard 13407:1999, 1999.

[9] A. Jardosh, E. BeldingRoyer, K. Almeroth, and S. Suri. Towards realistic mobility models for mobile ad hoc networks. In *Proceedings of MobiCom*, 2003.

[10] J. Jing, K. E. Huff, B. Hurwitz, H. Sinha, B. Robinson, and M. Feblowitz. Wham: Supporting mobile workforce and applications in workflow environments. In *Proceedings of the Tenth International Workshop on Research Issues on Data Engineering: Middleware for Mobile Business Applications and E-Commerce (RIDE 2000)*, pages 31–38, 2000.

[11] M. Kloppmann, D. Koenig, F. Leymann, G. Pfau, A. Rickayzen, C. von Riegen, P. Schmidt, and I. Trickovic. WS-BPEL Extension for People - BPEL4People, 2005.

[12] J. Li, C. Blake, D. S. J. De Couto, H. I. Lee, and R. Morris. Capacity of Ad Hoc Wireless Networks. In *Proc. 7th International Conference on Mobile Computing and Networking (MOBICOM 2001)*, pages 61–69, 2001.

[13] B. S. Manoj and A. Hubenko Baker. Communication Challenges in Emergency Response. *Communincation of ACM*, 50(3):51–53, 2007.

[14] J. McEndrick. BPEL4People Advances toward the Mainstream. Blog entry prompted on December 4th, 2008 at http://blogs.zdnet.com/service-oriented/?p=1061., 2 2008.

[15] A. Neyem, S. F. Ochoa, and J. A. Pino. Integrating service-oriented mobile units to support collaboration in ad-hoc scenarios. *Journal of Universal Computer Science*, 14:88–122, 2008.

[16] N. Nicoloudis and D. Pratistha. .NET Compact Framework Mobile Web Server Architecture. http://msdn2.microsoft.com/en-us/library/aa446537.aspx, 2003. Prompted on June 8th, 2008.

[17] L. Pajunen and S. Chande. Developing workflow engine for mobile devices. In *Proceedings of 11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007)*, pages 279–286, 2007.

[18] N. Russell, W. M. P. van der Aalst, A. H. M. ter Hofstede, and D. Edmond. Workflow resource patterns: Identification, representation and tool support. In *Proceedings of CAiSE*, pages 216–232, 2005.

[19] R. Sen, R. Gruia-Catalin, and C. Gill. Cian: A workflow engine for manets. In *Proceedings of the 10th international conference on Coordination Models and Languages (Coordination'08)*, pages 280–295, 2008.

[20] R. Sternberg. *Cognitive psychology.* Wadsworth Publishing, 3rd edition, 2002.

[21] C. Wohlin and M. Höst. Controlled experiments in software engineering. *Information and Software Technology*, 43:921–924, 2001.

Appendix – Proof of the Statement in Section 6.1

From the first assumption, we can say that, if the i-th node successes in transmitting, then $(i-1)$-th, $(i-2)$-th, $(i+1)$-th and $(i+2)$-th cannot.

Let us name the following events: *(i)* D_n be the event of delivering a packet in a chain of n links and *(ii)* S_n^i be the event of delivering at the i-th attempt.

Let us name $T_{i,n}$ as the probabilistic event of delivering a packet in a network of n links (i.e., $n+1$ nodes) after i retransmissions [14].

For all n the probability of delivering after one attempt is the same as the probability of deliver a packet: $P(T_{1,n}) = P(D_n)$. Conversely, probability $P(T_{2,n})$ is equal to the probability of not delivering at the first $P(\neg S_n^1)$ and of delivering at the second attempt $P(S_n^2)$:

$$P(T_{2,n}) = P(S_n^2 \cap \neg S_n^1) = P(S_n^2) \cdot P(\neg S_n^1 | S_n^2) \tag{3}$$

Since, for all i, events S_n^i are independent and $P(S_n^i) = P(D_n)$, Equation 3 becomes:

$$P(T_{2,n}) = P(S_n^2) \cdot P(\neg S_n^1) = P(D_n) \cdot (1 - P(D_n))$$

In general, the probability of delivering a packet to the destination node after i retransmissions is:

$$\begin{aligned} P(T_{i,n}) = P(S_n^i) \cdot P(\neg S_n^{(i-1)}) \cdot \ldots \cdot P(\neg S_n^1) = \\ = P(D_n) \cdot (1 - P(D_n))^{i-1} \end{aligned} \tag{4}$$

We can compute the average number of retransmissions, according to Equation 4 as follows:

$$\begin{aligned} T_n = \textstyle\sum_{i=1}^{\infty} P(T_{i,n}) = \\ = \textstyle\sum_{i=1}^{\infty} P(D_n) \cdot (1 - P(D_n))^{i-1} = \frac{1}{P(D_n)} \end{aligned} \tag{5}$$

In a laboratory, all nodes are in the same radio range. Therefore, independently on the nodes number,

$$P(D_n^{lab}) = 1/n \tag{6}$$

On the field, we have to distinguish on the basis of the number of links. Up to 2 links (i.e., 3 nodes), all nodes interfere and, hence, just one node out of 2 or 3 can deliver a packet in a time slot. So, $P(D_1^{field}) = 1$ and $P(D_2^{field}) = 1/2$. For links $n = 3, 4, 5$, two nodes success: $P(D_n^{field}) = 2/n$. For links $n = 6, 7, 8$, there are 3 nodes delivering: $P(D_n^{field}) = 3/n$. Hence, in general we can state:

$$P(D_n^{field}) = \frac{\lfloor \frac{n}{3} \rfloor + 1}{n} \tag{7}$$

[14] Please note this is different with respect to S_n^i, since $T_{i,n}$ implies deliver did not success up to the $i-1$-th attempt

By applying Equations 6 and 7 to Equation 5, we derive the number of retransmission needed for delivering a packet:

$$\begin{array}{l} T^{field}(n) = \frac{n}{\lfloor\frac{n}{3}\rfloor+1} \\ T^{lab}(n) = n \end{array} \tag{8}$$

Fixing the number of packets to be delivered, we can define a function f that expresses the throughput in function of the number of sent packets. If we have a chain of n links and we want to deliver a single packet from the first to the last node in the chain, then we have altogether to send the number n of links times the expected value for each link T_n. Therefore:

$$\begin{array}{l} Q_{lab}(n) = f(T^{lab}(n) \cdot n) = f(n^2) \\ Q_{field}(n) = f(T^{field}(n) \cdot n) = f(\frac{n^2}{\lfloor\frac{n}{3}\rfloor+1}) \end{array} \tag{9}$$

From our laboratory experiments described in Section 6, as well as from other theoretical results [12]), we can state $f(n^2) = \frac{\alpha}{n^\beta}$. By considering it and Equations 9, the following holds:

$$\frac{Q_{lab}(n)}{f(n^2)} = \frac{Q_{field}(n)}{f(\frac{n^2}{\lfloor\frac{n}{3}\rfloor+1})} \Rightarrow Q_{field}(n) = Q_{lab}(n) \cdot \big(\lfloor\frac{n}{3}\rfloor + 1\big)^{\frac{\beta}{2}} \tag{10}$$