Mobile Process Management through Web Services

Massimiliano de Leoni, Massimo Mecella
Dipartimento di Informatica e Sistemistica
SAPIENZA - Università di Roma, Rome, Italy
{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. This paper illustrates ROME4EU, a fully-fledged PAIS that can be entirely developed on Windows Mobile PDAs. To our knowledge, all existing PAISs are equipped with an engine meant to run only on laptop/desktop. And this prevents them from being used in mobile scenarios, such as emergency management. ROME4EU is based on a mobile Web service middleware and a WS-BPEL orchestrator engine. The feasibility for mobile settings introduces new challenging issues to face with, such as reduced computational power, small screen size, battery consumption, automatic adaptability to anomalous events. The paper details also an evaluation of the ROME4EU's performances and illustrates its use by civil protection operators to manage the aftermath of a (simulated) emergency.**

I. Introduction

Over the last decade there has been an increasing interest in process-aware information systems (PAIS), a.k.a Process Management Systems (PMSs) or formerly Workflow Management Systems (WfMSs). A PAIS is, according to [1], "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 (among the others) the tasks comprised in the processes and control flow, i.e., 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 define 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.

The core of a PAIS is 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 is ready for being assigned, the engine is also in charge of assigning it to proper participants; this step is performed by considering the participant "skills" required by the single task: a task will be assigned to those participants that provide all of the skills required.

Participants are provided with a client application, part of the PAIS, named *Task Handler* or Task-list Handler. It is aimed at receiving notifications of task assignments. Participants can, then, use this application to pick the next task to work on.

Nowadays, 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 tens of members. Therefore, the exploitation of PAISs to support the process enactment seems to be very helpful. Let us consider a typical pervasive scenario that is gaining momentum in the last period: emergency management. After the occurrence of a disaster, such as an earthquake, a rescue team is sent to the affected area in order, firstly, to give support to the involved people and, then, to make an assessment. Team rescue operators are equipped with low-profile devices, such as PDAs, and they should use them for coordinating the activities in accordance with a certain process specification. One of operators is elected as leader, of whom device hosts the PAIS engine.

To the best of our knowledge, all of current PAISs are characterized by engines that can run only on desktop or laptop machines that acts as servers. Task handlers typically are deployed on desktops/laptops, and if strictly needed can be deployed – with reduced functionalities – on PDAs or smartphones (in order to allow operators to work "out of the office"), but they continuously need to interact with the back-end engine deployed at the headquarters of the organization. Conversely, in emergency management, carrying out laptops is infeasible, as would greatly reduce moving capabilities of the team. Therefore, both full-fledged task handlers should be able to work on smart devices and the engine must reside on a certain device on the spot. Moreover, automatic adaptability to some anomalous events (disconnections of devices, crashes, etc.) should be present, as these events are much more frequent than in classical business scenarios.

In the light of the above considerations, we have conceived and developed a PAIS, namely ROME4EU (the Roman Orchestration Mobile Engine for Emergency Units), whose engine resides on a PDA. In the ROME4EU's vision, processes are executed through the orchestration of several services, of which some rely on some actions executed by humans and others are totally automatic (e.g., for retrieving data from sensor networks). The enactment of services to execute human-based tasks are mediated through task handlers, which are in charge of managing the interaction between the engine and process participants. Task handlers are installed on the PDAs of all participants, and the engine is installed on the PDA of the leader.

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 a certain number of hours. It is also 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, ROME4EU takes into account that pervasive and dynamic scenarios are characterized as being very instable. Several unexpected events may happen, which break the initial assumptions and, hence, make impossible the process to be carried out successfully. These unforeseen events are far from being infrequent and, hence, the process can often be invalidated. ROME4EU provides a high degree of flexibility and is able to restructure the process specification automatically. Adaptability is based on some sensors that are monitoring the status of the surrounding environment, such as network coverage, devices' location, speed, distance or battery level. On the basis of the monitoring, ROME4EU learns when some exogenous events happen or are about to happen and adapts the process schema accordingly.

ROME4EU has been developed in concert with several classes of end users and their feedback has guided the design throughout the development life-cycle. According to the User-Centered Design methodology [2], several prototypes have been developed, where every new prototype was closer to the final product.

In the context of a research project, ROME4EU has been concretely showcased for emergency management in an on-the-field drill that took place in Calabria (south region of Italy) in June 2009. Different storyboards were executed in collaboration with the Italian Civil Protection.[1] The corresponding processes were carried out not only by rescue operators of the Italian Civil Protection but also of other organizations that are typically involved in managing the aftermath of an emergency, such as Fire Brigades and Red Cross.

A video that shows the drill is available on YouTube at http://www.youtube.com/watch?v=48Hs5Qwg0ho.

II. The ROME4EU's Architecture and Process Model

Figure 1 shows the reference architecture that ROME4EU relies on. As introduced in Section 1, different rescue operators need to be located on the affected area and equipped with low-profile devices, like PDAs. All software components, both client and server ones, are installed on such small devices,

[1]According to the adopted method, a storyboard is a small and complete execution of a realistic use case through the system.

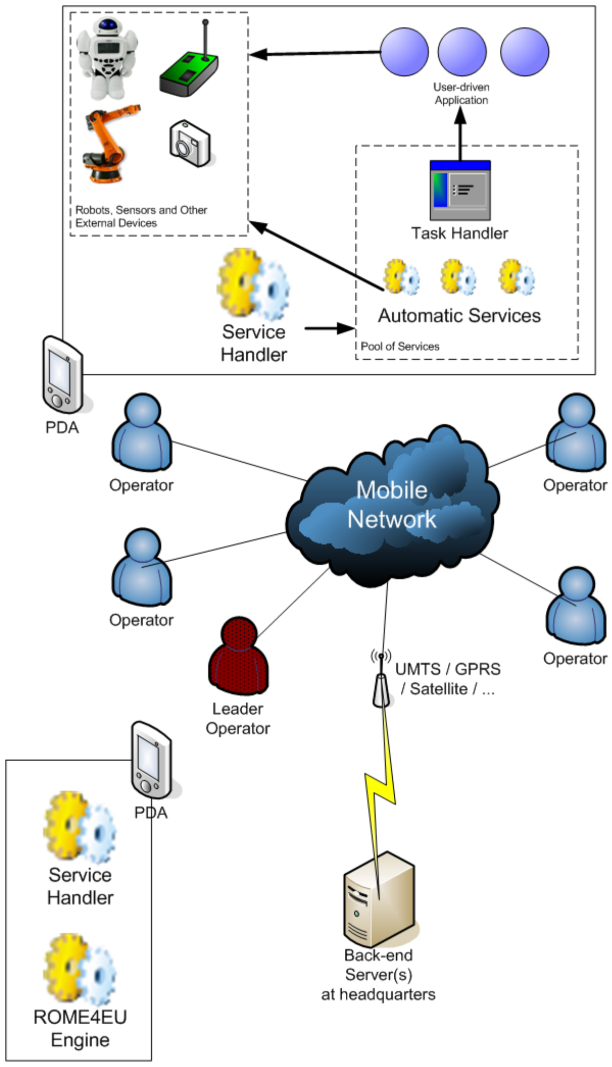


Fig. 1. The ROME4EU's Deployment

which are connected with each other through mobile networks. ROME4EU works indifferently over any (mobile) network and, hence, the figure abstracts out the specific mobile network used.

Among the rescue operators, the leader's device hosts the ROME4EU engine, which drives the execution of emergency management tasks. The device of every rescue operator, including the leader one, comes with a set of services. Such services are software applications that range from those to drive a certain hardware to those to retrieve data from back-end servers/sensors to those for visualizing maps and tracking locations of colleagues [3].[2] The interaction between the ROME4EU engine and the pool of services is mediated by the

[2]For instance, a service may be the application that provides a set of primitives to instruct a robot or an articulated arm to execute some activities, but it may also be a sensor to monitor some environmental parameters (e.g., humidity, raining intensity). Services are logically the entities that execute tasks, although in many cases they act only as proxy for a certain hardware component, which are the actual task executors.

Service Handlers. The service handler of the device of a certain operator is in charge of routing the requests of task assignment made to that device to the appropriate service running on it. When a service completes the execution of an assigned task, it returns the result to the service handler in term of output values. The service handler, in turn, forwards values to the ROME4EU engine, which updates the process variable values accordingly.

Among offered services, every operator PDA always hosts a special service, the *Task Handler*, which is a sort of proxy for the human operator. From the ROME4EU's perspective, the task handler is like any other service, but, differently from any other, it is a GUI-based application used to manage the execution of those tasks that require an interaction with the users. For the other services, when a task is assigned to them, the execution can start immediately by using the appropriate hardware/sensor/application. In fact, since the user is not needed, there is no reason to wait for the user to be free from any other duty. Conversely, when a task is assigned for being executed through the task handler, that means the task can only be executed with the support of an operator; the assigned task is not automatically started by using a certain application, but the operator is requested to give the proper acknowledgement when she is able to execute it. Once given, the task handler invokes the proper GUI-based application, needed for the task performance. Then, the operator uses that application and, when the application is closed, the task is considered as concluded. At this stage, the task handler returns the output to the service handler, which, in turn, will forward to the ROME4EU engine similarly to other services.

ROME4EU relies on a service oriented architecture where tasks are, generally speaking, executed through services that range from the Task Handlers to automatic applications. Therefore, in the light of this service-oriented view, processes specifications are naturally given in form of WS-BPEL.

As far as the resource perspective, tasks need to be assigned to proper services. The appropriateness of a service is driven by the mechanism of the so-called *capabilities*. From the one side, services declare to provide some capabilities, e.g., "able to build a tent" (for an operator able to do it) or "headquarters connection" (for the automatic application able to act as gateway). From the other side, tasks declare to require some capabilities. Every task is assigned to and executed by a service that provides all required capabilities.

Emergency management processes are highly critical and often need to be carried on within strict deadlines. Therefore, the only effective way to make a task assignment is to give a task to one service for performance and to assign no more than one task to a single service.

Network solutions. The system has been thoroughly tested on MANETs and Wireless Mesh Networks (WMNs). A MANET is a self-configuration network of mobile devices connected by wireless links. Since no infrastructure is needed (e.g., access points), it can be built up ad-hoc in a few minutes and, hence, suits emergency managements very well. A different alternative is a WMN, which is characterized by a backbone composed by several routers connected with each other through multi-hop paths of intermediate routers. Mobile devices can connect to one of the routers and, consequently, communicate with any device connected to any router of the same WMN. When devices move in the area and, hence, switch transparently from the coverage of a router to a second one, WMNs guarantee a seamless communication, even during the handoff. [3] Unlike MANETs, WMNs perform dedicated routing and path discovery in the routers, thus reducing the work load of devices. On the other hand, WMNs need routers, which are very unlikely available and, consequently, need to be taken to the area together with some power generators (if some power supply is unavailable).

III. The ROME4EU Internals

The internal architecture of ROME4EU is depicted in Figure 2 and is composed by two main subsystems: the ROME4EU Engine, and the ROME4EU Client. In addition there is the external Process Design Tool, which is used to define the process to be executed.

ROME4EU is completely developed on the .NET Compact Framework. The interaction between the engine and clients is based on web-service invocations. Specifically, we started from a pre-existing Mobile Web Server[4] that has been extended to handle complex data types, required to exchange complex input/output data sets, and one-way Web service invocations. In particular, one-way Web service invocations is a key requirement in mobile settings since it is difficult and battery consuming to keep alive TCP/IP connections (and, hence, SOAP) for long times.

The process execution is driven by our own implementation of a WS-BPEL engine, developed in Microsoft .NET C#, which has been later integrated with the Mobile Web Service mentioned before. As previously 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 executions, ROME4EU adopts an adaptability approach driven by *context-awareness*. Context-aware Information Systems allow gathering information, which makes possible for other systems to adapt their own behavior according to the current environmental context without explicit user intervention. ROME4EU uses contextual information managed and stored by *COSINE* [4]. This includes, e.g., the GPS position of team members or the status of the battery of their PDAs.

The Engine subsystem is constituted by the following components:

- **Core** manages and coordinates the execution of processes. It performs task assignments, manages and stores information about involved team members, tasks to be completed and variables produced or modified during tasks' executions.

[3]In mobile telecommunications, the term handoff refers to the process of transferring a data session from one channel connected to an antenna to another one

[4]It is available at http://msdn2.microsoft.com/en-us/library/aa446537.aspx

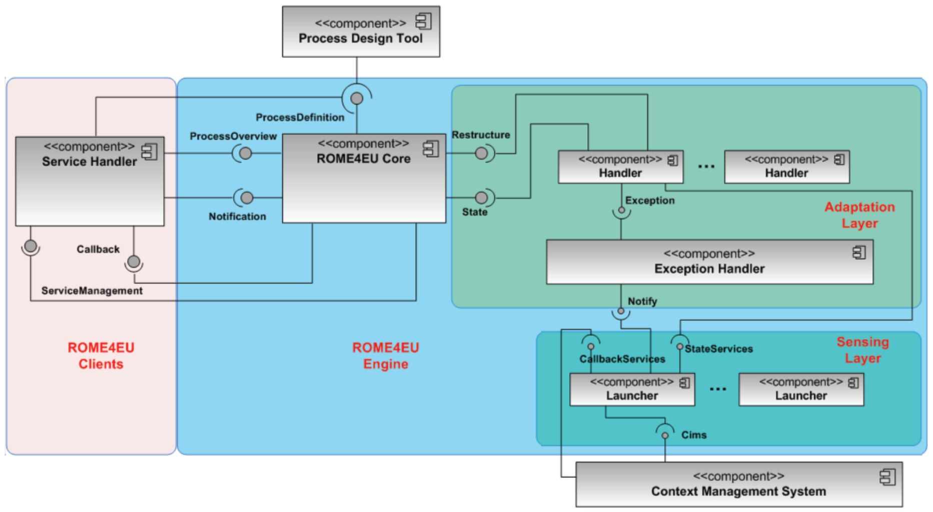


Fig. 2. The overall ROME4EU's architecture.

- **Launchers** are modules responsible for processing the information coming from *COSINE* and launching the appropriate exception to the *Exception Handler*, after an unexpected event. Their role is to monitor the whole context in which the Engine works. Every Launcher is developed to control a well defined context portion; when the managed contextual data highlight an unexpected event, the Launcher triggers an exception to the upper level, which is supposed to take the correct remedial action.
- **Exception Handler** processes the exception received from the Launchers and invokes the appropriate Handler to manage that specific error condition.
- **Handlers** are software components that implement 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%, a device is disconnecting, etc.) it performs a set of activities in order to adapt the process to the unexpected event. Each Handler is able to query the Core to get information about process execution status. On the basis of such an 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.

The **Service Handler** subsystem provides a unique end-point, on each team member PDA, for the communication with the Engine (i.e., manages remote calls to/from it). When receiving requests for task assignments from the engine, the Service Handler forwards the request to the appropriate services that will be later performing the respective tasks.

An insight into the ROME4EU's adaptivity feature

In the ROME4EU architecture, launchers are the modules computing information from the sensing layer and throwing appropriate exceptions to exception handlers, in order to handle that specific error condition. To be able to manage a new kind of error/exception, the system requires the definition of an appropriate module to interact with the sensing layer (launcher) and of a module to organize corrective actions (handler). The exception handler module (a kind of dispatcher) offers in the *Notify* interface the operation:

```
void notify(String exceptionName, Service service)
```

It catches the exception thrown by the launcher and activates the appropriate handler. The handler is informed of the module to be invoked thanks to some information of handlers stored into a XML file. This file is a sort of registry structured as a list of pairs $< launcher, handler >$ Each pair is made up by a launcher and an handler and defines where these can be reached over the network. There are also additional information concerning the sensors required for making function that module.

The adaptivity in ROME4EU is needed in order to cope with anomalous events, very frequent in mobile scenarios. In particular, so far we have identified three main categories:

- **Disconnections.** These events refer to the situation in which a device is going outside of the connection range of the mobile network. A solution is to predict the movement of the devices in the network, in order to know in advance the devices that are probably going to disconnect and to take the right recovery actions to maintain the network connected [5].
- **Consumption of device resources.** These events concern with the physical state of the devices in use (e.g., the battery level of a device could go down under a certain threshold, or the storage space of a device could be full).
- **Crashes.** These events signal an unexpected unavailability of a device, caused by accidental circumstances (a damage of the device), a sudden interruption of network connectivity, a software crash in the service handler component.

These events, if not managed in the right way, could cause problems; in particular, at the system level, the exchange of data among the engine and the clients may not be completed and the mission could not reach the normal termination; at the network level, in such cases in which the underlying network relies on the connection mutually provided by the devices (i.e., the mobile network is a MANET), the unavailability of a node could cause one or more partitions in the network. The approach used to manage external events is to have specific techniques for each type of events realized in the handlers, adapting the process according to specific adaptation patterns [6].

It worthy concluding WS-BPEL concepts such as fault/-compensation handlers are not used here. We do not want some external services to be invoked upon exogenous events. The ROME4EU system itself should take care of enacting appropriate actions to recover.

IV. A Working Example: Emergency Management

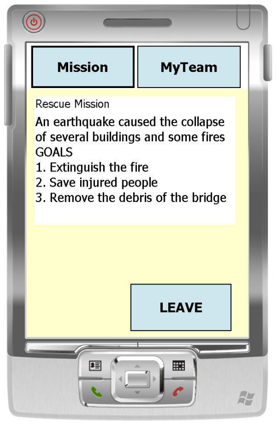
ROME4EU was completely deployed in a realistic setting in accordance with the architecture described previously. In particular, we simulated the occurrence of an earthquake in the surrounding of the small village of Pentidattilo (Reggio Calabria, Italy) on 18 June 2009. Real users from different rescue organizations were asked to face against the situation by using ROME4EU.

Two storyboards were executed. The first concerned building some medical tents to provide health support to injured people, whilst a second dealt with saving people trapped into some buildings, such as in their own houses or offices, and assessing such buildings. These storyboards are stemmed from a thorough analysis on the operational procedures of the Civil Protection and of other organizations involved in emergency management (for further details please refer to [7]). Space limitations prevent us for providing a deeper explanation of showcases. Further information is available in [8].

The most important graphical user interface is used by human participants to start executing interactive tasks and invoking the required software application. Figures 3 shows some screenshots of the application. It is worthy highlighting that users can switch to colors that are more suitable in case of adverse environmental light (e.g., in the evening or under direct sun); the choice of highly contrasting colors follows the guidelines in [9], which stems from a deep cognitive analysis with users. Since the participants to rescue operations could not have both of hands free, the use of the PDAs' stylus should be prevented as much as possible. Thus, the buttons and the other widgets of Task Handlers are sized so as to be touchable directly with fingers.

V. System Evaluation

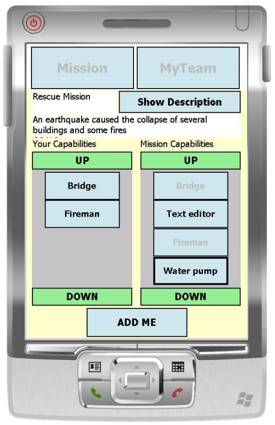
This section aims to show the practical feasibility of ROME4EU on PDAs. Firstly, we illustrate the ROME4EU requirements in term of CPU and memory consumption and how these are compatible with modern PDAs. Secondly, we show that the ROME4EU can be flexible and adapt process



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

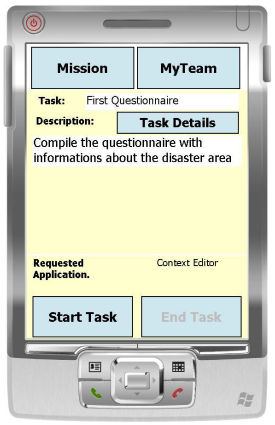
(a) When a human participant joins by Task Handler, she retrieves information about the mission goals pursued through the process.



Rescue Mission

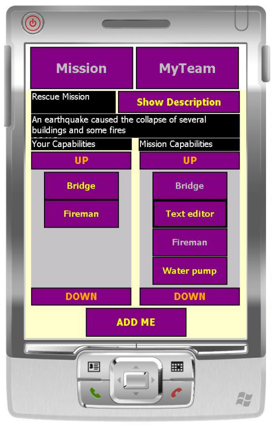
An earthquake caused the collapse of several buildings and some fires

(b) Task Handler allows participants to choose the capabilities they can provide.



Task: First Questionnaire
Description:
Compile the questionnaire with informations about the disaster area
Requested Application. Context Editor

(c) When the engine decides to assign a non-automatic task to a certain human participant, she is informed through Task Handler. Then, by clicking on button "Start Task", the application to execute that task is launched.



Rescue Mission

An earthquake caused the collapse of several buildings and some fires

(d) Users can switch to colors that are more suitable in case of adverse environmental light. Here is the same interface as in Figure 3(b) but with colors changed.

Fig. 3. The graphical user interface of *Task Handler*

specifications to fit occurring changes in the environment. The feasibility is motivated by the fact that the computation time to reason over the process and decide how to restructure the schema is of orders of magnitude smaller than the process execution time; hence, it is feasible.

As far as the users' acceptation of ROME4EU, details are provided in [8]. It is only worthy telling here that over 70% of users judged ROME4EU as effective and, hence, worthwhile to improve emergency management, and nearly everyone found the system easy to use.

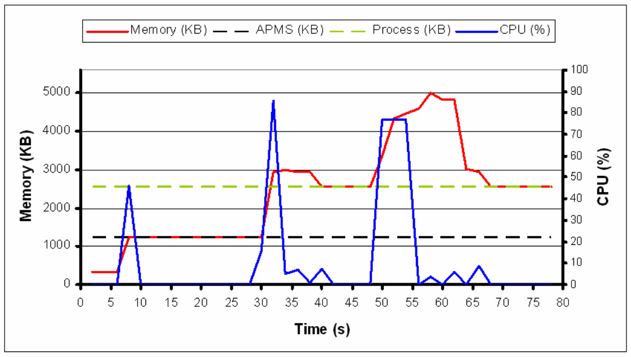


Fig. 4. CPU and memory usage patterns

a) CPU and Memory Consumption: The first set of tests aims to evaluate the use of resources (mainly, CPU and memory) in case of heavy use. Tests have been conducted loading a process composed by 30 tasks. Concurrent requests were simulated by Apache JMeter[5]. In the specific case, JMeter simulated requests by team members such as the notification of the beginning of a task execution, to which ROME4EU replies by enclosing the value of input task variables, or the notification of a task completion, which includes the value of the output variables. Figure 4 provides a detailed view of CPU and memory usage, as measured when processing 30 concurrent requests. In the first half of the chart there are two CPU peaks and some memory usage growths in correspondence with the ROME4EU activation and the process loading. When processing concurrent requests CPU usage reaches a peak of roughly 80% and memory usage increases up to 5 MB; once all requests have been processed, memory usage returns approximately to its starting level. Recalling that typical front-end teams are composed by nearly 10 members, peaks of memory usages of 5 MBs are by far acceptable. Modern PDAs are typically equipped with 128/256 Megabytes of memory of which 30/40 MBs are available in normal configurations. In terms of the CPU, the load is less than 5-10%, except during the set-up phases when ROME4EU is started and the process is loaded.

A second set of tests have been performed in order to evaluate ROME4EU during the execution of an entire process in normal and heavy-load conditions. For this purpose, we have performed some laboratory tests by executing the processes associated to the storyboards that were later showcased in the demo drill in Calabria (see Section IV). The input and output data sets were approximately of 200 KBs. Test results are shown in Figure 5. In the first part of the chart (i.e., $t < 125$) it is possible to identify CPU and memory usage patterns corresponding to ROME4EU activation, process loading and team set up. It should be clarified that execution time is not relevant, since for testing purpose tasks have been executed and completed in a few seconds, without affecting test results: in a real scenario performing a task may last even tenths

[5]http://jakarta.apache.org/jmeter/

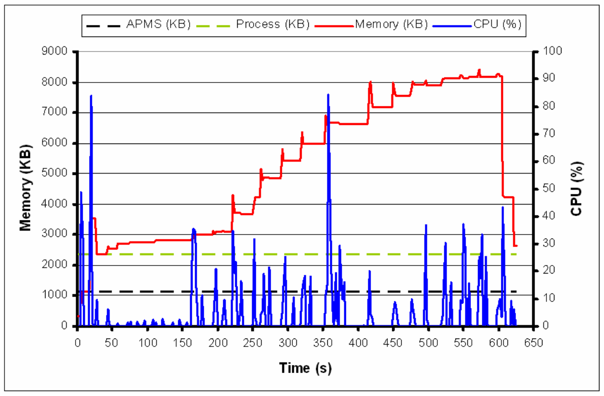


Fig. 5. Resources needed for process execution in case of data set of 200 KBs.

of minutes, but while team members execute their tasks no interaction occurs with ROME4EU.

In correspondence to notifications of the completion of tasks, CPU usage peaks of 35% have been measured. As far the memory use, peaks have been experimented during the initial start-up and when team is being built. Small peaks are also measured when tasks complete because of the incoming data to be processed. Indeed, the input/output data of tasks are kept in memory in order to improve the parsing performance of content. During the testing phase, has been experienced a situation in which 15 tasks completed almost concurrently; since the content to be parsed was nearly 200 KB per task, 3 Megabytes were required to be free in the many memory.

It is worthy pointing out that input/output data sets of 200 KB are by far a case more than pessimistic. In real scenarios, the data exchanged as input/output variables are, in fact, on average smaller. However, even in case of data sets of 200 KB, these results are also acceptable since CPU is not overloaded and the quantity of memory used is compatible with the current-day PDA configuration, in which 30/40 Megabytes are available in main memory.

b) Efficiency of the Adaptability Features: Here we want to discuss the efficiency of ROME4EU to adapt processes when unexpected events occur that can prevent processes from being carried out successfully. Specifically, we have performed some tests to detect the time elapsed from the moment when the invalidating events occur to when the process is restructured to cope with them. This time includes both the time amount to recognize their occurrence and that to restructure the process accordingly, possibly reassigning tasks to other members/services.

Tests have been conducted to measure the reaction of ROME4EU to three kinds of events: battery discharges, device crashes and disconnections of nodes. The test bed consisted of eight real PDAs, of which one was running the ROME4EU's engine and the others provided services, including Task Handler.

As far as the events of battery discharges and device crashes, those were made occur by, respectively, letting reach the

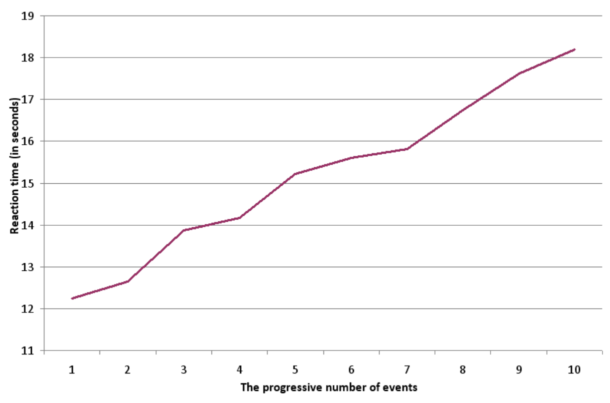


Fig. 6. Time required to adapt processes

battery level of PDAs under a certain established threshold or switching off suddenly PDAs.

For what concerns disconnections, such events were simulated through an emulator which, among other things, generates and handles a map with the virtual position of the nodes. Specific modules are installed on every PDA and instruct the emulator through special commands to move the virtual nodes on the map. These modules are also periodically querying for the node positions, which are used to predict disconnections.[6]

Firstly, it is worthy pointing out all crashes and discharges of nodes are correctly detected. Whenever a recovery existed to manage such events, ROME4EU adapted the process correctly. Ten test rounds were performed. For each test round, we simulated the occurrence of ten consecutive events, and ROME4EU was in charge of adapting the process to manage sequentially all of them.

Figure 6 shows the time amount needed to adapt processes upon unexpected invalidating events. Axes x and y measure, respectively, the progressive number of events occurred and the time amount to sense the event and adapt accordingly. The results show that the operations to recover from unexpected events are quite efficient, especially if compared with the execution of single tasks, which last from few to tenths of minutes. It has been noticed that the time required to adapt the process grows linearly as new events happen. This is motivated by the fact that adapting the process requires some data to be read and other to be stored from/into certain structures. Consequently, as new events occur, the size of these structures becomes bigger, and, hence, the access to them requires additional time. However, the adaptation requires only 18 seconds after 10 unexpected events. In prospect, if the growing trend remains unchanged, after managing 30 events, the time to adapt show be less than 40 seconds. The fact that the process would come to a halt for 40 seconds is not a big issue, since this timing is significantly lower than the typical execution times of tasks and processes.

[6]In real scenarios, these modules still exist, but they do not instruct the emulator to move nodes. In addition, the position of nodes are harvested by the GPS hardware of PDAs.

VI. Related Work

To the best of our knowledge, the majority of the existing Process-aware Information Systems cannot be fully deployed on smart devices, such as PDAs or smartphones. This holds both for open source products, e.g., jBPM[7] and Together Workflow[8], as for commercial systems, e.g., SAP Netweaver[9], Flower[10] or TIBCO's iProcess Suite[11]. 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. The recent literature provides some interesting developments of PAISs running on mobile devices. CiAN [10] is a language and system that supports collaboration through workflows involving mobile devices and participants. It has been targeted at mobile networks and, hence, addresses some of the issues of mobile environments introduced before. Unfortunately, the prototype has been implemented in Java using J2SE 5.0 and, consequently, it cannot be installed on PDAs. Moreover, it does not provide enough flexibility to handle those exogenous events which take the system to situations where processes cannot carried out successfully. The lack of flexibility is also the main issue of other mobile PAISs, such as the Nokia's implementation [11] or WHAM [12]. Additionally, WHAM binds every task to a specific service at design time. Therefore, it is unable to integrate new external services that may become available later at run time.

The BPEL4People and WSHumanTask standards [13] are currently under definition to extend BPEL processes to support activities performed by humans. As ROME4EU does, BPEL4People models every human as a service covering some roles and the interaction with each service, including humans, is modeled as message exchanged by 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 [15], but it runs only on desktop/laptop machines. Moreover, BPEL4People shows some limitations, as also discussed in [16]. There is no support for a detailed definition of specific resources to distribute tasks (e.g., as ROME4EU defines via capabilities), nor there are facilities to improve the process throughput (e.g., by optimizing the tasks' assignment). As previously motivated, these aspects are key requirements in many mobile and pervasive scenarios, such as emergency management.

[7]http://www.jboss.com/products/jbpm
[8]http://www.together.at/together/prod/tws/
[9]http://www.sap.com/usa/platform/netweaver
[10]http://global.pallas-athena.com/products/bpmflower_product/
[11]http://www.tibco.com/software/business_process_management/

VII. Lesson Learned and Conclusion

The results of the system performance evaluation and the feedbacks from users have allowed drawing some conclusions and useful guidelines for future developments. Firstly, the vision of looking at emergency management as a set of processes to carry on is also well understood by end users, who recognize the usefulness of ROME4EU and, in general, of process management.

It is also worthy highlighting that the current-day technology provides PDAs and smartphones that are powerful enough to run complex systems, such as ROME4EU. The outcomes of the ROME4EU testing phase have shown that ROME4EU guarantees response times that are of several orders of magnitude lower than the execution times of the entire processes. Therefore, running ROME4EU (or any other PAISs) on PDAs is feasible in practice.

In addition, the effort of thinking of emergency management from a process-oriented perspective has allowed people working in civil protection departments to analyze carefully the current-day procedures for facing against disasters. This analysis has resulted in systemizing thoroughly the procedures followed to manage emergencies, which finally translates to a better process control. Surprisingly, during the initial phases of user-requirement collection, we learned that civil-protection operators did not have clearly in mind the actual procedures and activities that they followed to face against emergencies. In sum, the adoption of ROME4EU would guarantee a more systematic management of the aftermath of emergencies, thus yielding to an overall improvement of the response time that is not only motivated by the mere use of the system.

Stemming from the experience acquired during the demo drill, in pervasive scenarios, such as emergency management, information processing and task execution is fully integrated with the physical environment and its objects. The physical interaction with the environment increases the frequency of unexpected contingencies with respect to classical scenarios. Being pervasive scenarios very dynamic and turbulent, providing a higher degree of operational flexibility/adaptability is a key requirement of every PAIS for pervasive scenarios. And this requirement is adequately met by ROME4EU.

From a visualization viewpoint, Task Handlers and other client applications supporting the execution of tasks should be conceived for being used in extreme conditions and under direct sunlight. Therefore, the use of highly-contrasting colors is important (e.g., white on black, yellow on blue). Moreover, users might not have free hands to use PDA's stylus. Therefore, the GUI widgets should be sized in a way that the use of styluses can be avoided (e.g., participants should be able to touch and press buttons by fingers).

Acknowledgement

This work has been partly supported by the FP6 project WORKPAD and the 2009 University/Faculty grant METRO. Authors wish also to thank all people who contributed to the development of the system, namely Daniele Battista, J. Valerio Franchi, Daniele Graziano, Paolo Manfre', Andrea Marrella, Alessandro Russo.

References

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

[2] A. Dix, J. Finlay, G. Abowd, and R. Beale, *Human Computer Interaction (3rd Edition)*. Prentice Hall, 2004.

[3] M. Bortenschlager, "Location-oriented Coordination in Pervasive Environments for Collaborative Work Scenarios," *Journal of Location Based Services*, vol. 3, no. 4, pp. 229–248, 2009.

[4] L. Juszczyk, H. Psaier, A. Manzoor, and S. Dustdar, "Adaptive Query Routing on Distributed Context - The COSINE Framework," in *MDM 2009: Proceedings of the 10th International Conference on Mobile Data Management*. IEEE Computer Society, 2009, pp. 588–593.

[5] M. de Leoni, M. Mecella, P. Manfre', J. V. Franchi, and D. Graziano, "Disconnection prediction in mobile P2P networks using publish/subscribe," in *ICUMT '09: International Conference on Ultra Modern Telecommunications & Workshops*. IEEE, 2009.

[6] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - Enhancing flexibility in process-aware information systems," *Data & Knowledge Engineering*, vol. 66, no. 3, pp. 438–466, 2008.

[7] S. R. Humayoun, T. Catarci, M. de Leoni, A. Marrella, M. Mecella, M. Bortenschlager, and R. Steinmann, "Designing Mobile Systems in Highly Dynamic Scenarios. The WORKPAD Methodology," *Journal on Knowledge, Technology & Policy*, vol. 22, no. 1, pp. 25–43, 2009.

[8] M. Mecella, M. de Leoni, A. Marrella, T. Catarci, M. Bortenschlager, and R. Steinmann, "The WORKPAD Project Experience: Improving the Disaster Response through Process Management and Geo Collaboration," in *Proceedings of the 7th International Conference on Information Systems for Crisis Response and Management (ISCRAM2010)*, 2010.

[9] T. Agostini and N. Bruno, "Lightness contrast in CRT and paper-and-illuminant displays," *Perception & Psychophysics*, vol. 58, no. 2, pp. 250–258, 1996.

[10] 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)*, 2008, pp. 280–295.

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

[12] 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)*, 2000, pp. 31–38.

[13] 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.

[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] 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)*, 2008, pp. 547–555.

[16] N. Russell and W. M. P. van der Aalst, "Work Distribution and Resource Management in BPEL4People: Capabilities and Opportunities," in *CAiSE '08: Proceedings of the 20th international conference on Advanced Information Systems Engineering*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 94–108.