

# The RASCAL System for Managing Autonomic Communication in Disruptive Environments

Roberto Ghizzioli  
Whitestein Technologies AG  
Zürich, Switzerland  
Email: rgh@whitestein.com

Dominic Greenwood  
Whitestein Technologies AG  
Zürich, Switzerland  
Email: dgr@whitestein.com

**Abstract**—This paper presents RASCAL, a contingency manager system with autonomic capabilities which enables collaboration among a set of ubiquitous services deployed in the infrastructure and/or in the ad-hoc network. Autonomicity is shown in the context of disruptive environments, that is, a series of locations that may be subject to intermittent or complete disruption to communications equipment. In this paper we show how the RASCAL autonomic architecture and its adaptable policies determine the types of actions to be taken in order to exhibit stability to the end user.

## I. INTRODUCTION

For many, mobility is now a central aspect of everyday life to the extent where mobile users expect to be *always-best-connected*, i.e., they expect anywhere and anytime access with the maximum capacity on offer. As providers rush to deploy the network infrastructure required to deliver high-quality on-demand services, mobile devices including laptops, PDAs, and cellphones can begin to deploy their own ubiquitous services. Just a sample of emerging services used within the health-care and emergency response domains include collaborative technologies such as map/location services, interactive shared whiteboards and wireless health monitors [12].

Empowering the anywhere/anytime access aspect of such ubiquitous services we can now observe the emergence of multi-technology systems and software supporting flexible communication [7]. That is, using any available network technology to communicate, whether infrastructure based (e.g., WLAN, WiMAX or Cellular), or ad-hoc based (e.g., Bluetooth, Ultra-Wideband, or even Infrared).

Naturally such communicative flexibility becomes increasingly important as the operating environment becomes more dynamic or disruptive in nature, and the availability of different network technologies can change rapidly. This is yet more critical in emergency scenarios such as disaster zones and major incidents.

This paper reports on a technology prototype based on concepts drawn from *autonomic communication* research [1], [14],

to allow mobile devices to autonomically self-manage connection endpoints and data transmission over available network technologies.

The prototype is termed the RASCAL system, where RASCAL is an acronym meaning *Resilience and Adaptivity System for Connectivity over Ad-hoc Links*. RASCAL is a novel middleware communication mechanism that automatically ensures (to such degrees as are possible within the operating environment) the continued operation of ubiquitous application services where communication may be subject to disruption. RASCAL has been designed and implemented as a deliverable of the EU-funded PalCom (Palpable Computing) project<sup>1</sup>.

To achieve this goal the RASCAL system shifts the burden of tasks such as configuration, maintenance and fault management from users to a specialised self-management subsystem. Each RASCAL system uses a local policy engine for *self-configuration* purposes allowing it to adjust its behaviour in accordance with environmental changes. RASCAL is also *self-optimizing* because it monitors network resources and adapts its behaviour to meet the end-user and application service needs, i.e., automatically switching between WLAN and Cellular connections to maximise the always-best-connected goal. Furthermore, RASCAL is also *self-healing* when managing multiple bearer technologies, such as WLAN, 3G or Bluetooth; for example, switching to an ad-hoc connection in the temporary absence of an infrastructure connection. Finally, RASCAL also offers an intuitive interface allowing the user to inspect ongoing activities, decisions and internal state of the system.

Currently, the RASCAL system operates in conjunction with *palpable devices*, defined by the PalCom project as discoverable user devices that offer one or more services that may serve as constituents of an *assembly*. An assembly is a dynamically composed collection of devices and services delivering on a transient task defined by the user [10]. Although in this respect the RASCAL system uses the PalCom-defined communication stack, it has no specific dependency and can thus operate as an independent autonomic control subsystem on multiple devices and platforms.

The remainder of this paper is organized as follows: Section II provides some background on disruptive environments highlighting the communication complexity and requirements

Presented at the 1st IEEE Workshop on Autonomic Communication and Network Management (ACNM), as part of the 10th IFIP/IEEE International Symposium on Integrated Network Management, May 21-25, 2007, Munich, Germany.

<sup>1</sup>IST-002057

it places on RASCAL. Section III discusses some relevant previous work on autonomic communication dealing with disruptive environments. Section IV then presents the autonomic features of RASCAL and Section V presents the RASCAL architecture. Section VI shows a scenario where RASCAL was evaluated before concluding in Section VII.

## II. DISRUPTIVE ENVIRONMENTS

Our definition of a disruptive environment is a physical location wherein there is a high probability that some disruption will occur diverging from normal, expected behaviour. In this paper we are, in particular, concerned with disruption to channels of electronic communication in environments such as those impacted by *natural disasters* (e.g., tsunamis, hurricanes, earthquakes, floods, forest fires, etc.), sites of *major incidents* [12], [13] (e.g., plane crashes, multi-vehicle road traffic accidents, building fires, etc.), *theatres of military operations* and also other more *everyday situations* such as healthcare telemedicine [15] and remote working.

In all of these environments there is often a pressing need to communicate information from one locale to another, both within, from and to the environment. A straightforward example is the communication of information between rescue workers operating in disaster zones. The devastation caused by recent global events such as 9/11, the Asian tsunami, and hurricane Katrina, to name but a few, demonstrate that emergency services must coordinate at multiple levels and with absolute guarantees of information finding its way from sender to receiver via one means or another. Emergency response services must, for example, build ad-hoc rescue teams before acting at an incident site, continuously communicating with one another to exchange orders, share findings, request help, etc. (see Figure 1(a)).

Assured means of communication in such scenarios is highly important because it helps save lives. This can also be the case with telemedicine where health workers must receive patient monitoring information at all times (see Figure 1(b)). And even the remote business worker attempting to remain connected to a company network may be subject to transient availability of access network connections in certain locations.

Due to the potentially disruptive nature of such environments, communication is often subject to unpredictable network conditions. Radios may fail due to electromagnetic interference, infrastructure networks may be inoperative due to physical damage, cellular and satellite communication may be impossible due to poor or intermittent signal strength and ad-hoc connections may be limited due to the availability of local nodes. In general terms, we consider a disruptive communications behaviour as an event occurring in a disruptive environment that alters, modifies, or interferes with data transmission as it travels through interconnected channels between a source and a receiver.

To maximize the probability for successful transmission we require an intelligent, autonomic messaging platform that allows real-time, secure, bi-directional communication of any information from source to destination(s) while remaining

agnostic to the devices, networks or carriers required to transfer the information. This requires migrating communication (e.g., message handling) intelligence into the cooperating user devices to allow iterative delivery decision-making throughout the communication route. This is manifested by *recipient pursuit* where intended receivers are tracked down by attempting to move a message closer to their estimated location with each hop (see Figure 1(c)).

The architectures of today's infrastructure networks assume that physical connectivity exists on an end-to-end basis between sources and destinations for extended periods of time. For networks operating in a disruptive environment, these assumptions are no longer valid, and new approaches to routing, congestion and flow control are required. The RASCAL project proposes an architecture and policies supporting end-to-end reliable communication in environments with such intermittent connectivity.

## III. RELATED WORK

There are several published studies available regarding the deployment of autonomic communication and network management systems in disruptive environments. However, we find that the majority consider only specific aspects of the domain. For example, when addressing network aspects many papers focus only on either infrastructure or on ad-hoc networks (MANET) without considering the synergy of using the two type of networks concurrently, and in support of one-another. A specific case in point is the work of Chadha *et al.* [3] who present an autonomic system developed under the U.S. Army CERDEC DRAMA (Dynamic Re-Addressing and Management for the Army) program deployed in military scenarios. This system in particular only addresses mobile ad-hoc networks without consideration of potentially available infrastructure networks.

Those papers that deal with hybrid networks (infrastructure and mobile ad-hoc), few consider either the requirements of the end user applications running on top of the presented autonomic systems, or the roles of end-users in various deployment scenarios.

An example of hybrid network management is provided in Hauge *et al.* [8] who present an interesting approach to the combination of 3G cellular and ad-hoc networks. They conclude that hybrid networks provide the opportunity to transmit service data to a higher percentage of interested mobile terminals than when using only an infrastructure network. Nevertheless, the role of their user-level service (multicast) within hybrid networks is not considered. The same can be said of the work of the Delay Tolerant Networking Research Group [5]: they mainly focus on network aspects providing end-to-end connectivity in disruptive environments without considering how application contexts influence the achievement of connection/delivery goals. We address this issue as a component of the RASCAL usage-aware approach (see section IV).

Another work in this area is by Kappler *et al.* [11] who use a policy-engine to address hybrid network composition.

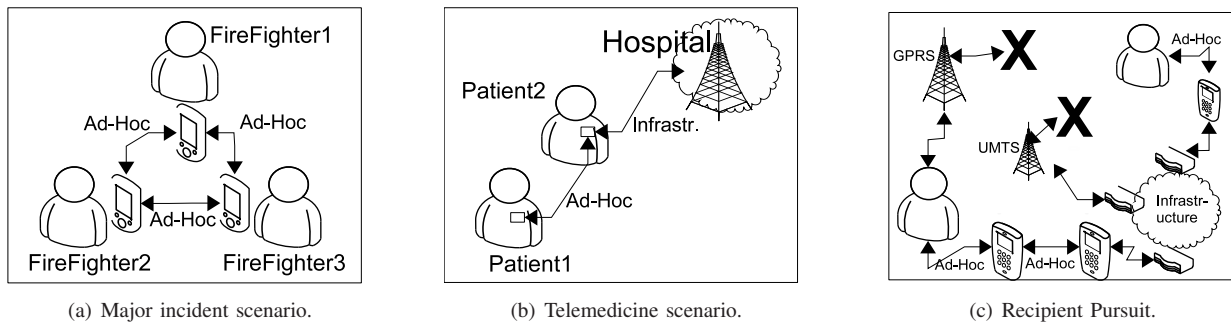


Fig. 1. Disruptive environments.

Although this aspect is in line with our approach, the authors do not consider the discovery of relevant network nodes and policies are only used for the composition of network devices, whereas our approach also considers user-level service composition.

On the other hand, previous work on ambient and pervasive computing used in disruptive environments or disaster management tends to focus more on service composition or other application-oriented aspects without considering the underline networks issues. Such an example is the reported work of Kristensen *et al.* [13] who focus on IT support in major incidents, such as the use of bio-monitors, person identification and collaboration tools for response units, without considering how these application should behave when faced with network disruptions.

The RASCAL system reduces this gap between specific autonomic aspects purely based on the network management and the autonomic aspects based on the user-level services deployed in disruptive environments.

#### IV. RASCAL AUTONOMIC FEATURES

RASCAL is a middleware communication layer which resides between the user applications and the underlying networks enhancing the user experience when these applications are used in disruptive environments. When RASCAL is used by a device the device becomes "RASCALized" and from that moment on, all messages sent by applications are given to RASCAL which decides, based on a set of policies, the most appropriate actions to take on them. Enhancing the user experience in such situations implies making *autonomic decisions* when sending/receiving application messages to/from a target node. Disruptive environments are normally highly dynamic and normally collaborative applications such as instant messengers, VOIP clients, GPS or GIS map services do not exhibit the autonomic behaviours necessary to deal with such environments.

RASCAL brings about *connection awareness* through a set of autonomic behaviours which are triggered by changes in network resources. Several autonomic decisions are available including:

- **Network handover** : Switching from one network type to another when communicating with other devices. This

decision can be taken based on the reachability of a device over different networks (e.g. infrastructure or ad-hoc) and on the networks availabilities. For example, we can consider handovers from an infrastructure technology (e.g., UDP) to an ad-hoc technology (e.g., Bluetooth) when communicating with a device in the neighborhood in the face of network problems.

- **Routing optimization** : Enhancing multi-hop routing among network nodes. Parameters which affect these decisions can be based on several QoS parameters such as response delay, nominal and available bandwidth between network nodes, transmission errors, etc. For example, a self-optimization feature of RASCAL fitting into this group is to proactively evaluate the transmission delay between two interoperating devices and consequently use another path to reach the same target node.

Other behaviours within this category include those acting in response to failovers or high network load, etc.

RASCAL also offers *usage-aware communication* consisting of a set of autonomic behaviours related to the usage of deployed user-level services. For example, a group of firefighters are on the site of a major incident and must constantly communicate both with one another and with a response unit about their findings and the positions of injured people. Due to the disruptive nature of the environment, network availability may be intermittent, but the goal to reliably deliver communication must persist. In this example autonomic decisions can be taken such as:

- **Transmission contingency**: Providing alternatives to the default means of transmitting a message. This specifically includes making best use of multi-technology transmission paths including cellular networks, IP infrastructure networks, satellite systems, MANETs, etc. An important parameter able to affect these decisions is the importance of the message to be sent. An example is simultaneously sending high priority messages via two or more different technologies, and therefore routes, to improve the chances that they are successfully delivered to target nodes. The use of extra resources is justified by the importance of the content to deliver.
- **Content adaptation**: Adapting the content of a message (or stream). These decisions can be based on several

parameters like the number and the importance of the messages/streams to be sent. Examples include applying a codec to reduce the used bandwidth of a video stream, or simply stripping out the audio component and sending this in lieu of the video.

- **Deferred service provisioning:** Waiting until a connection is available before making routing decisions to mitigate uncertainty relating to the choice of optimal technologies or paths. Also in this case the parameters able to trigger this type of decisions are the number and the importance of the information to be sent. This includes the need to buffer messages while awaiting a connection.
- **Role management:** Specifying user defined conditions which must be met before taking a particular action. A parameter which affects these decisions is the role of the end-user in a particular scenario. For example within a major incident response workers are divided into response units, structured in a hierarchical way. In this scenario we can envisage policies which ensure a message (e.g. notification of an event) is sent to the right recipients in the role hierarchy (e.g. escalating).

In order to enable the end user to quickly define or modify policies which govern the autonomic features of the RASCAL System running on the his/her device, a mechanism for on-the-fly updates is adopted.

Additionally, another important feature of RASCAL is the notification and interaction aspect with the end user. In many systems when a problem occurs during a communication the end user is not notified. RASCAL is able to trace the communication and to provide *inspection* capabilities using an appropriate GUI. More details are given in Section V-D.

## V. THE RASCAL ARCHITECTURE

The RASCAL software architecture, depicted in Figure 2 with solid lines, is fully compliant with the well known IBM autonomic control loop [9], in that it:

- *collects* information from the system and the external world,
- *make decisions* using a policy engine,
- and *adjusts* the system as necessary.

The architecture consists of six main components, described in the following subsections. In overview they consist of three managed elements (applications service, networks service and policy service), an autonomic manager (RASCAL agent), a policy engine and a GUI.

RASCAL is able to communicate and interact with different layers of the user-device communication stack (depicted in Figure 2 with dotted lines). For example, it normally interacts with components dealing with media layer technologies, e.g. UDP, Bluetooth, etc., or with components implementing high communication layers, e.g., routing protocols, discovery systems, user applications. This is the case in the PalCom project, but different scenarios where different third-party components or implementations are in place, are not excluded.

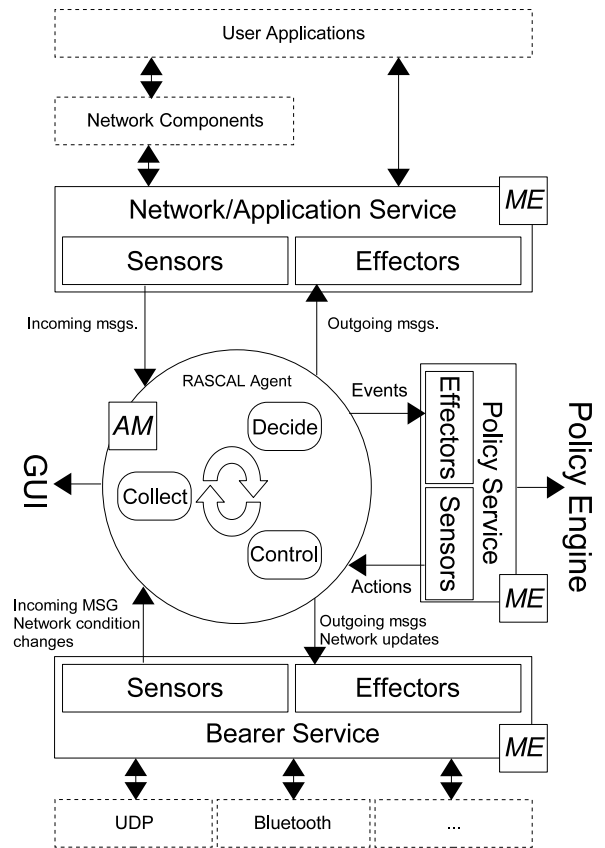


Fig. 2. The RASCAL software architecture.

The integration between a third-party network component and RASCAL is possible through the software interfaces offered by the RASCAL system.

The RASCAL architecture is implemented using JADE (Java Agent Development Framework), a software agent platform and development system [2]. Within the JADE runtime are contained the autonomic manager (implemented as a *software agent*) and the managed elements (implemented as *kernel services*). In general terms a *software agent* is defined by Wooldridge [18] as a “computer system that is situated in some environment and that is capable of autonomous action in this environment in order to meet its design requirements”. A JADE kernel service is defined by Bellifemine *et al.* [2] as a “software component which implements platform level features that can be grouped together according to their conceptual cohesion”.

### A. The Managed Elements (ME)

The RASCAL agent interacts with the external world via JADE kernel services. Each service is controlled through sensors and effectors. Effectors produce actions relating to instructions received from the RASCAL agent; they implement the *Command* design pattern [6]. Sensors collect information from the external world and provide it to the RASCAL agent for processing; they implement a simplified version of the *Half-Sync/Half-Async* design pattern [16].

The RASCAL system contains three JADE kernel services:

*Network/Application Service.*: This is used to receive messages sourced from end user applications or high level network components. A selection of interfaces allow communication with a broad range of applications, examples of which include the previously mentioned PalCom services designed for disruptive environments, or any other user level application such as VOIP clients, IM, etc.

*Bearer Service.*: This is used to interact with the underlying infrastructure or ad-hoc networks. This service is able to send/receive messages over different technologies and to generate events based on network status.

*Policy Service.*: This is used by the RASCAL agent to communicate with a local policy engine (see Section V-C).

### B. The Autonomic Manager (AM)

The autonomic manager component is implemented by the RASCAL software agent. It exhibits the following behaviours:

*Sensing.*: By installing sensors in the managed elements the agent is able to gather new application messages to be sent, new messages received from the network or new network status events. Data is collected both asynchronously (the managed elements notify of a status changing) or synchronously (the agent explicitly request for information).

*Compiling Knowledge.*: Received events are used to model an internalized representation of the external world. This compiled knowledge base will contain information relating to discovered devices, the services they provide, and the physical addresses of these services.

*Decision Control.*: Once the internal knowledge base is updated, RASCAL triggers a policy engine which controls decisions on which action to apply on the system. This aspect is detailed in Section V-C.

*Proactivity.*: Actions can be executed by RASCAL immediately or postponed until some time in the future. In order to schedule such future actions RASCAL implements a model of time allowing proactive planning.

### C. Decision Making

A core aspect of the RASCAL system is its reasoning system. The RASCAL agent receives messages from user-level applications and probes the environment using the previously discussed kernel services. Decisions on how to treat the received messages are made locally using *policies*; a set of constraint rules governing system behaviour. One of the goals of RASCAL is to provide the end user with an easy means of authoring policies that will control the various autonomic features (see Figure 3). In order to modify the RASCAL behaviour at runtime, these policies are dynamically loaded when the device is running.

Policies are not coded directly within the agent behaviours. To be more flexible, the agent uses the Policy Service, shown in Figure 2, to issue events to a policy engine and wait for a set of recommended actions to perform. The particular policy server employed by RASCAL is Ponder2 [4], used as a local library, which uses an XML-based policy description

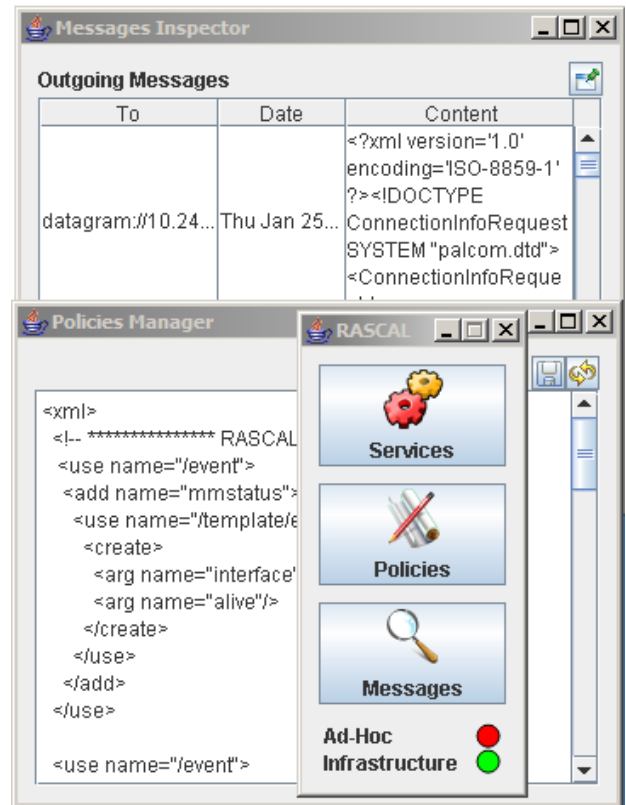


Fig. 3. The RASCAL GUI

language to define events and policies to be processed by the Ponder2 policy engine. The result of a policy is an action the RASCAL agent has to perform. Currently, RASCAL deals with *obligation policies*. An obligation policy is an Event Condition Action (ECA) rule in the deontic sense [17]. Given  $E$ , and  $C$  is true, it is obligatory that the agent performs  $A$ .

### D. Graphical User Interface

The RASCAL user interface is designed for control and inspection using event-based interaction with the RASCAL agent. The main panel, shown in Figure 3, indicates the status of the various network interfaces available on the local device and a set of buttons to open inspection views. One of these views provides a list of all remote devices interacting with user-level services running on the local device. Additionally, a second view is dedicated to message inspection and a third to inspecting, editing and controlling policies definitions. Currently, the RASCAL GUI has been implemented to work only on laptops or personal computers but it can be easily adapted to other consumer devices like PDA, smartphones, etc.

## VI. EVALUATION

RASCAL has recently been integrated into the iterative, participatory design process practiced in PalCom. We are currently carrying out experiments with end users in major incident emergency response scenarios. This section presents

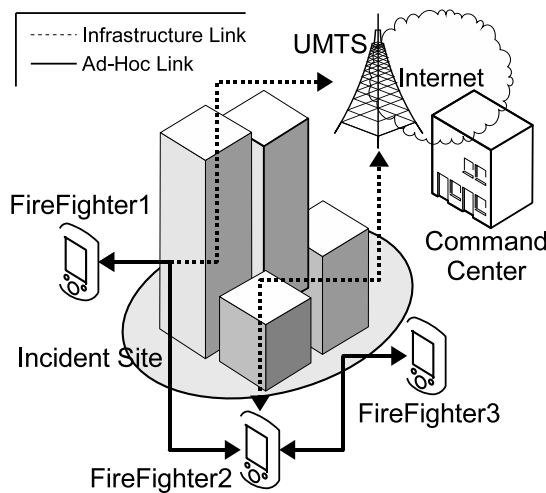


Fig. 4. The conducted mocked-up situation.

an overview of one of the mocked-up situations of a real world major incident conducted at a recent PalCom project review.

Within this experiment a building fire is considered. This particular scenario demands fast and effective actions, often in life-threatening situations. It also requires collaboration between numerous people located in different, often changing, areas: personnel at the incident site (e.g. firefighters, police, doctors, etc.), at the command center, in vehicles, etc.

Each of the people involved, and many of the vehicles and other equipment, are associated with one or more electronic devices such as radios, biosensors, GPS, health recorders, handhelds, tablet PC, etc. In the fire scenario different devices run different crisis-relevant applications including VOIP clients, instant messengers, map services and other collaborative tools.

A view on aspects of the presented scenario is depicted in Figure 4. Here three firefighters (FF) are moving relatively close to one another to evacuate people from a building on fire. They are using special "RASCALized" tablet PCs, each with a built-in camera running a map service. Their duty is to notify the command center (CC) and the other workers of the team of findings related to the visited building(s), e.g., the positions of injured people. To do this, they make special marks on the map displayed on their tablet PC. Firefighters can also take pictures to assist the command center with gaining a visible overview of the overall incident status. Furthermore, in this example, FF1 and FF2 are connected via both ad-hoc (HOC) and infrastructure (INFR) networks and FF3 only via an ad-hoc connection. In this situation, through the multi-hop capabilities of RASCAL all the four actors (the three firefighters and the command center) are able to communicate one another. For example FF3 communicates with the command center via the ad-hoc connection with FF2.

The RASCALized devices used by the firefighters are equipped with the following policies:

- 1) IF (infrastructure\_connected) THEN send map data to CC and FF via INFR.

```

<use name="/policy">
  <add name="contentAdapt">
    <use name="/template/policy">
      <create type="obligation"
        event="/event/mmstatus" active="true">
        <arg name="alive"/>
        <condition>
          <eq>protocol;datagram</eq>
          <eq>!alive;false</eq>
        </condition>
        <action>
          <use name="/rascalmo">
            <sendLowResolutionPicture
              protocol="bluetooth"/>
          </use>
        </action>
      </create>
    </use>
  </add>
</use>

```

Fig. 5. Example of policy to send low resolution pictures to the command center towards the ad-hoc network when the infrastructure connection is not working anymore.

- 2) IF (!infrastructure\_connected) THEN send map data to CC and FF via HOC.
- 3) IF (infrastructure\_connected) THEN send high resolution pictures to CC via INFR.
- 4) IF (!infrastructure\_connected) THEN send low resolution pictures to CC via HOC.

Figure 5 shows the definition of the final policy in the list presented above. This policy sends low resolution pictures to the command center when the device is not infrastructure connected. It receives *mmstatus* events which denote if a network interface is up or not. This event has two associated parameters: the interface's protocol and its status. The only condition that triggers this policy is that the infrastructure connection is down (see the XML *condition* element). The action *sendLowResolutionPicture* triggered by this policy notifies the RASCAL agent to decrease the resolution of the pictures for the command center and to send them using the ad-hoc network (see the *protocol* attribute of the *sendLowResolutionPicture* XML element).

In our experiment, for example, FF1 has policies 1) and 3) activated. When FF1 moves into an area where their infrastructure connection fails, policies 2) and 4) automatically become active. The RASCAL agent running on the device is thus notified by the policy engine and hands over all communication with FF2 to the available ad-hoc connection. Given the importance of sending images to the command center and giving the low nominal bandwidth of the BT technology, pictures are first automatically reduced in quality (i.e., resolution) before transmission.

Later, when FF1 returns to an area with infrastructure network coverage, communications with the command center are automatically returned to the infrastructure connection with images once again sent in normal, high resolution.

A sequence diagram of the actions taken by FF1 to send

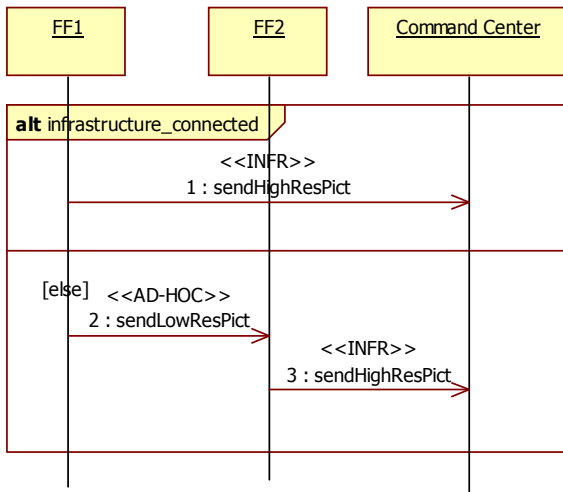


Fig. 6. Sequence diagram of the actions taken by FF1 to send pictures to the command center.

pictures to the command center is shown in Figure 6. The diagram considers the discussed situation both with and without the infrastructure connection.

## VII. CONCLUSION

This paper has presented the RASCAL system, the outcome a work package of the European PalCom project. In particular, this paper gives an overview of the RASCAL software architecture and of its internal mechanisms able to exhibit autonomic capabilities. Using agent technologies in conjunction with a policy engine the end user is able to define communication policies able to deal with disruptive environments such as major incidents.

The RASCAL system is currently under a continuous refinement and improvement process and will remain so until the end of the PalCom project in December 2007. Future work will concentrate on additional autonomic aspects of the RASCAL system. One particular aspect targeted for improvement is the policy language, where a new structure using a process-algebra over actions will be introduced. This will allow the expression of a set of actions (i.e., workflow) rather than the current limitation to atomic actions. Other planned improvements relate to the inspectability of the RASCAL system and its use in composing dynamic assemblies of computational devices and services with flexible, self-adaptive communicative connections.

## ACKNOWLEDGMENT

The authors acknowledge the EU Palpable Computing (IST-002057) FET project, which funded a proportion of the RASCAL project, and the various PalCom consortium members that contributed toward this work.

## REFERENCES

[1] D. F. Bantz, C. Bisdikian, D. Challener, J. P. Karidis, S. Mastrianni, A. Mohindra, D. G. Shea, and M. Vanover, "Autonomic personal computing," *IBM Syst. J.*, vol. 42, no. 1, pp. 165–176, 2003.

[2] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.

[3] R. Chadha, H. Cheng, Y.-H. Cheng, J. Chiang, A. Ghetie, G. Levin, and H. Tanna, "Policy-based mobile ad hoc network management," *Workshop on Policies for Distributed Systems and Networks, International*, pp. 35–44, 2004.

[4] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. London, UK: Springer-Verlag, 2001, pp. 18–38.

[5] S. Farrell and V. Cahill, *Delay and Disruption Tolerant Networking*. Artech House Publishers, 2006.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[7] D. Greenwood and M. Calisti, "The living systems connection agent: Seamless mobility at work," in *Communication in Distributed Systems (KiVS 07)*, 2007.

[8] M. Hauge and O. Kure, "Multicast service availability in a hybrid 3g-cellular and ad hoc network," in *International Workshop on Wireless Ad-Hoc Networks*, 2004.

[9] IBM, "An architectural blueprint for autonomic computing," 2006. [Online]. Available: [http://www-03.ibm.com/autonomic/pdfs/AC\\_Blueprint\\_White\\_Paper\\_4th.pdf](http://www-03.ibm.com/autonomic/pdfs/AC_Blueprint_White_Paper_4th.pdf)

[10] M. Ingstrup and K. M. Hansen, "Palpable assemblies: Dynamic composition for ubiquitous computing," in *Proceedings of the seventeenth international conference on software and knowledge engineering*, 2005.

[11] C. Kappler, P. Mendes, C. Prehofer, P. Poyhonen, and D. Zhou, "A framework for self-organized network composition," in *Proc. of the 1st IFIP International Workshop on Autonomic Communication*, Berlin, Germany, 2004.

[12] M. Kyng, E. T. Nielsen, and M. Kristensen, "Challenges in designing interactive systems for emergency response," in *DIS '06: Proceedings of the 6th ACM conference on Designing Interactive systems*. New York, NY, USA: ACM Press, 2006, pp. 301–310.

[13] E. T. N. Margit Kristensen, Morten Kyng, "IT support for healthcare professionals acting in major incidents," in *3rd Scandinavian conference on Health Informatics*, 2005, pp. 37–41.

[14] J. Strassner, "Seamless mobility - a compelling blend of ubiquitous and autonomic computing," in *Dagstuhl Workshop on Autonomic Networking*, 2006.

[15] U. Varshney and S. Sneha, "Patient monitoring using ad hoc wireless networks: reliability and power management," *IEEE Communications Magazine*, vol. 44, pp. 49–55, 2006.

[16] J. M. Vlissides, J. O. Coplien, and N. L. Kerth, *Pattern languages of program design 2*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[17] R. J. Wieringa and J.-J. C. Meyer, *Applications of deontic logic in computer science: a concise overview*. Chichester, UK, UK: John Wiley and Sons Ltd., 1993, pp. 17–40.

[18] M. Wooldridge, *An introduction to multiagent systems*. John Wiley & Sons, June 2002.