

Autonomic Communication with RASCAL Hybrid Connectivity Management

Dominic Greenwood and Roberto Ghizzioli

Abstract. This paper presents an approach to manipulating available hybrid connectivity to autonomically maximize the potential for sustained connectivity in the event of path disruptions. The approach is documented in terms of the features, architecture and deployment modes of an autonomic communications module, termed RASCAL. This module employs software-agent logic supported by a state-of-the-art policy engine to dynamically determine best options for packet transmission over available infrastructure and ad-hoc connections.

Keywords. autonomic, hybrid, ad-hoc, contingency, policy, agent.

1. Introduction

The ability to seamlessly communicate when mobile is now, for many, an inescapable component of day-to-day life. It is of course the electronic communications revolution has brought about this reality; one where in many respects we simply cannot perform many common tasks without access to communicative devices including cellphones, PDAs, laptops, and GPS. This fact is especially resonant in environments where communication is critical to sustaining coordination between individuals that need to remain *always-best-connected* anywhere, anytime, using any available network technology and with the maximum quality and capacity on offer.

We consider key examples of such environments to include those where human life is a critical concern, such as sites of natural disasters (e.g., tsunamis, hurricanes, earthquakes, floods, forest fires, etc.), major incidents [8, 13] (e.g., plane crashes,

To be published in Calisti, M., Meer, S. and Strassner, J. (eds.) <i>Advanced Autonomic Networking and Communication</i> - Whitestein Series in Software Agent Technologies and Autonomic Computing.
--

multi-vehicle road traffic accidents, building fires, etc.), and theaters of military operations. In all of these there is very often a pressing need to communicate information between individuals, whether they be in either localized, or widely distributed groups. A straightforward example is the coordination of ad-hoc teams of rescue workers that need to share multiple forms of information (i.e., audio, video, sensor data, medical data, etc.) while operating in the field.

With this work we thus aim to address the problem of maximizing the assurance that communication will remain established even when communicative channels are disrupted due to environmental events. Specifically, we propose one contribution to solving this problem: an autonomic communication system providing real-time, secure, bidirectional communication of data messages from source to destination(s) while remaining agnostic to the devices, networks or carriers required to transfer the information. The reported technology prototype draws on concepts defined by many researchers and practitioners in the field of autonomic communication [1, 12].

The prototype is termed RASCAL¹ (*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. This requires some migration of message handling intelligence into user devices to allow iterative delivery decision-making throughout the communication route. RASCAL thus shifts the burden of tasks such as configuration, maintenance and fault management from users to a specialised self-regulating subsystem. A local policy engine is used by each RASCAL deployment for *self-configuration* purposes, allowing autonomic adjustment of behaviour in accordance with environmental changes. RASCAL is also *self-optimizing* as it monitors network resources and adapts its behavior to meet the end-user and application service needs, i.e., automatically handing-over sustained sessions 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.

The remainder of this paper is organized as follows: section 2 discusses some relevant previous work on autonomic communication. Section 3 then presents the autonomic features of RASCAL and Section 4 presents the RASCAL architecture. Section 5 presents some initial results from laboratory-based experimentation and Section 6 illustrates a real-life scenario within which RASCAL has been evaluated. The paper is concluded in Section 7.

¹RASCAL has been designed and implemented as a deliverable of the European Union 6th Framework Program Palpable Computing project (PalCom) - IST-002057.

2. Related Work

As mentioned in the previous section, there are a variety of published studies available regarding the deployment of autonomic communication and network management systems in disruptive environments. The majority of these tend to consider quite specific aspects of the domain. For example, when addressing the networking aspect most papers focus on either infrastructure or on ad-hoc networks (MANET) without considering the coordinated use of both concurrently. A specific case in point is the work of Chadha *et al.* [20] 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. In particular, this system only addresses mobile ad-hoc networks without consideration of potentially available infrastructure networks.

Those papers that deal with hybrid networks consisting of a combined infrastructure and mobile ad-hoc connectivity, few then consider either the requirements and influence of the user applications running over the autonomic communications subsystem, or the roles of end-users in deployment scenarios.

A good example of hybrid network management is provided in Hauge *et al.* [21] who present an interesting approach to the combined use 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* [3] chartered as part of the Internet Research Task Force (IRTF). This group primarily 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 3).

Another important work in this area is that of Kappler *et al.* [22] who use a policy-engine to address hybrid network composition. 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 the possibility of user-level service composition.

The notion of Unified Messaging (UM), as reported in van der Meer *et al.* [24] for example, is also closely related to our work from the perspective of supporting both fixed and mobile users with universal access to communication services. The central concept of UM is the capability of the messaging system to select the most appropriate terminal or application for an incoming message according to availability, status, and other parameters. A UM system is designed to adapt terminals to different kinds of services via content adaptation processes guided by user rule policies. However, the particular approach documented in [24] identifies CORBA as a suitable means of engineering the middleware software for handling

UM; an approach, in our opinion, not entirely suitable for application in pervasive environments where seamless mobility is a paramount issue and small footprint devices are the norm, as is particularly the case in disruptive environments.

Additionally, published work relating to the use of ambient and pervasive technologies in disruptive environments or disaster management tends to focus more on service composition or other application-oriented aspects without considering the underlying networks issues. Such an example is the reported work of Kristensen *et al.* [13] which focuses on IT support in major incidents, such as the use of bio-monitors, patient identification and collaboration tools for response units, without considering how these applications could, or should, behave in the presence of 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.

3. The Autonomic Features of RASCAL

The RASCAL software system is a middleware communication layer offering vertical interfaces to communicative applications (typically user-driven) and to low-level network bearer modules. The purpose of the layer is to intercept all, or a selection of, application-specific messages passing through the local communication subsystem of a device in accordance with a set of dynamically configurable policies defining the prioritized actions to take regarding forward routing of the messages. These policies are thus used to guide the autonomic decisions that the RASCAL autonomic controller can take in response to events sensed from the environment. The most straightforward example is the detected failure of an infrastructure connection (say WLAN), whereby a predefined obligation policy may mandate that RASCAL re-route messages via an alternative network technology (say Bluetooth) to either their final destination or another node with an active infrastructure connection. An example of such a policy is described in Section 6.

This technology thus improves the potential for communicative applications to remain connected when deployed in environments subject to disruptive behavior. When RASCAL is deployed within the local communications stack of a device, we term the device as having become "*RASCALized*".

The primary features of RASCAL fall into two classes: *connection-aware* and *usage-aware* communication.

Connection-aware communication implies the ability of RASCAL to be aware of all available (active and inactive) network connections, their parameterization and performance characteristics (e.g., Quality of Service (QoS) characteristics). This set of autonomic behaviors are triggered by changes in network resources, with some of the most significant operations being:

- **Network handover** : The capability of dynamically 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 infrastructure or ad-hoc networks and on network 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 local neighborhood.

- **Routing optimization** : The capability of 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 the proactive evaluation of the transmission delay between two interoperating devices and consequently the use of an alternative path to reach the same target node.

Other behaviors within this category include those acting in response to fail-overs or high network load, etc.

As mentioned, 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 rescue workers are on the site of a major accident with human casualties and must constantly maintain communication with both with one another and with a response unit concerning their findings and the positions of injured people. Due to the potentially disruptive nature of the environment network availability may be intermittent, but the goal to reliably deliver communication must persist. In this scenario the typical autonomic decisions to be taken include:

- **Transmission contingency**: The capability of providing alternatives to the default means of transmitting a message. This specifically includes making the best possible use of multi-technology transmission paths including cellular networks, IP infrastructure networks, satellite systems, MANETs, etc. An important parameter in 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 such as the number and the importance of the messages/streams to be sent. Examples include applying a codec to reduce the bandwidth consumed by 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 volume and the significance 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 manner. 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).

All of these capabilities are controlled by user-definable policies that may be simply and rapidly specified/modified on-the-fly by either the user or local/remote automated routines.

Additionally, one other important feature of RASCAL is the notification, inspection and control interface available to the end user. This GUI specifically allows the user to remain aware of decisions made by the autonomic controller and to affect them if necessary. More details on this are provided in Section 4.4.

4. The Architecture of RASCAL

The RASCAL software architecture is depicted in Figure 1 and is specifically designed to be compliant with the standard autonomic control loop [2], in that it:

- *collects* sensory information from the system and the external world,
- *decides actions* using a policy engine,
- and *effects* those decisions to affect system behavior.

The architecture consists of three managed elements, which are in fact software services: an applications service, a networks service and policy service, all of which will be described later. Also present is the core decision control logic in the shape of an autonomic manager (RASCAL software agent), a policy engine and the user interface control.

The RASCAL software is designed to interact with different layers of the user-device communication stack. In the default case it interacts with lower-layer communications components dealing with bearer technologies, e.g., UDP, Bluetooth, etc., and with upper-layer components dealing with, for example, routing protocols, discovery, and services/applications. Other components may also interface to RASCAL through provided software interfaces.

As we believe a highly beneficial means of implementing the control logic of an autonomic manager is with a software agent, we elected to design the RASCAL architecture for implementation as a JADE (Java Agent Development Framework) software system. JADE ² is an open source software agent platform and development environment [4]. Thus the RASCAL autonomic manager is designed for deployment as a JADE agent and the managed element interfaces are deployed as JADE kernel services, both of which are executable within the JADE runtime. In general terms a *software agent* is defined by Wooldridge [15] as a “*computer*

²JADE documentation and software is available from <http://jade.tilab.com/>

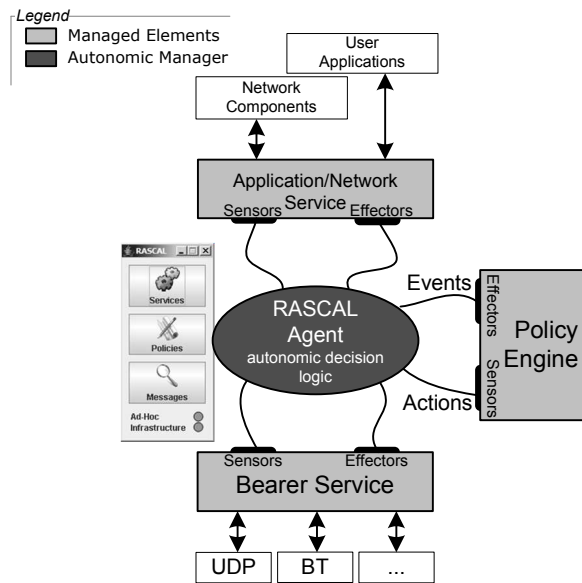


FIGURE 1. The RASCAL software architecture.

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.* [4] as a "software component which implements platform level features that can be grouped together according to their conceptual cohesion".

4.1. The Managed Elements

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 [14]. 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 [5].

The RASCAL system contains three JADE kernel services:

The Network/Application Service. This is used by the RASCAL agent to interact with upper-layer network services, such as routing and discovery, and application services facing the user. Messages passing through this interface may be inspected by RASCAL to determine whether any action is necessary by the autonomic manager in accordance with specified policies. A selection of interfaces are available allowing communication with a broad range of network services and applications.

The Bearer Service. This is used by the RASCAL agent to interact with the lower-layer bearer interfaces to both infrastructure and ad-hoc network endpoints.

Messages passing through this interface may be inspected by RASCAL to determine whether any action is necessary by the autonomic manager in accordance with specified policies. A selection of interfaces are available for a broad range of network technologies.

The Policy Service. This is used by the RASCAL agent to interact with the local policy engine (see Section 4.3). In brief, this engine possesses the operational rules that must be applied to control (or not control) the way in which messages are treated by RASCAL. The RASCAL agent uses these policy rules to effect this control.

4.2. The Autonomic Manager

This is the RASCAL software agent that controls the RASCAL system such that it exhibits the following autonomic behaviours:

Sensing. By installing sensors in the managed elements the agent is able to monitor 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. Intercepted messages and received events are used to create an internalized model of the external world. This compiled knowledge base contains information such as statistical flow data, historical fault logs, active and treated faults, discovered devices and the services those devices provide.

Decision Control. Whenever the internal knowledge base is updated, the RASCAL agent triggers a call to the local policy engine which dictates the policy constraints that must guide decision-making by the agent. This aspect is discussed in more detail in Section 4.3.

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

4.3. 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 managed element 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 2). 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 managed element, shown in Figure 1, 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 Ponder2³ [16] citerussello,

³see <http://ponder2.net/>

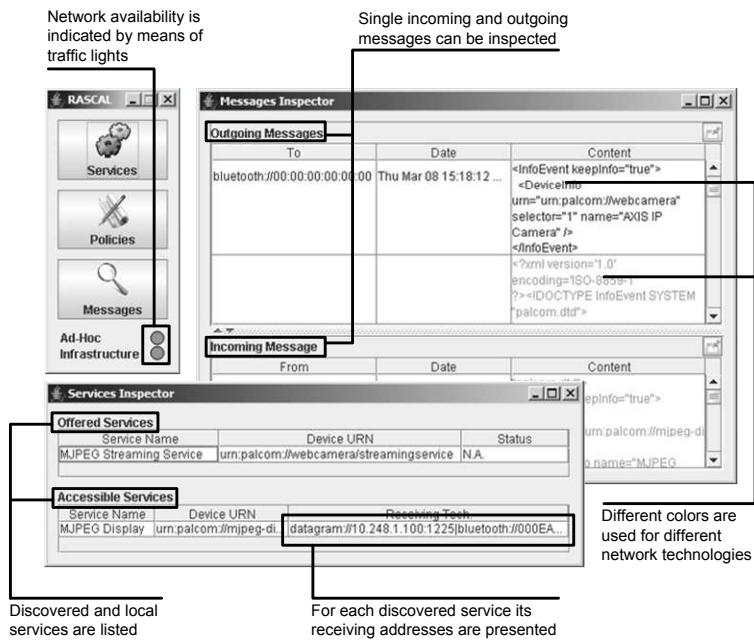


FIGURE 2. The RASCAL GUI

used as a local library, which uses an XML-based policy description 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 [18]. Given E , and C is true, it is obligatory that the agent performs A .

4.4. 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 2, 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.

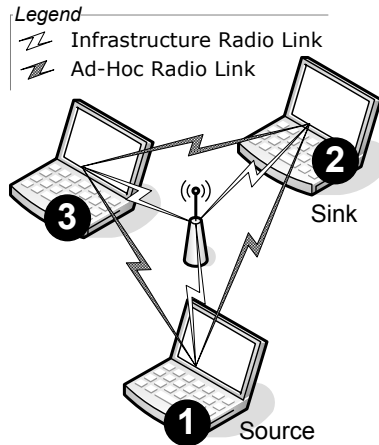


FIGURE 3. Laboratory experimentation setup.

5. Laboratory Experimentation

This section presents preliminary laboratory experiments conducted in order to validate RASCAL capabilities in terms of ensuring connectivity remains established even when some channels are disrupted.

The experimental setup for this evaluation was composed of three laptop nodes, each equipped with WLAN and Bluetooth adapters. Each node could communicate with the others using either of the two available network technologies (see Figure 3).

Each node was also equipped with the PalCom communication stack¹, which provided discovery services and multi-hop routing capabilities via the DSDV [23] routing algorithm. Selected test applications were deployed on top of the PalCom communication stack: The application running on node-1 (the source) was to send heartbeat messages every 3 seconds to node-2 (the sink), once it had been discovered. The application running on node-2 was capable of receiving and counting incoming messages. In this scenario, node-1 could reach node-2 directly, or via node-3. It could also occur that messages could be transferred via hops across different bearers.

To add uncertainty to the experiments, aperiodic network failures were simulated. Each node was equipped with a failure generator which, based on a mathematical model, blocked the transmission of messages over a particular network adapter for a certain time. The mathematical model was based on the *Markovian property* that the probability of the occurrence of an event does not depend on the history of previous events. Based on this property, technology failures were simulated with an occurrence rate equal to the inverse of the λ parameter of a *negative exponential* distribution. Furthermore, the duration of the failure was simulated

TABLE 1. Table of average stochastic failure occurrence time and duration per network bearer technology

Technology	Rate (mins)	Duration (mins)
UDP	2.5	2 ± 1
Bluetooth	1.25	2 ± 1

using the *Erlang-k* distribution. The expected average and standard deviation failure duration time were used to define the distribution. Table 1 shows the average failure occurrence time and the relative duration for each bearer endpoint. The choice of these values was based on experience gained when performing real-world evaluations (see Section 6).

In the experiment, every node was also equipped with a software component named *Failure Generator* which simulated the unavailability of a particular network bearer. In particular, using the parameters presented in Table 1 the component generated failure events that described when and for how long a network adapter had to be considered deactivated. When the event occurred, the failure was simulated and the node prevented from sending messages using that particular adapter. Each time an event was consumed, a new one was immediately generated. The failure generator reactivated an adapter when the failure duration time elapsed.

The entire experiment consisted of 20 runs of 10 minutes each. Every experiment contained a stochastic number of failure events.

To measure the capabilities of RASCAL to ensure that communication remained established even when channels were disrupted, experiments were conducted both with and without the RASCAL component deployed. A node without RASCAL was only capable of communicating using a UDP bearer, with no ability to autonomically adapt. The measured output variable was the *number of messages successfully delivered to the sink* (node-2).

5.1. Results

Figure 4 shows the value of the measured output variable over 10 minutes when analyzing a single experimental instance. As can be observed, after the 10 minute cycle the number of messages delivered with RASCAL enabled was substantially higher than without it. In fact, during this period of time several network failures occurred, but nodes with RASCAL deployed continued to discover one another with heartbeat messages sent continuously using different routes. On the other hand, with nodes without RASCAL deployed when a failure occurred the sink was no longer discovered and no heartbeat messages were therefore transmitted from the source to the sink (this is represented by the flat parts of the dotted line in Figure 4).

A more significant result is given by Figure 5 which shows the value of the output variable over the all 20 considered instances. This box-plot shows that the average number of messages successfully delivered to the sink when RASCAL was

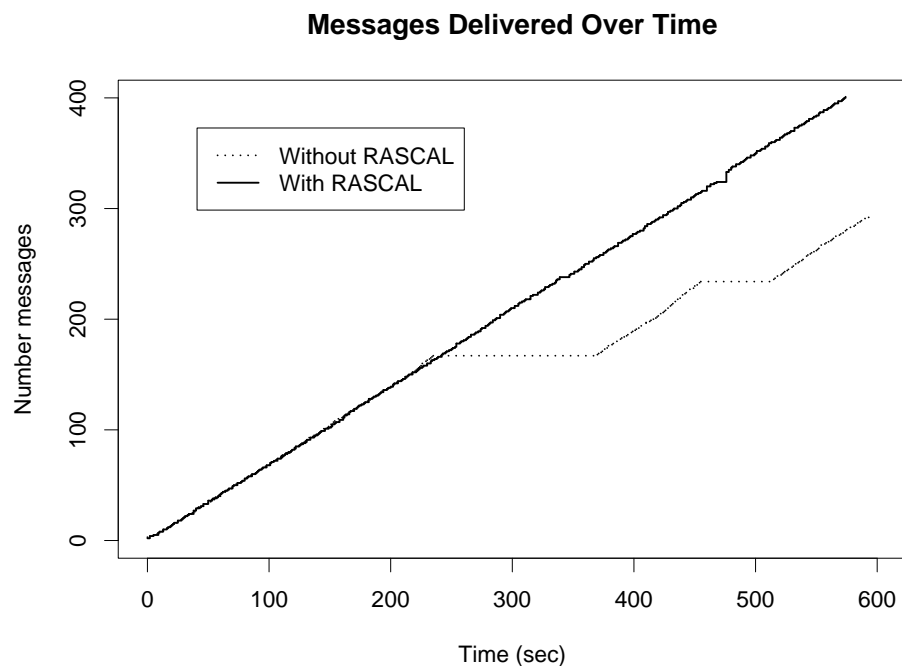


FIGURE 4. Number of messages successfully delivered to the sink in an experimental instance.

deployed, were definitely better than without it (339 messages against 267). Naturally, for simple instances with no failures the presence of RASCAL was irrelevant. In fact in both cases the maximum number of delivered messages is almost the same. The results change however when many failures occurred since the minimum of this output variable over 20 instances without RASCAL is 110 and with RASCAL is 221, i.e., system performance improved by more than 100%. This allows us to conclude that the more disruption occurs to the network infrastructure, the more benefit is provided by the presence of RASCAL technology - thanks to its capability of utilizing alternative paths over different technologies to deliver messages.

To be certain of the real significance of the obtained results over 20 instances the Wilcoxon Paired Rank Sum Test was applied. This test stated with a confidence level higher than 95% (p-value = 0.0008909) that the improvements generated by RASCAL are statistically significant.

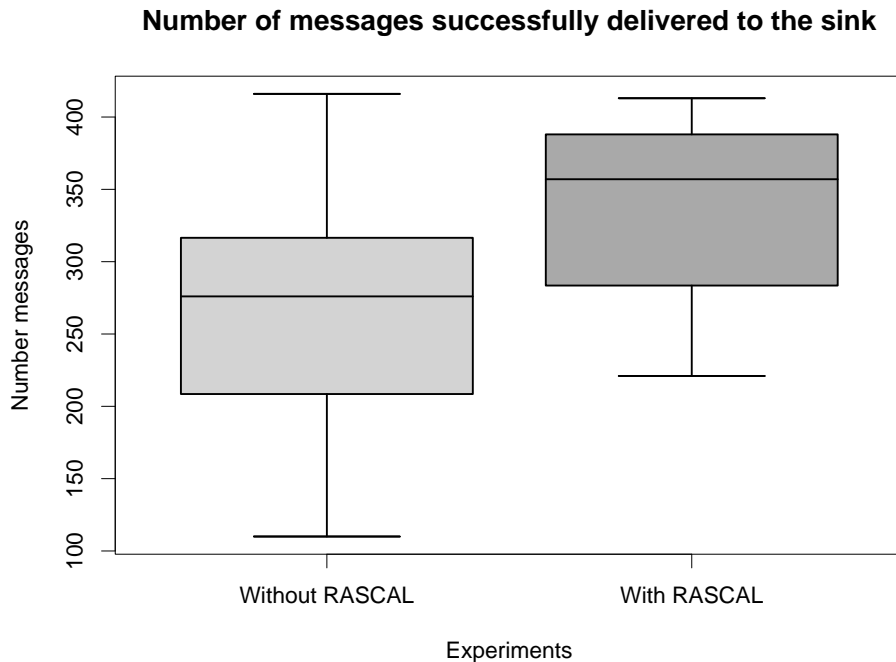


FIGURE 5. This box-plot represents the number of messages successfully delivered over the 20 experimental instances, with and without RASCAL.

6. Real World Evaluation

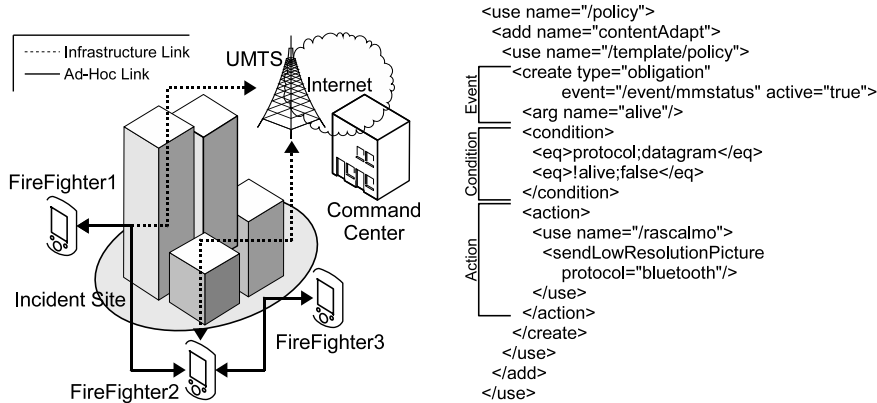
RASCAL has recently been integrated into the iterative, participatory design process practiced in the PalCom project¹. We are currently carrying out experiments with end users in major incident emergency response scenarios. This section presents an overview of one of the mocked-up situations of a real world major incident conducted recently.

Within this experiment a building fire is considered. This particular scenario demands fast and decisive action, often in life-threatening situations. It also requires collaboration between numerous people located in different, often changing areas: personnel at the incident site including firefighters, police, and paramedics, at the command center, in vehicles, and others.

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.

The particular scenario is illustrated by Figure 6(a). Three firefighters (FF) are moving in relative proximity to one another attempting to evacuate people from a building on fire. They are using small, wearable "RASCALized" PDAs, each with a built-in camera running a map service. Their duty is to notify the command center (CC) and the other local team members of findings related to visited building(s), i.e., the positions of injured people. To do this, they make special marks on the map displayed on their PDA. They may also take pictures to assist the command center with gaining a visual overview of the incident location and status.



(a) The conducted mocked-up situation.

(b) 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.

FIGURE 6. Real World Scenario.

In terms of connectivity, 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 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.

In this particular scenario, 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.
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 6(b) 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 whether a network interface is available or not. This event has two associated parameters: the interface protocol and its current status. The only condition that triggers this policy is that the infrastructure connection is unavailable (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 this scenario, FF1 has policies (1) and (3) activated. When FF1 moves into an area where the infrastructure connection fails, this is detected and policies (2) and (4) automatically become active. The RASCAL agent running on the FF1's PDA 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 ad-hoc channel, 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 pictures to the command center is shown in Figure 7. The diagram considers the situation both with and without the infrastructure connection.

Further real world experimentation with emergency services is ongoing.

7. Conclusion

This paper has provided an overview of the RASCAL autonomic communications software component designed to manage connectivity in environments subject to disruptive events. The component can be interfaced with device communication stacks and offers features including automated network handover, routing optimization, transmission contingency, content adaptation, deferred service provisioning and role management.

The key components of each local RASCAL deployment consist of a software agent autonomic control logic and a Ponder2 policy engine with which the user is able to define rules guiding and constraining the agent control logic. The autonomic controller interfaces with three (or more) managed element services, of which the policy engine is one.

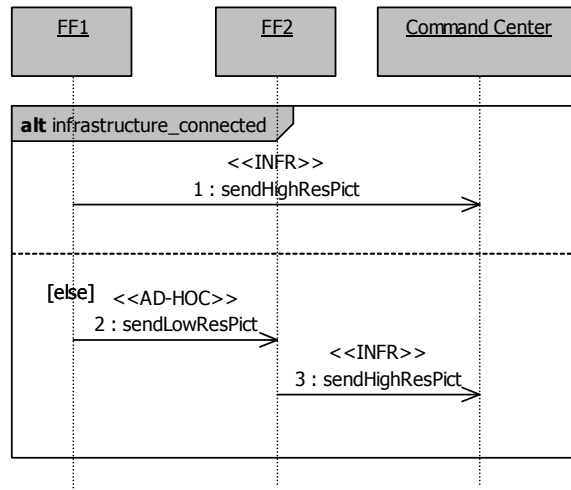


FIGURE 7. Sequence diagram of the actions taken by FF1 to send pictures to the command center.

The reported laboratory experimentation has demonstrated the operation and performance of the system under controlled conditions, with results proving the intuitive conclusion that RASCAL significantly aids the maintenance of sustained connectivity in disruptive environments. Moreover, the reported real world evaluation offers an insight into how RASCAL has been deployed in an actual setting involving firefighters working collaboratively at an incident site.

As a work in progress, RASCAL remains under a continuous refinement, in particular as feedback from real world evaluations is gathered. Particular ongoing work includes areas that enhance the autonomic capabilities of the system including the incorporation of improvements to the Ponder2 policy language (undertaken as an independent project to RASCAL), and to the autonomic controller logic. The improvements to the policy language include a new means of expression using a form of process-algebra over actions. This allows the expression of a set of actions (i.e., workflow) rather than the current limitation to atomic actions. Further 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.

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. **42:1**, 2003, 165–176.
- [2] IBM *An architectural blueprint for autonomic computing*. 2006.

- [3] S. Farrell and V. Cahill *Delay and Disruption Tolerant Networking*. Artech House Publishers, 2006.
- [4] F. L. Bellifemine, G. Caire and D. Greenwood *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, 2007.
- [5] J. M. Vlissides, J. O. Coplien and N. L. Kerth *Pattern languages of program design 2*. Addison-Wesley Longman Publishing Co., 1996.
- [6] F. Hu and S. Kumar *The integration of ad hoc sensor and cellular networks for multi-class data transmission*. Elsevier Ad-Hoc Networks Journal, **4:2**, 2006, 254–282.
- [7] U. Varshney and S. Sneha *Patient monitoring using ad hoc wireless networks: reliability and power management*. IEEE Communications Magazine, **44**, 2006, 49–55.
- [8] M. Kyng, E. T. Nielsen and M. Kristensen *Challenges in designing interactive systems for emergency response*. DIS '06: Proceedings of the 6th ACM conference on Designing Interactive systems, 2006, 301–310.
- [9] Z. Obrenovic, D. Starcevic, E. Jovanov and V. Radivojevic *An agent based framework for virtual medical devices*. AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, 2002, 659–660.
- [10] M. Ingstrup and K. M. Hansen *Palpable assemblies: Dynamic composition for ubiquitous computing*. Proceedings of the seventeenth international conference on software and knowledge engineering, 2005.
- [11] D. Greenwood and M. Calisti *The Living Systems Connection Agent: Seamless Mobility at Work* Communication in Distributed Systems (KiVS 07), 2007.
- [12] J. Strassner *Seamless Mobility - A Compelling Blend of Ubiquitous and Autonomic Computing*. Dagstuhl Workshop on Autonomic Networking, 2006.
- [13] M. Kristensen, M. Kyng, E. T. Nielsen *IT support for healthcare professionals acting in major incidents*. 3rd Scandinavian conference on Health Informatics, 2005, 37–41.
- [14] Gamma, Erich and Helm, Richard and Johnson, Ralph and Vlissides, John *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.
- [15] M. Wooldridge *An introduction to multiagent systems*. John Wiley & Sons, 2002.
- [16] N. Damianou, N. Dulay, E. Lupu and M. Sloman *The Ponder Policy Specification Language*. POLICY '01: Proceedings of the International Workshop on Policies for Distributed Systems and Networks, Springer-Verlag, 2001, 18–38.
- [17] G. Russello, C. Dong, and N. Dulay *Authorisation and Conflict Resolution for Hierarchical Domains*. IEEE Workshop on Policies for Distributed Systems and Networks, Bologna, Italy, 2007.
- [18] R. J. Wieringa and J.-J. Ch. Meyer *Applications of deontic logic in computer science: a concise overview*. Deontic logic in computer science: normative system specification, John Wiley and Sons Ltd., 1993, 17–40.
- [19] D. Siegrist *Advanced information technology to counter biological terrorism*. SIGBIO Newsl., **20:2**, 2000, 2–7.
- [20] 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, 2004, 35–44.

- [21] M. Hauge and O. Kure *Multicast Service Availability in a Hybrid 3G-cellular and Ad Hoc Network*. International Workshop on Wireless Ad-Hoc Networks, 2004.
- [22] C. Kappler, P. Mendes, C. Prehofer, P. Poyhonen and D. Zhou *A Framework for Self-organized Network Composition*. Proc. of the 1st IFIP International Workshop on Autonomic Communication, 2004.
- [23] C. E. Perkins and P. Bhagwat *Highly dynamic Destination-Sequenced Distance-Vector routing (DSDV) for mobile computers* ACM SIGCOMM Computer Communication Review, **24:4**, 1994, 234–244.
- [24] S. van der Meer, S. Arbanowski and, T. Magedanz *An Approach for a 4th Generation Messaging System*. Proc. of 4th IEEE Symposium on Computers and Communications (ISCC'99), 1999, 156–163.

Acknowledgment

The authors acknowledge the EU PalCom:Palpable Computing (IST-002057) FET project, which contributed funding toward development of the RASCAL project. This acknowledgment includes those PalCom consortium members that contributed toward this work including David Svensson from Lund University, Sweden, and Jacob Mahler-Andersen, Jacob Frølund, Henrik Gammelmark and Michael Christensen from Aarhus University, Denmark. We would also like to thank our colleagues at Whitestein Technologies, Monique Calisti, Giovanni Rimassa, Thomas Lozza and Stefan Thurnherr for their contributions to the work.

Dominic Greenwood
Whitestein Technologies AG
Zürich, Switzerland
e-mail: dgr@whitestein.com

Roberto Ghizzioli
Whitestein Technologies AG
Zürich, Switzerland
e-mail: rg@whitestein.com