AutoHome: an Autonomic Management Framework for Pervasive Home Applications

J. BOURCIER
University of Grenoble and Imperial College, London, UK
AND
A. DIACONESCU AND P.LALANDA
University of Grenoble, Grenoble, France
AND
J. A. MCCANN
Imperial College, London, UK

This paper introduces the design of the AutoHome service oriented framework to simplify the development and runtime adaptive support of autonomic pervasive applications. To this end, we describe our novel open infrastructure for building and executing home applications. This includes the amalgamation of the two computing areas of Autonomics and Service Orientation, to produce a Component-based platform providing facilities including monitoring, touchpoints and other common autonomic services. This infrastructure uniquely blends the advantages of distributed autonomic control with global conflict management in a management hierarchy. We discuss this platform in terms of pervasive home systems and show how one would develop such a system for two examples of automated home applications: intruder detection and medical support respectively. Both applications were built within our framework and evaluated showing that the use of the framework introduces minimal overheads but provides many benefits. We then conclude by highlighting the contributions of AutoHome and a discussion about the lessons learned, limitations and future research directions.

Categories and Subject Descriptors: D.2.13 **[Software]** – Reusable Software -– Domain engineering.

General Terms: Design, Performance, Experimentation, Reliability.

Additional Key Words and Phrases: Autonomic Computing, Service Oriented Computing, Pervasive computing, Framework

ACM File Format:

1. INTRODUCTION

Recent evolutions in embedded technologies are bringing about a proliferation of ubiquitous computing devices. Pervasive computing [1] capitalizes on such capabilities to provide a wide variety of innovative applications, which are progressively changing the way we interact with our environments. At the same time, several important scientific and technical challenges remain to be solved before fulfilling the pervasive computing vision. There is no doubt that provisioning high-level services built on top of heterogeneous, distributed, dynamic devices and applications is a difficult task.

Authors' addresses: J. Bourcier and Julie A. McCann Department of Computing, Imperial College, 180 Queens Gate, LONDON, SW7 2BZ, UK. E-mail: jbourcie@doc.ic.ac.uk, jamm@doc.ic.ac.uk; A. Diaconescu and P.Lalanda, Laboratoire LIG, Equipe ADELE, Bat. C, 220 rue de la Chimie, Domaine Universitaire, BP 53, 38041 Grenoble Cedex 9, France; E-mail : johann.bourcier@imag.fr, philippe.lalanda@imag.fr, ada.diaconescu@imag.fr

In previous work [2, 3] we proposed a software platform simplifying the development and execution of pervasive home applications. It was based on a service-oriented approach, providing modularization, loose-coupling and dynamic properties. Although this platform supported system adaptability and evolution such as dynamic service bindings, it lacked functionalities to dynamically monitor and reconfigure each service as well as the facilities for enforcing and controlling dynamic modifications that maintain global system coherence to adherence to goals. Indeed, managing complex, adaptive pervasive applications is no easy task.

Autonomic Computing [4] promises a solution to the aforementioned problem, by endowing software systems with self-management capabilities that would minimize or eliminate the need for human intervention. If successfully implemented, autonomic pervasive applications would inherently feature critical properties such as safety (including fault-tolerance and security) and self-adaptation to internal and external changes (including self-configuration, self-optimization and self-repair).

This paper presents AutoHome, a project for providing reusable support for the development of viable Autonomic Management (AM) applications in the pervasive computing domain. The main contributions of this project are:

- The design, development and open-source contribution of a specific service-oriented component framework for autonomic applications. This proposition has been implemented as specific iPOJO [5] extensions enabling dynamic service monitoring and reconfigurations (autonomic touchpoints).
- The design and development of an open and dynamic hierarchical autonomic management infrastructure to enable the creation of autonomic solutions in the pervasive home environment.
- The experimentation and validation of the proposed framework through collaboration with industrials within the ITEA ANSO (http://anso.vtt.fi/) and OSAMI COMMONS (http://www.osami-commons.org/) projects.

2. BACKGROUND & MOTIVATING EXAMPLES

Home automation [6] corresponds to a subset of pervasive applications that deals with the automation of home device control. Several application areas are covered by home automation ranging from the monitoring of convalescents at home, to home theater and energy consumption control. This work is evaluated using two such application scenarios: a security-oriented application and a patient monitoring system.

The purpose of the security system is to monitor home intrusion through the use of a webcam and motion detector. The application structure is made of several services. When particular motion patterns are recognized through the image stream, an alert is sent according to user preferences and the current availability of communication mediums. The second application uses medical sensors to gather patient data. Such sensors include blood pressure sensors, heart-rate monitors or thermometers. This application is composed of several services ensuring crisis situation detection and daily production of health report. These two applications' structure is presented in appendix A.

The inherent nature of these applications requires continuous execution and maximum quality of service. Therefore, dynamic adaptation to their current environment is essential to ensure that these applications remain within the functional boundaries defined by application designers, as well as maintain their performance, security and safety properties. Thus, a framework supporting the development of such applications has to support dynamic monitoring and reconfiguration of running applications, dynamic integration of new applications and inter-application management.

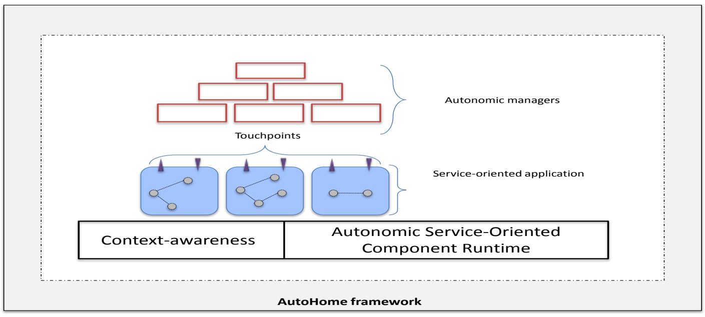
3. THE AUTOHOME AUTONOMIC FRAMEWORK

A review of autonomic frameworks enabling dynamic adaptation to their execution context is presented in Appendix B.

3.1 Overall Architecture

Our proposal is to build a middleware serving as a framework to host autonomic home applications. Our approach is to separate the design and development of the application itself from autonomic management components. The goal is thus to enable the development of autonomic management functions, seamlessly integrate them with the applications and finally integrate the resulting autonomic application with other applications sharing the same execution environments. Typically an application on top of AutoHome, illustrated in Figure 1, has the following architectural elements:

- A middleware for autonomic service-oriented applications composed of an autonomic service-oriented component and a context facility. The runtime includes a set of generic and configurable touchpoints allowing dynamic application monitoring and reconfiguration.
- A set of Service-Oriented applications built on top the middleware, which represent pervasive components to be autonomously managed.
- A set of autonomic managers organized in a hierarchy.


Autonomic managers
Touchpoints
Service-oriented application
Context-awareness
Autonomic Service-Oriented Component Runtime
AutoHome framework
Figure 1. An application on top of the AutoHome framework.

As a reminder, Service-Oriented Computing (SOC) [11] is a recent trend in software engineering that uses services as basic elements for building applications. A service represents a computational entity described by a specification covering both functional (service interface) and non-functional (QoS) aspects. At runtime, available services are registered in one or more service brokers where they can be discovered by service consumers. A service consumer is then able to invoke the service based on the service specification. An important consequence of this interaction pattern is that SOC technologies support dynamic service discovery and lazy inter-service binding. Such characteristics are essential when building applications with strong adaptability requirements, such as pervasive and residential applications.

iPOJO [5] is the Apache service oriented component runtime built on top of OSGi [12] SOA Platforms. The iPOJO framework merges the advantages of component and service oriented paradigms. Specifically, application functionalities are implemented following the component paradigm. Each component is fully encapsulated, self-sufficient and provides server and client interfaces as services. An iPOJO component is actually managed by a reusable container which provides common middleware functionalities. Each component container can be configured with a different set of middleware services.

3.2 A Hierarchical Management Architecture

As rapidly introduced, autonomic managers are organizes in a hierarchy. This hierarchy relates to two aspects: *authority* and *abstraction*. A manager of a higher level has higher authority, and therefore precedence over a lower level manager. The hierarchy is also used to mask the management complexity by raising the abstraction of models used by higher managers.

These two properties provide the scalability feature to our management architecture while avoiding the conflict management issue of entirely decentralized architectures. In our system, an autonomic manager is integrated with the autonomic managers' hierarchy and the system to be managed. Inter managers communications are thus possible and aim at guarantying or modifying the behavior of non managed resources. Managers remain sovereign of their decisions on the resource they control.

Our architecture, as seen in figure 2, offers three types of autonomic managers: service managers, application managers and a gateway manager. Service managers finely control the internal behavior of the service and optimize its functionalities. Application managers control the life cycle of an application and its constituent services. One gateway manager is attached to the overall framework. It governs the physical resources of the gateway and arbitrates conflicts between applications.

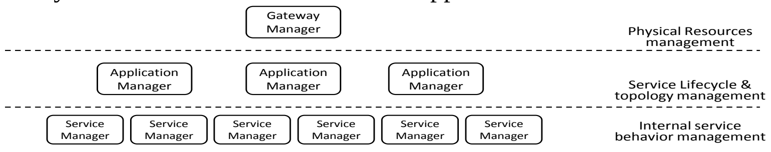


Figure 2. Global Management Architecture of the AutoHome Framework.

3.3 Middleware's Touchpoints

Our middleware is an extension of iPOJO. The use of a service-oriented approach is mainly motivated by its inherent support for plural management authority, and dynamic reconfiguration whereby a service's implementation can be replaced during runtime. Such dynamic capabilities are essential for autonomic home applications.

Our middleware offers a dedicated component model for autonomic pervasive applications by proposing a set of generic *touchpoints*, which monitor and control the execution of pervasive applications. *Touchpoints* constitute the interface between autonomic managers and running applications. Our middleware offers dedicated *touchpoints* to each type of autonomic manager presented in the previous section.

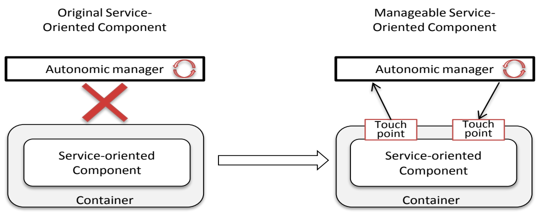

Original Service-Oriented Component
Manageable Service-Oriented Component
Autonomic manager
Autonomic manager
Touch point
Touch point
Service-oriented Component
Service-oriented Component
Container
Container

Figure 3. Integration of *touchpoints* into a Service-Oriented Component

AutoHome offers a set of generic iPOJO handlers, covering the whole set of possible monitoring and reconfiguring activities. These handlers can be independently added after the development of a service (principle illustrated in figure 3). We identify three *touchpoints* for monitoring: **property value**, **method invocation**, and **container's state**. The property value *touchpoint* monitors either actively or passively any service attribute. The method invocation *touchpoint* enables monitoring through the call of a particular method. Finally, the container's state *touchpoint* enable precise monitoring of the container's state.

We identify three actuators *touchpoints*: **property value modification**, **method invocation**, and **container's state reconfiguration**. The property value modification *touchpoint* offers the possibility of modifying the value of a property. The method invocation *touchpoint* offers the possibility to call a method in order to modify the internal service behavior. The container's state reconfiguration *touchpoint* enables the modification of the container's configuration.

Application managers have the possibility to control the life cycle of service instances with their associated autonomic manager. Thus an application manager may add, withdraw, start or stop services. Connections between services are carried out by the created instances. The particular instance configurations and their behavior adaptations are carried out by the service manager. Using these touchpoints, an application manager is able to dynamically modify applications' topology.

The gateway manager is in charge of maintaining gateway performance and attending to conflict management between applications. It is also in charge of the regulation of the memory load, processor usage and the framework bandwidth. Hence, the touchpoints available to the gateway manager allow the monitoring of physical parameters of the framework. The action touchpoints correspond to the deployment and withdrawal of new components.

4. EVALUATION – PERFORMANCE ASSESSMENT

To comparatively evaluate the performance of our framework, we have implemented two autonomic managers with and without using our framework; allowing us to evaluate the framework overhead in terms of performance. We thus compared the CPU consumption and did not find a significant difference between the two. This is mainly due to the time scale of the reconfigurations in our testing scenario. Indeed, as reconfiguration does not happen very often (one every 30-40 seconds), the overhead of the framework is not visible in the scenario's time scale. We also have evaluated the overhead in terms of memory footprint. For the security application, the overhead is between 6% and 13% for our two applications. We have also compared the number of lines of code of the autonomic management part for both applications (These applications are presented in section 2 and their implementation are presented in Appendix C). For the security application, the management part is 354 lines of code with our framework, while it is 503 lines of code without our framework. For the patient monitoring application, the management part is 118 lines of code with our framework while it is 156 without. We therefore consider that giving the low overhead and the benefit in lines of code, our framework succeeds in helping developers design autonomic applications in an open pervasive environment. More details about the evaluation can be found in Appendix C.

5. CONCLUSION

In this paper we discussed the design of AutoHome, a framework based on a service-oriented component platform simplifying the development and supporting the executions of multiple autonomic pervasive applications in parallel. The first contribution of our work is the provision of a dedicated middleware to support the execution of autonomic smart home applications through a dedicated service-oriented component model to build smart home autonomic applications. This model includes a set of touchpoints that safely and non-intrusively monitor and reconfigure services. The second contribution of our framework is the hierarchical management infrastructure ensuring consistency throughout the smart home gateway, and enabling dynamic integration of new applications.

Some parts of the presented middleware are currently used on several open source projects and in the context of European projects. The evaluation of the prototype presented in section 4 and appendix B highlights the benefits of AutoHome: reducing the complexity of building autonomic applications in pervasive environments, providing guidelines on the development of these applications, and finally providing the basic infrastructure to support the co-existence of several independent autonomic applications.

From this work, we derive three main lessons. **Hierarchical management architectures are essential** when dealing with such complex systems. Indeed, different types of reactions, time scales and concerns apply at different levels of the hierarchy. As such, policy expressions and algorithms to enforce them may largely differ between these managers. **The openness of such infrastructure is essential** to maintain users' freedom. Indeed, new features, electronic devices and applications will become progressively available, and users should have the possibility to seamlessly integrate them in their smart home environment. And finally, **the complexity of providing a reliable runtime** enabling safe monitoring and dynamic reconfiguration is often underestimated. Indeed, synchronization issues are very complex when dealing with dynamic systems.

AutoHome currently provides very limited integrated autonomic loop. Thus, we are working on providing specific autonomic loops. Specifically, we seek to improve the automatic choice between service providers based on particular needs of consumers, and to provide feedback on the perceived *trust* of available service provider. Another limitation of our framework lies in the management policy expression, which is currently done by developing autonomic managers. We are therefore planning to integrate well known policy expression, such as goal-based or utility function based policies.

7. REFERENCES

[1] M. Weiser, "The computer for the 21st century," Scientific American, pp. 66-75, September 1991.

[2] J. Bourcier, C. Escoffier, and P. Lalanda, "Implementing Home-Control Applications on Service Platform," in Consumer Communications and Networking Conference, 2007.

[3] C. Escoffier, J. Bourcier, and P. Lalanda, "Towards a home application server," in IEEE Consumer Communications and Networking Conference (CCNC'08), Las Vegas, 2008.

[4] J. Kephart and D. Chess, "The Vision of Autonomic Computing," Computer, vol. 36, pp. 41-50, 2003.

[5] C. Escoffier, R. S. Hall, and P. Lalanda, "iPOJO An extensible service-oriented component framework," in IEEE Service Computing Conference, Salt Lake City, 2007.

[6] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, E. Jansen,"The Gator Tech Smart House: A Programmable Pervasive Space," Computer, vol. 38, no. 3, pp. 50-60, Mar., 2005.

[7] M. C. Huebscher and J. A. McCann, "A survey of Autonomic Computing—Degrees, Models and Applications," ACM Computing Surveys, vol. 40, August 2008

[8] S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao, "AUTONOMIA: an autonomic computing environment," in IEEE International Performance, Computing, and Communications Conference, 2003.

[9] M. Parashar, H. Liu, Z. Li, V. Matossian, C. Schmidt, G. Zhang, and S. Hariri, "AutoMate: Enabling Autonomic Grid Applications," Cluster Computing: The Journal of Networks, Software Tools, and Applications, Special Issue on Autonomic Computing, vol. 9, p. 14, 2006.

[10] D. Garlan, S. W. Cheng, A. C. Huang, B. Schmerl, and P. Steenkiste, "Rainbow: architecture-based self-adaptation with reusable infrastructure," Computer, vol. 37, pp. 46-54, 2004.

[11] M. P. Papazoglou and D. Georgakopoulos. Service-oriented computing. Commun. ACM, 46:24–28, 2003.

[12] OSGi Alliance. "OSGi Service Platform Core Specification Release 4", http://www.osgi.org, August 2005.

[13] J. A. McCann, M.C. Huebscher "Evaluation Issues in Autonomic Computing", International (AAC-GEVO'04), 3rd International Conference on Grid and Cooperative Computing, Wuhan, China, 2004.

APPENDIX

Appendix A: Related Work

The study of various research domains that make up the landscape of autonomic computing reveals the diversity of research conducted in this area. We note that a significant number of work build autonomic applications without using any software engineering techniques or tools. Consequently, most current approaches target specific problems and too few are interested in building reusable frameworks or tools capitalizing on the available research in this domain. This notice corresponds to one of the conclusions of [9], in which authors present a state of the art of autonomic computing and establish a set of challenges for this domain. Thus our related work focuses on comprehensive solutions for simplifying the production of autonomic applications.

IBM has produced an autonomic architecture defining a generic structure enabling the creation of autonomic applications [4]. This architecture intends to be very general and establish a structure for building autonomic applications and the decomposition of the autonomic management function. The IBM autonomic manager is composed of five distinct components: knowledge-base, monitoring, analysis, planning and execution. An autonomic manager is attached to a managed system through touchpoints allowing interaction (monitoring and actions).

This architecture is mainly a generic platform to which developers should refer but does not propose any concrete solution for developing applications. Knowledge representation is at present a very complex problem to which no generic solution exists. Another major issue concern communication between autonomic managers. Indeed, the content and methods used to exchange information between autonomous elements are not specified. Finally, this system is assuming, more or less explicitly, fully decentralized management architecture for the system which is not always possible or welcomed (as we discuss later). Therefore, solely relying on this architecture to develop autonomic applications is not sufficient, thus the more recent focus on Autonomic Frameworks.

Autonomia [10], from the University of Arizona, study a holistic approach to control and manage resources and services available in the networks, and is essentially an implementation of the IBM framework. The management architecture proposed can be either hierarchical or fully decentralized. AutoMate [11], developed at Rutgers University, covers all key technologies including programming models, platforms and middleware to develop autonomic applications for the Grid. AutoMate uses ECA rules to manage applications and extends IBM's architecture with the concept of global application level policies, which are then decomposed and interpreted by each autonomic manager. Rainbow [12] is a project developed at Carnegie Mellon University. This project aims at adapting complex systems by using an architecture-based approach. This system is based on an architecture model constantly updated through a set of probes.

These projects exemplify and complement the IBM architecture by specifying autonomic management techniques, such as policy expression, management architecture, and runtime model. Nevertheless, none of them proposes a reusable framework natively including generic touchpoints to easily plug autonomic managers. Furthermore, these projects do not consider pervasive computing specificity: that is, managing multiple independent applications with very volatile software and hardware resources.

A.1 Centralized versus Decentralized architectures

The most commonly used architecture in the literature is centralized, corresponding to a single autonomic manager for the entire system. Despite its large adoption, this architecture does not suit the home environment for many reasons. A residential gateway is intended to run a group of independent applications each containing several services. These applications can be deployed, updated or removed independently from each other. The use of centralized management requires global and detailed knowledge of all applications present on the framework and thus must be updated as soon as an application appears or is removed.

Alternatively, decentralized architectures can overcome the limitations of centralized architectures. However, their use to deal with autonomic residential applications also brings thorny issues. Even if the architecture permits integration of applications coming from different authorities, handling conflicts emerging from two contradictory decisions may become very complex. Indeed, arbitrating such conflicts may involve complex negotiation protocols, which are time consuming and hardly predictable.

A compromise between these approaches can be obtained through a hierarchical architecture. Indeed, this architecture enables different applications and service suppliers to control their own software while maintaining a global vision to handle conflicts and guarantee global properties.

Appendix B: AutoHome Evaluation

As described in [19], the evaluation of autonomic systems is a complex task and it requires the consideration of several aspects. We evaluate the AutoHome application in terms of behavior, through the implementation of the two aforementioned motivating examples presented in section 2. We assess AutoHome performance by comparing applications with and without AutoHome's autonomic extensions. All presented results were conducted on a laptop with Intel core 2 duo at 2.4GHz and 2 GB of memory.

The implementation of the security system involves two versions of the motion detector service: one is very accurate but demands a high level CPU, while the other is less accurate but demands less CPU. We have added four autonomic managers to this application. An autonomic manager for the *image retrieval service*, which has two modes of operation: restricted and unrestricted. The unrestricted mode produces images at high resolution (640*480) and frequency (5 seconds), whereas the restricted mode produces images at low resolution (320*240) and frequency (10 seconds). An autonomic manager for the *image storage service*, which again has two operation modes: normal and replacement. The normal mode adds images to a database. The replacement mode removes the oldest image and stores a new one. This manager also embeds a full autonomic control loop to provide switching between these two modes when memory thresholds are reached. An autonomic manager for the *alarm service*, is in charge of choosing the best communication medium for alerting users. This manager also takes charge of propagating emergency situation detection to application manager. An autonomic manager for the *global application* chooses between the two implementations of the motion detector. This manager is also in charge of dealing with the emergency situation by ordering all services to operate in their most accurate mode.

Figure 8 presents the curves of CPU and memory usage of this application for the following scenario. This scenario begins by injecting a high load on the processor (Label 1). The gateway manager detects the high CPU usage and asks the application manager to lower its CPU consumption. The application manager then asks the image retrieval manager to switch its operating mode to "restricted" and also changes the motion detection service implementation (Label 2). The storage service then reaches its maximum size and the manager switch its operational mode to replacement (Label 3). An alarm is then triggered and the manager consequently wants to have optimum detection conditions, meaning that the camera driver runs in "unrestricted" mode and that the motion detector service mode is at its most accurate (Label 4). The current level of load on the CPU is too high to allow this change. We reduce the injected load and observe that the reconfiguration is completed. Finally, we dump the memory and note that the storage service returns to its normal operation mode (Label 5).

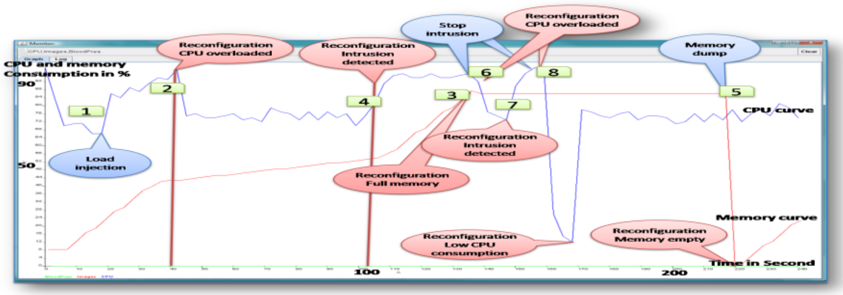


Figure 8. Results of the home monitoring application

The second application has been enhanced with three autonomic managers. An autonomic manager for a *blood pressure sensor*, which has two modes: an accurate mode with a high frequency of blood pressure sampling, and a normal mode with low frequency of blood pressure sampling. An autonomic manager for the *anomaly detector* monitors the internal state of the anomaly detector. When a problem is detected, the autonomic manager asks the blood pressure sensor manager to switch to the accurate mode. An autonomic manager for the *report creation service*, is in charge of managing the memory. When no more space is available, the manager automatically creates the daily report and sends it to the hospital thus freeing up memory.

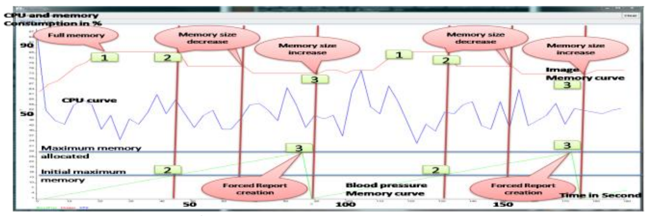


Figure 10. Results of the two combined applications

We finally evaluate the framework by running both applications concurrently showing how they conflict. The results are presented in figure 10. At the beginning, very little memory is allocated to the medical supervision application since little disk space is required. The image storage manager reaches its maximum quota and switches from normal to replacement mode (Label 1). Then, the report builder manager reaches its maximum allocated space (Label 2). It asks its application manager for more disk space; the application manager transmits the request to the gateway manager, which then reduces the disk space allocated to the intrusion monitoring application and increases the space allocated to the medical monitoring application. After that, the report builder manager reaches again its maximum disk space (Label 3), but its request is denied because the intrusion monitoring application cannot operate with a smaller disk space. Therefore, the manager triggers the creation of a report to clear memory.