An Integrated Resource Management Architecture for Wireless Smart Environments

Paolo Bellavista, Antonio Corradi, Silvia Vecchi
Dip. Elettronica, Informatica e Sistemistica - University of Bologna
Viale Risorgimento, 2 - 40136 Bologna - ITALY
Phone: +39-051-2093001; Fax: +39-051-2093073
{pbellavista, acorradi, svecchi}@ deis.unibo.it

Abstract — Pervasive and ubiquitous computing is enabling the implementation of "smart environments", i.e., environments where applications support and enhance the abilities of their occupants in executing tasks. To provide the appropriate behavior, these applications must be able to acquire and manage information about the resources populating the smart environment. We propose an integrated resource management solution targeted to highly dynamic and heterogeneous smart environments. The proposed middleware is based on the Java Management Extensions (JMX) and on the Mobile Agent (MA) technology. JMX provides a unifying interface to different monitoring/management mechanisms, thus simplifying the integration with very heterogeneous distributed resources. MAs simplify the processing and the aggregation of raw management data to dynamically consider and achieve application-specific management goals, thus providing the flexibility and the level of abstraction needed in the addressed scenario.

***Index Terms* — Smart Environments, Resource Management, Java Management Extension, Mobile Agent**

I. INTRODUCTION

The spreading of wireless portable devices, wireless networking solutions, and embedded computing devices makes more and more feasible different kinds of "smart environments", i.e., computing environments where mobile objects, augmented with computing and communication capabilities, are available for seamless utilization in their proximity, without any static mutual knowledge and explicit configuration operations [1, 2]. In order to support the activities of the environment occupants these applications must be context-aware, i.e. able to modify their behavior on the basis of many different information about the resources available in the environment, and about the users populating the environment. For instance, a context-aware tourist guide may use the knowledge about user location and interests to select relevant information to display, and may use the knowledge about available network bandwidth and user device to decide the presentation format of that information.

On the one hand, the design and implementation of smart environments certainly requires dynamic support solutions for spontaneous communications and interactions of heterogeneous and statically unknown resources, with very different computing capabilities. Many research activities have recently focused on these communication/binding aspects [3, 4].

On the other hand, we claim that smart environments need very flexible support infrastructures capable of inspecting, processing and aggregating potentially huge amounts of monitoring/management data from heterogeneous resources dispersed in the environment, to provide applications with an updated concise view of the dynamic state of locally available resources of interest. In other words, the enormous number of objects dynamically available in smart environments forces rethinking traditional solutions for network/systems/service management to overcome the high heterogeneity of provided management data.

The paper presents the design and implementation of an integrated Java-based management architecture targeted to support the provisioning of pervasive services in smart environments and capable of addressing the requirements of this scenario, i.e., mainly to face the heterogeneity of resources, monitoring tools, and monitoring data in highly dynamic environments. The infrastructure acts as an intermediate layer between the distributed and dynamically available resources and the service logic: it gets information about managed resources, provides core management services, and composes core services to provide more advanced management services to be configured depending on application-specific requirements.

The proposed management architecture is integrated with the Java Management Extensions (JMX) [5] and exploits the Mobile Agent (MA) technology.

We adopt the JMX resource representation to model the resources composing the target environment. Managed resources can span from network/system hardware devices to service software components and are modeled as Java objects; in case of non Java-based resources, the modeling object is a Java wrapper. Modeling objects provide homogeneous management interfaces by exposing the relevant management parameters. We extend the JMX service infrastructure to implement core management services, by monitoring resource parameters, by establishing relationship between resources, and by enabling the communication with other environments through protocol adapters. Finally, we exploit the MA tech-

nology to realize advanced and application-specific management services. The MA approach provides useful features both in the implementation of management tasks and in the realization of "smart services". The MA-based management can take advantage of their characteristics of mobility, to operate locally and avoid micro-management problems due to remote and centralized interrogations, and of autonomy, to go on with management tasks even in presence of network partitioning. The MA-based provisioning of "smart services" can take advantage mainly of their characteristics of location-awareness and personalization, to adapt services to access locality and user profiles, and of dynamicity, to modify the environment behavior at provisioning time by dynamically installing/discarding service components.

The paper also shows the modularity and scalability of the proposed architecture: managed resources, management services and protocol adapters are components that can be added dynamically, by registering to responsible registry components and by exploiting discovery and class loading mechanisms, to provide new and personalized management functionality.

The paper is structured as follows. Section 2 presents a smart environment scenario pointing out the main management requirements. The features of the enabling technologies used in our solution, JMX and MAs, are discussed in Section 3. Section 4 provides an overview of JMX, while Section 5 and 6, respectively, give the architecture of the proposed resource management solution and show the management components at work in the previously sketched scenario. Related work, conclusive remarks and directions of current research end the paper.

II. Scenario and Management Requirements

Let us start by illustrating a possible practical scenario of typical management requirements of smart environments. Consider a campus area equipped with IEEE 802.11/Bluetooth Service Access Points (SAP) and very different types of wireless devices carried by students while roaming in classrooms and localities within the campus. Devices can span from Wi-Fi PDAs and laptops to Bluetooth cell phones and digital cameras. An example of service in the campus smart environment could be the following. A student called Alice is in the study-room B, waiting for the beginning of a class. Alice realizes that she needs to download the software package x, licensed to all the students in the campus, and to install it on her laptop to better follow her next class. Any teaching room in the campus provides wireless connectivity to allowed Internet sites and to a set of local campus servers, in particular to one FTP-based replicated repository with downloadable software. Alice starts to download package x in room B, but the available bandwidth is very limited because several students are surfing the Web there. It is a monitoring/management support distributed in the smart environment that estimates the time required to terminate the download depending on local network conditions and on the download server status.

Let us observe that it is generally impossible to assume that clients directly take care of monitoring the smart environment status and of understanding autonomously the best solution, mainly because wireless client devices have strict constraints on computing capabilities. The integrated management infrastructure should monitor the available bandwidth in nearby rooms (network-layer management data) and also the status of available download servers, together with the list of software packages available for download (application-layer management data). A dialog box can pop up on Alice's laptop display to suggest her to move to study-room C, where the management infrastructure estimates she can complete the software download in less time than staying in study-room B.

The above scenario shows specific management aspects. The first one is the need of getting and merging monitoring/management data at different levels of abstraction from heterogeneous resources. Monitored resources, in fact, spread from low-level network/system resources to high-level service ones. For instance in the previous scenario, , the Wi-Fi SAP typically exposes network-layer monitoring data, e.g., the currently available bandwidth in the wireless locality, while the FTP download server should provide application-level information such as the current load status and the list of downloadable software packages. Another point is the requirement that the management infrastructure for smart environments should collect monitoring/management data by exploiting different (often legacy) monitoring mechanisms and by handling heterogeneous monitoring representation formats. For instance, SNMP can be used to monitor network-layer information from a network device [6], while language-specific instrumentation or profiling strategies [7, 8] could get information about a Java-based application-layer resource. Monitoring data could present very different representation formats and very different granularity. Resources equipped with SNMP present monitoring information that depends on the type of resource: for example, the 802.11 SAP MIB contains information such as the station ID, the medium occupancy limit, the desired service set ID, and so on. The monitoring data of a Java resource inspected via the Java Virtual Machine Profiler Interface (JVMPI) [9, 10] may include the number of invocations of specified methods, the number of object allocations and deallocations, and so on. The heterogeneity of monitoring data in smart environments requires the management support to merge information at very different levels of abstraction by combining knowledge from very different layers in the system.

III. Enabling Technologies: JMX and Mobile Agents

As shown in the above scenario, smart applications need to know properties of both physical entities and available resources such as: the location of users, the location and properties of servers, the amount of network bandwidth available, the CPU load on various servers, etc. As discussed above, the support in charge of acquiring and managing these data has to face two main issues: resource heterogeneity and knowledge

extraction/provision. The first goal is to overcome the heterogeneity of the resources composing the managed environment, of the sensing mechanisms, and of the monitoring data representation formats. The second goal is to compose and to aggregate rough and low-level data translating it to an application-understandable format, and to provide this knowledge to interested applications when and where needed. We exploited JMX to achieve the first goal, and the MA technology to achieve the second one.

JMX provides a unifying interface to different monitoring/management mechanisms, thus integrating very heterogeneous resources, both physical and logical, Java-based and not, etc. This characteristic helps us to create a homogeneous and manageable view of the resources in our target environment, which spread from network equipments to application-level service components. Moreover, JMX provides a "management container" where resources and management services can be registered and deregistered at any time; this flexibility defines a dynamically extensible architecture of pluggable components, which suits very well to environments characterized by a very high number of objects dynamically available and by very variable management requirements.

MAs can represent a suitable effective technology to implement high level management functionality and "smart services", requiring both dynamicity, mobility, location awareness and personalization capabilities. The MA-based management can take advantage of their characteristics of mobility, to operate locally and avoid micro-management problems due to remote and centralized interrogations, and of autonomy, to go on with management tasks even in presence of network partitioning. The MA-based provisioning of "smart services" can take advantage mainly of their characteristics of location-awareness and personalization, to adapt services to access locality and user profiles, and of dynamicity, to modify the environment behavior at provisioning time by dynamically installing/discarding service components. To take advantage of the MA features our management solution operates on top of the SOMA platform, which is a Java-based general-purpose middleware for the design, development and deployment of MA-based applications [11].

The recent literature in the field widely recognizes the advantages of the MA adoption in the management area, especially in mobility-enabled and pervasive environments [12,13]. In the following, we focus on JMX, which is a specific and original choice of the proposed management architecture.

IV. The Java Management Extensions

Several different management areas present a dominant management technology: Common Management Information Protocol (CMIP) and Telecommunication Management Network (TMN) in the telecommunication area, Simple Network Management Protocol (SNMP) in the device and network management area, Web-Based Enterprise Management (WBEM) in the management of enterprise computing environments, etc. In the application management area, on the contrary, there is no dominant technology. In case of applications or resources running on only one operating system, or on one vendor's system, the choice can be guided by the preferences of that vendor. Microsoft's Windows, IBM's AIX, Sun's Solaris, and Hewlett-Packard's HP-UX have their own management system. However, one of the main advantages of Java-based applications is that they are portable on many different vendor platforms. This means that making manageable Java-based software may need to support multiple management technologies and systems, and may be consequently very hard and expensive.

These considerations have suggested SUN researchers to extend the Java platform with JMX to fulfill the manageability requirements of Java-based applications, by following two main guidelines: interoperability and dynamicity. JMX is interoperable in the sense that integrates with different legacy management systems, and is dynamic in the sense that managed resources and management functionality can be added or removed at any time.

JMX is an isolation layer between the managed resources and the management systems. This layer provides general purpose facilities to support the management of very different resources, which can be applications (Java-based or not), devices, or software implementations of services or policies, and to make these resources manageable by very different management systems. JMX provides transparency and interoperability through interface components from the Java environment towards other environments in two directions, as depicted in Figure 1. In one direction, from the JMX layer towards the underlying managed system, provides access to the manageability capabilities of non-Java resources via the Java Native Interface (JNI) and wrappers [14]. In the other direction, from the JMX layer towards the above management application, supports the interaction with non-Java management applications via protocol adapters.

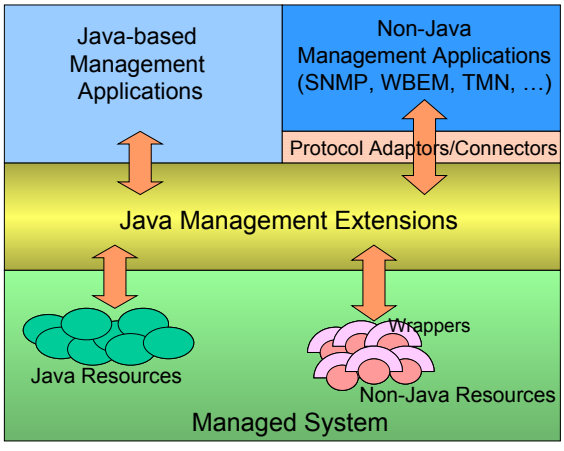

Fig. 1. The JMX isolation layer

A. JMX Architecture

Figure 2 shows the JMX architecture built according to a three-level model: Instrumentation, Agent and Manager Level. The Instrumentation Level gives manageability to managed

resources. The Agent Level provides core management services implemented as components contained in JMX agents. The Manager Level provides management entities that can operate as managers or as proxies between JMX and an external manager for distribution of management services. In addition, JMX provides a number of Java APIs, called Additional Management Protocol APIs, for existing management protocols. These APIs are independent of the three-level model, and are essential to enable Java-based JMX applications to link with existing management technologies.

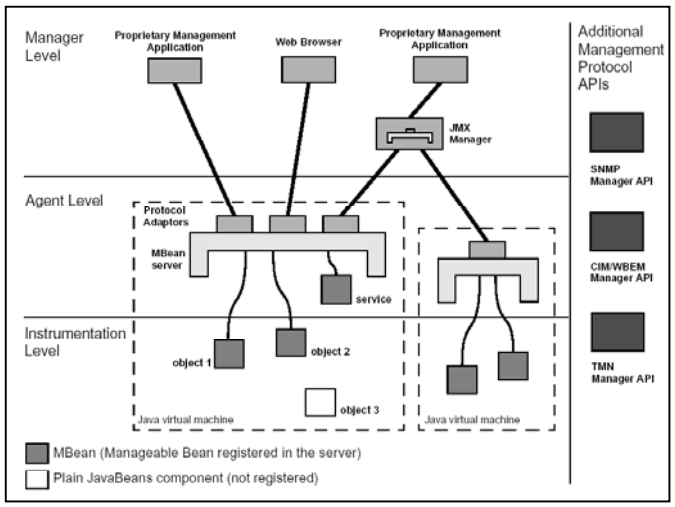

Fig. 2: The JMX Architecture

B. JMX Components

The main JMX components are: JMX Manageable Resources, JMX Agents, and JMX Managers.

A *JMX Manageable Resource* is a resource that has been instrumented in accordance with the JMX Instrumentation Level Specification and tested against the Instrumentation Level Compatibility Test Suite. JMX introduces a JavaBeans model called Management Bean (MBean), for representing both the management interfaces of the managed resources and the management services. A JMX manageable resource is instrumented with a MBean that expose attributes, operations, and notifications used to manage it. Each MBean has metadata in MBeanInfo, which defines the exposed attributes, operations, and notifications supported by the MBean.

JMX specifies four types of MBeans: standard, dynamic, open and model. A *Standard MBean* can be any JavaBean or JavaBean-style program that has been registered with the MBeanServer. Standard MBeans can be useful if the application already has management-oriented classes to support its own manager. *Dynamic MBeans* allow the application or domain-specific manager to define or generate the management interface for the resource at runtime. This provides a simple way to wrap existing nonbean-style or even non-Java resources. *Open MBeans* are dynamic MBeans that are restricted to accepting and returning a set of standard data types. By using open MBeans and these primitive data types it is possible to eliminate the need for class loading. A *Model MBean* is a customizable, standardized, dynamic MBean implementation. An implementation class of a model MBean named RequiredModel MBean must come with the JMX agent; this class can be instantiated and customized by the application with the proper management interface information reusing existing implementation. Managed Resources communicate data and events to management systems with their MBean through the JMX Agent.

A *JMX agent* is a management entity implemented in accordance with the JMX Agent Specification and tested against the Agent Level Compatibility Test Suite. A JMX Agent is composed of an MBeanServer, a set of MBeans representing managed resources, and at least one protocol adaptor or connector. A JMX Agent may also contain management services, also represented as MBeans. The service MBeans are at an higher level of abstraction than the resource MBeans, so they are represented at the Agent Level in the Architecture, as shown in Figure 2.

The *MBeanServer* is a registry for MBeans in the agent. The MBeanServer is the component which provides the services allowing the manipulation of MBeans. All management operations performed on the MBeans are done through Java technology-based interfaces on the MBeanServer. The MBeanServer runs on the JVM local to the managed resources' MBeans, provides a query service for the MBeans and acts as a delegator to the MBeans, returning the results to the requester. The MBeanServer also provides access to the metadata about the MBeans in the MBeanInfo instance, and notification registration and forwarding support to MBeans representing adapters, services and resources.

The JMX agent includes a set of required services: the monitoring service, the timer service, the relation service, and the MBean class loader. Services are MBeans registered with the MBeanServer that provide some generic functionality that can be used by MBeanServers, MBeans, and adapters. Additional management services can be added dynamically as service MBeans by applications or management systems, making the JMX agent flexible and extensible. The *monitoring service* runs monitoring MBeans on a scheduled basis. It must support basic monitoring MBeans, including Gauge, Counter, StringMatch, and StateChange. Additional or specialized monitoring MBeans can also be developed and used. The *timer service* executes an operation on a timed basis; it is used by the monitoring service. The *relation service* supports relationships MBeans, which contain the names of a set of MBeans that are related in some way. Some kinds of possible relationships include "contains" and "depends on". The *Mlet (Management Applet) service* is an MBean class-loading service that loads an MBean across a network when an MLET tag in a HTML page is encountered.

Protocol adaptors and *connectors* let management applications access a JMX agent and manipulate the MBeans it contains. Protocol adaptors give a representation of the MBeans directly in another protocol, such as HTML or SNMP. Connectors include a remote component that provides end-to-end communications with the agent over a variety of protocols (for

example HTTP, HTTPS, IIOP). Adapters are also MBeans, and they are registered with the MBeanServer. Common, though nonstandard, adapters are RMI, HTTP, and SNMP. CIM and IIOP adapters are in the process of standardization. Protocol adaptors and connector are fundamental to allow interoperability and integration between JMX applications and other management/monitoring systems. This is a very important characteristic when dealing with extremely heterogeneous resources such as composing smart environments.

A *JMX manager* is a management entity implemented in accordance with the JMX Manager Specification and tested against the Manager Level Compatibility Test Suite. A JMX manager provides an interface for management applications to interact with the agent, distribute management information, and provide security. JMX managers can control any number of agents, thereby simplifying highly distributed and complex management structures.

V. Resource Management Architecture and Implementation Guidelines

We propose a resource management architecture structured in three levels: a Resource Description Level (RDL), a Resource Management Level (RML), and a Context Management Level (CML), as shown in Figure 3.

The first level implements a set of managed resources, in accordance with the specifications of the JMX Instrumentation Level. The second level defines a set of basic management services to facilitate the implementation of advanced management services by extending the required functions of the JMX Agent Level. The third level exploits the services realized by RML and the MA features to provide the applications with a view of the environment at the proper level of abstraction.

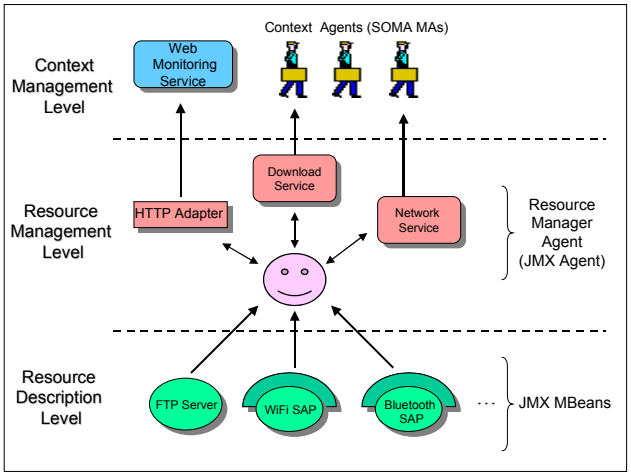

Fig. 3: Resource Management Architecture

Before describing RDL, RML and CML, let us introduce the adopted locality abstractions, to better understand where the management functions are deployed and performed. In our model the target *environment* is the whole equipped area, for instance the campus area, while the *localities* composing the environment are the wireless LANs covered by WiFi SAPs.

The RDL and the RML operate locally, i.e., within a single locality. This implies that each managed locality presents a RDL, which describes its resources, and a RML, which provides management core services operating on the local resources. The CML provides distributed services involving resources in different localities, by moving the management entities it is composed of within the target environment.

A. The Resource Description Level

As described above, JMX represents the management interfaces of the managed resources by exploiting the mediation of a MBean. The lower level of our architecture, the RDL, adopts the JMX resource model to represent the resources composing the localities of the managed environment. The set of implemented MBeans covers very different kind of resources, that can span from network/system hardware devices, such as hosts, printers, WiFi and Bluetooth SAPs, to service software components, such as HTTP and FTP servers. In case of non Java-based resources, the modeling MBean is a Java wrapper.

The wrappers could reside in a node different from the one where the modeled resource is, and do not require limited devices to host the JVM. This is a significant advantage because we can integrate devices with very limited computing capabilities in our management middleware by simply communicating with their management entity wrapped by a modeling object. For instance, an 802.11 SAP cannot run a JVM instance but is able to communicate with a remote SNMP agent. As shown in Figure 4, our architecture wrap this SNMP agent with a modeling Java object, the SNMP peer, representing the SAP resource.

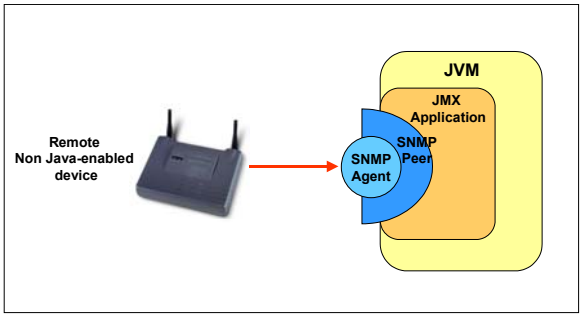

Fig. 4: Remote SNMP wrapping mechanism

B. The Resource Management Level

Each locality is equipped with a RML that realizes the core management services mainly by monitoring resource parameters. The RML is composed of a *Resource Manager Agent* implemented by following the JMX Agent Level specifications. The agent includes one MBeanServer and one protocol adapter. The MBeanServer registers the local resources, described by the resource MBeans in the underlying layer, and the provided management services, implemented via service MBeans defined at this level. The main protocol adapter used is the HTTP adapter, implemented in the Java Dynamic Man-

agement Kit (JDMK) [15]. It is used to communicate with Web browsers and permits the implementation of the Web Monitoring Service, defined in the CML.

RML provides two main basic management services: the first one providing information about the local network availability (Network Service), and the second one providing information about the software available for download from the local servers (Download Service). To implement these two services we have defined two monitoring MBeans: the Network MBean, and the Download MBean. They collect information from the RDL on-demand, on the basis of the CML requests. The *Network MBean* collect information about the network bandwidth from the resource MBeans representing network devices, such as the WiFi SAP and the Bluetooth SAP. The *Download MBean* collects information about the software available for download from the resource MBeans representing the FTP servers.

C. The Context Management Level

The CML is composed by manager components that coordinate services provided at the RML to realize more high-level advanced-management services. The CML provides the Web Monitoring Service, and some application-specific services.

The *Web Monitoring Service* allows to access the state of resources in the target environment via a standard Web browser: the user can select the locality, and the Web Monitoring Service, exploiting the HTTP adapter, asks the Resource Manager Agent in the requested locality about the resources it manages. Then the Web Monitoring Service provide the user with a description of that locality by showing the received monitoring data. This allows the user to know, for instance, if a certain room is suitable to host a meeting with specific equipment requirements.

The application-specific services are implemented in terms of MAs, the Context Agents (CA). Different type of CAs implements different advanced-services. Once instantiated, the CA works as care-of entity of the user requesting the service and follows the user movements during the service session by carrying the user profile with itself. In our scenario, for instance, the advanced service is the suggestion of the best FTP downloading source, and is implemented with a specific CA in charge of performing this task: the *FTP Download Session Agent*.

VI. Deployment Scenario

To clarify how the management components interoperate and coordinate in an actual service scenario, we present them at work in the campus scenario described in Section 2.

When Alice requests the software download from the study-room B, the management support instantiates one CA, the FTP Download Session Agent, to assist her in the download operation. This CA asks the local RML for the download rate and verifies that in this locality it is under a pre-defined threshold. In order to suggest the best download source, the CA asks the RMLs of all the campus localities about the available bandwidth, by exploiting the service provided by the local Network MBeans, and about the software available in the FTP servers, by exploiting the service provided by the local Download MBeans. By processing the gathered information the CA ascertains that another campus locality, corresponding to the study-room C, provides, at that moment, the highest bandwidth, and that in the same locality there is also one active FTP server providing the requested software. Therefore, the CA pops up a dialog box on Alice's laptop display to suggest her to move to study-room C. When Alice moves to the new locality, the CA migrates to go on assisting her. From the Alice user profile and from the class/software profile the CA infers that Alice could need also another software to follow the class is waiting for. The CA verify that this software is available in the locality where Alice is and that with the current download rate she is able of downloading it before the beginning of the class; therefore, it pops up a new dialog box on Alice's laptop display to suggest her to download this software. Once Alice terminates the downloads, she disconnects her laptop and the CA terminates its execution.

VII. Related Work

Several research activities in the pervasive computing area have investigated applications for smart environments and their supporting systems, by focusing in particular on the management of context-awareness.

Some projects specifically address the flexibility and scalability of context information aggregation and dissemination. The Context Toolkit is a distributed architecture supporting context information fusion and delivery [16]. It uses components, called *widgets*, which are responsible for acquiring a piece of information from the environment and for making it available in a generic manner, regardless of how is actually sensed. Pre-defined *aggregators* process data from widgets and provide commonly used aggregated information, while *interpreters* are responsible for deciding the meaning of this aggregated information, i.e. what type of situation can be inferred from a certain aggregated context information. The separation of the interpretation from applications allows reuse of interpreters by multiple applications, thus reducing application development cost. The same objective is the main focus of Solar [17]. The Solar platform allows dynamic injection of context processing modules that can be shared across applications. The main idea is that many adaptive applications ask for similar context information; it is then natural to re-use the overlapping context aggregation functions among applications. The Solar approach is to decompose the context-aggregation process into a series of modular and re-usable operators, each of which is an object that subscribes to and processes one or more input event streams, by publishing one event stream. Since the inputs and outputs of an operator are all event streams, the applications can use a tree of recursively connected operators (operator graph) to collect and aggregate the desired context.

Other projects focus on the interaction between context-

aware applications and context information source. The consideration that drives the Contextual Services in AURA [18], for instance, is that context-aware applications may have very different context information requests, and the system providing this information must be able to support all kind of requests. In order to fulfill this requirement, Contextual Services provide applications with an SQL-like query interface that allows them to easily synthesize information from different resources.

The above researches mainly investigate the management aspects related to context view generation and on providing context information to context-aware applications. However, they do not take into account the issues related to the heterogeneity of the managed resources, and their integration with other monitoring/management systems.

By considering the implementation technology, the portability features and the wide diffusion of the Java technology encourage the development of Java-based management solutions [19], and JMX has been created specifically to this purpose.

Even if not targeted to smart applications, there are some research activities investigating the usage of JMX in the implementation of management/monitoring tools. In [20], JMX is adopted in the implementation of a Java API for advanced fault management, the JFMX API, which aims at facilitating the development of diagnostic tests and performance measurements. The JFMX API is organized in three packages mapped to the JMX layered architecture. The instrumentation level package is composed of MBeans modeling resources under test and/or monitoring. The agent level package implements a set of dynamic functionality to control test and monitoring activities, while the manager level package provides Manager MBeans to coordinate and manage a set lower level agents triggered by a user session request. The goal of the MobiMon project [21], instead, is to allow the monitoring of remote systems and applications using Java-enabled mobile devices. It exploits a MIDlet-based user interface presenting the user with a hierarchical view of monitored systems, attributes and actions which can be queried and/or invoked remotely. MobiMon MIDlets contact the MobiMon servlet via HTTP. The MobiMon servlet is the communication endpoint for MobiMon MIDlets, receives attribute queries and action requests from the MIDlets, forwards them to appropriate MobiMon agents, receives the results and sends them to the MIDlets. The MobiMon agents run on managed nodes and act as MBean servers according to the JMX agent specification. A MobiMon agent running on a specific node hosts all MBeans needed to manage resources of this node, which are deployed according to the JMX instrumentation level.

The above proposals confirm the interest in evaluating the feasibility and the potentialities of JMX-based management. However, to the best of our knowledge, our integrated resource management architecture is the first work that adopt JMX to support the gathering and management of context information in highly dynamic and heterogeneous smart environments.

VIII. Conclusion

We have designed and implemented an integrated resource management architecture for supporting the development of context-aware services in smart environments. The main advantages of our infrastructure are its capability of facing the heterogeneity of the environment, its modularity and its dynamic extensibility. These features are a consequence of the adopted enabling technologies: JMX and MAs. JMX provides a unifying interface to very different resources, and a modular architecture of pluggable components, by exploiting the MBean model. The MA technologies facilitates the implementation of "smart services" mainly because of their characteristics of location-awareness, personalization, and dynamicity, to modify the environment behavior at provisioning time.

The first encouraging results achieved in using the proposed middleware to support the realization of some simple services are stimulating further research work to produce a more usable and complete management solution. We intend to extend both the set of resource MBeans, to manage new physical and logical resources, and the set of service MBeans, to enrich the low-level management functionality. We are also evaluating the integration of a messaging service developed by exploiting the Java Messaging Service [22], to facilitate the communication between the smart environment and its users.

Acknowledgment

This work is supported by the Italian MIUR (FIRB WEB-MINDS project) and CNR (IS-MANET project).

References

[1] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges", IEEE Personal Communication, Aug. 2001.

[2] B. Johanson, A. Fox, T. Winograd, "The Interactive Workspaces project: experiences with ubiquitous computing rooms", IEEE Pervasive Computing, Apr.-Jun. 2002.

[3] S.S. Yau, F. Karim, Yu Wang, Bin Wang, S.K.S. Gupta, " Reconfigurable context-sensitive middleware for pervasive computing", IEEE Pervasive Computing, Jul.-Sept. 2002.

[4] A. Misra, S. Das, A. McAuley, "Autoconfiguration, Registration and Mobility Management for Pervasive Computing", IEEE Personal Communications, Aug. 2001.

[5] Sun Microsystems - Java Management Extensions (JMX), http://java.sun.com/products/JavaManagement/doc.html.

[6] J. D. Case, et al., "Simple Network Management Protocol (RFC 1157)", DDN Network Information Center, SRI International, May 1990.

[7] G. Czajkowski, T. von Eicken, "JRes: A Resource Accounting Interface for Java", ACM OOPSLA Conference, Vancouver, 1998.

[8] P. Bellavista, A. Corradi, C. Stefanelli, "Java for On-line Distributed Monitoring of Heterogeneous Systems and Services", The Computer Journal, Vol. 45, No. 6, pages 595-607, Oxford University Press, Nov. 2002.

[9] Sun Microsystems - Java Virtual Machine Profiler Interface (JVMPI), http://java.sun.com/ products/jdk/1.3/docs/guide/jvmpi/jvmpi.html.

[10] Sun Microsystems - Java Native Interface (JNI), http://java.sun.com/products/jdk/1.3/docs/guide/jni.html.

[11] P. Bellavista, A. Corradi, C. Stefanelli, "An Integrated Management Environment for Network Resources and Services", IEEE Journal on Selected Areas in Communications, Vol. 18, No. 5, May 2000.

[12] D. Gavals, D. Greenwod, M. Ghanbari, M. O'Mahony, "Hyerarchical network management: a scalable and dynamic mobile agent based approach", Computer Networks, Elsevier Science, Apr. 2002.

[13] P. Bellavista, A. Corradi, R. Montanari, C. Stefanelli, "Dynamic binding in mobile applications - A middleware approach", IEEE Internet Computing, Mar.-Apr. 2003.

[14] Sun Microsystems - Java Native Interface (JNI), http://java.sun.com/products/jdk/1.3/docs/guide/jni.html.

[15] Sun Microsystems – Java Dynamic Management Kit (JDMK), http://java.sun.com/products/jdmk/

[16] A.K. Dey, G.D. Abowd, D.A. Salber, "A Context-based Infrastructure for Smart Environments", 1st Int. Workshop on Managing Interactions in Smart Environments (MANSE'99), Dec. 1999.

[17] G. Chen, D. Kotz, „Context Aggregation and Dissemination in Ubiquitous Computing Systems", 4th IEEE Int. Workshop on Mobile Computing Systems and Applications (WMCSA'02), 2002.

[18] G. Judd, P. Steenkiste, "Providing Contextual Information to Ubiquitous Computing Applications", Carnegie Mellon University Technical Report, Jul. 2000.

[19] J. Lee, "Enabling Network Management Using Java Technologies", IEEE Communications Magazine, Jan. 2000.

[20] M.H. Guiagoussou, R. Boutaba, M. Kadoch, "A Java API for Advanced Faults Management", IFIP/IEEE International Symposium on Integrated Network Management (IM'01), May 2001.

[21] The MobiMon Project – Mobile System Monitoring, http://mobimon.sourceforge.net/

[22] Sun Microsystems – Java Messaging Service (JMS), http://java.sun.com/products/jms/