Middleware for Reactive Components: An Integrated Use of Context, Roles, and Event Based Coordination

Andry Rakotonirainy[1], Jaga Indulska[2], Seng Wai Loke[3], and Arkady Zaslavsky[4]

[1] CRC for Distributed Systems Technology
Level 7, General Purpose, University of Queensland 4072, Australia
andry@dstc.edu.au
[2] School of Computer Science and Electrical Engineering
The University of Queensland 4072, Australia
jaga@csee.uq.edu.au
[3] School of Computer Science and Information Technology
RMIT University, GPO Box 2476V, Melbourne VIC 3001, Australia
swloke@cs.rmit.edu.au
[4] School of Computer Science and Software Engineering
Monash University, Caulfield VIC 3145, Australia
A.Zaslavsky@csse.monash.edu.au

Abstract. The proliferation of mobile devices and new software creates a need for computing environments that are able to react to environmental (context) changes. To date insufficient attention has been paid to the issues of defining an integrated component-based environment which is able to describe complex computational context and handle different types of adaptation for a variety of new and existing pervasive enterprise applications. In this paper a run-time environment for pervasive enterprise systems is proposed. The associated architecture uses a component based modelling paradigm, and is held together by an event-based mechanism which provides significant flexibility in dynamic system configuration and adaptation. The approach used to describe and manage context information captures descriptions of complex user, device and application context including enterprise roles and role policies. In addition, the coordination language used to coordinate components of the architecture that manage context, adaptation and policy provides the flexibility needed in pervasive computing applications supporting dynamic reconfiguration and a variety of communication paradigms. [1]

1 Introduction

Pervasive (ubiquitous) environments are characterised by an immense scale of heterogeneous and ubiquitous devices which utilise heterogeneous networks and

[1] The work reported in this paper has been funded in part by the Co-operative Research Centre Program through the Department of Industry, Science and Tourism of the Commonwealth Government of Australia

R. Guerraoui (Ed.): Middleware 2001, LNCS 2218, pp. 77–98, 2001.

computing environments that communities use across a variety of tasks and locations. The pervasive environment is an environment in which components, in spite of this heterogeneity, can react to environmental changes caused by mobility of users and/or devices by adapting their behaviour, and can be reconfigured (added or removed dynamically). Pervasive environments need to offer a variety of services to a diverse spectrum of entities. Also, services can appear or disappear, run-time environments can change, failures may increase, and user roles and objectives may evolve.

Adaptability and reactivity is the key to pervasive computing. Pervasive environments require the definition of a generic architecture or toolkit for building and executing reactive applications. Reactive behaviours are triggered by events; in many cases the trigger is not a single event, but a possibly complex composition of events. These behaviours exist in many domains and are very useful for e-business applications (stock market, sales alerts, etc.) There are already existing research and industry approaches which address many adaptation problems.

However, no integrated architecture that is open, flexible and evolvable yet exists. This situation makes it impossible to test, within a single architectural framework, concepts such as (i) rich context description, (ii) generic approach to decisions about adaptability methods, (iii) dynamic definition of communication paradigms, (iv) dynamic reconfiguration of the architectural components, (v) incorporation of enterprise aspects like roles of participants and policies governing roles.

Most existing pervasive architectures have been influenced by a specific type of application, or by the type of underlying technology, thereby resulting in a lack of generality (see Section 2). In order to avoid building a monolithic architecture we aim to provide generality, and to design an architecture that separates clearly the computation aspect from the coordination aspect. The computation aspect is in turn separated into dedicated services modelled as context, adaptation and policy managers. These aspects are often merged into a monolithic framework preventing flexibility and extensibility. Architecture Description Languages (ADL) have been proposed as modelling notations to support architecture-based development [19]. Their ability to scale to large systems and pervasive environments are yet to be proven [23]. To coordinate events of autonomous components without re-writing these components, we defined a rule and event based coordination script, which has a high-level declarative meaning in terms of overall structure of the architecture to specify cross-component interactions.

In this paper we describe the m3 architecture and its Run-Time Environment (m3-RTE). The basic concepts of the framework were presented in [14]. This paper presents extensions and detailed applications using the proposed architecture. The structure of this paper is as follows. Section 2 provides both brief characteristics of related work and their evaluation. Section 3 gives a high level description of m3-RTE. Section 4 describes the design requirements and modelling concepts used in the proposed architecture. Section 5 characterises the architecture and describes the functionality and interfaces of its main components. It also describes interactions between components. Section 6 discusses

several issues related to the implementation of the architecture (m3-RTE Run Time Environment) including coordination of events, descriptions of the current status of the architecture prototype and its main components. Section 7 shows examples of applications using the services of m3-RTE. Section 8 discusses the benefits of m3-RTE. Section 9 concludes the paper and mention future works.

2 Related Work

Sun[tm] Open Net Environment [26] (Sun ONE) provides the means for deriving context: information about a person's identity, role, and location, about security, privacy, and permissions - all the things that make services smart. In Sumatra [1], adaptation refers to resource awareness, ability to react and privilege to use resources. Sumatra provides resource-aware mobile code that places computation and data in response to changes in the environment. Rover [17] combines architecture and language to ensure reliable operations and is similar to Sumatra in the sense that it allows loading of an object in the client to reduce traffic. It also offers non-blocking RPC for disconnected clients. Bayou [8] takes a different adaptability approach by exposing potentially stale data to the application when the network bandwidth is poor or down. Bayou is similar to Coda [24] in that respect. Coda is a distributed file system that supports disconnected operation for mobile computing. Mobiware [3] and Odyssey [20] are more focused on adaptability measurement. Mobiware defines an adaptation technique based on a utility-fair curve. This micro-economic based method allows the system to dynamically adapt on the QoS (Quality of Service) variation of the data link layer network. Odyssey's approach to adaptation is best characterised as application-aware adaptation. The agility to adapt is defined as a key attribute to adaptive systems. It is based on control system theory where output measurements are observed from the input reference waveform variation. TSpaces [15] provides group communication services, database services, URL-based file transfer services, and event notification services. It is a networked middleware for ubiquitous computing. TSpaces can be succinctly described as a network communication buffer with database capabilities. JavaSpaces Technology [11] is similar to TSpaces and is a simple unified mechanism for dynamic communication, coordination, and sharing of objects between Java technology-based network resources like clients and servers. In a distributed application, JavaSpaces technology acts as a virtual space between providers and requesters of network resources or objects. PIMA, Aura, Limbo and Ensemble address issues about architecture as m3 does. Limbo [7] and Ensemble [22] are concerned with the mobile architecture and the programming model. Limbo offers an asynchronous programming model (tuple space) and defines an architecture for reporting and propagating QoS information that might be related to the system. Adaptability is achieved by filtering agents. Ensemble's architecture consists of a set of protocol stacks which are micro-protocol modules. Modules are stacked and re-stacked in a variety of ways to meet application demands. Adaptation is done transparently to the application. The lowest layer tries to adapt first. If it

can't, then it passes the notification to the layer above. This repeats until, eventually, the application itself has to reconfigure. PIMA [2] is a framework that focuses on providing a device-independent model for pervasive applications that facilitates application design and deployment. It provides functionalities such as an application adaptation enabler, and dynamic application apportioning to services. Aura [28] is a task-driven framework that helps the user to gather information about available services, selecting suitable services to carry out tasks and binding them together. A Service Coordination Protocol controls adaptability mechanisms required when the environment changes. Open-ORB [4] and dynamicTao [18] use reflection mechanisms to address dynamic configuration and adaptation of components. The two approaches make use of a reflective CORBA ORB that gives program access to meta information that allows the program to inspect, evaluate and alter the current configuration of components. In [10] is presented a set of requirements for future mobile middleware which support coordinated action between multiple adaptations triggered by multiple contexts.

As can be seen, the above work addresses specific types of adaptability for particular middleware and hence, caters for only specific kinds of contextual information. The cited works propose adaptation to the quality of service provided by the network or the Operating System and do not provide an open, evolvable architecture enabling the use of a broad range of concepts related to adaptation of enterprise applications. Furthermore, dynamic coordination of heterogeneous components, dynamic definition of communication paradigms and context awareness are not addressed in the existing solutions.

3 High Level Description of the m3 Architecture

In this section we first present a high level view of the m3-RTE (Run Time Environment). The m3-RTE can be considered as an ORB (OMG Object Request Broker) or an agent that adapts the delivery of information (protocol and content) with the use of context (user profile, server profile, etc), adaptation and policy services (see Figure 1). The m3-RTE has a programming interface that allows the system programmer to specify relevant contexts of interest, adaptation rules, policy rules and coordination within m3-RTE. Clients can access servers transparently. Servers are wrapped by m3-RTE so that all in/out-coming messages are filtered and coordinated by the m3-RTE coordination service.

Neither the user application nor the server do not need to know about the internal mechanism of m3-RTE: the adaptation of the request (protocol, content) to be delivered to a particular application can be done transparently. A Tickertape application is presented in Section 7 that demonstrates such an approach. Tickertape messages are pushed to applications based on subscriptions. m3-RTE was programmed to change dynamically the interaction protocol and the content of information delivery based on context (e.g. device capability, MIME content) and policy (e.g. level of security) without touching the client front end.

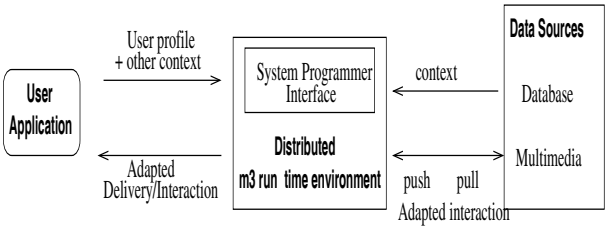

Fig. 1. m3-RTE located between the front end and back end

4 m3 Design Requirements and Modelling Concepts

In m3-RTE, we seek a balance in providing a high level model of a bounded pervasive environment (e.g., an enterprise) in terms of context, roles and events, and supporting dynamic lower level behaviours - such as reactivity and changes in configuration of components. More specifically, the design of the m3-RTE was driven by the following requirements:

1. Enterprise focus: As enterprise applications are complex and their complexity increases if they operate in pervasive environments. We need abstractions that capture the purpose, scope and policies of the system in terms of behaviour expected of different roles within the system by other entities (roles) within the enterprise. e.g the CEO is authorized to increase the salary of Project Leaders.
2. Reactivity/proactivity: reactivity is an important design guideline of the m3 architecture: the architecture should allow easy programming of reactions to context changes. The changing context of a component determines its interactions with other components and the service it can offer. Proactivity is the ability of an m3-RTE to foresee changes that might occur and prepare for those. This ability is crucial for successful adaptation and is achieved by generating sets of events to compensate for fluctuations in the system's behaviour and the surrounding environment. Patterns of behaviours and a history of previous executions are used for proactivity support.
3. Open dynamic composition: the system must support inter-operation with open components such as legacy applications or new components. New components may exhibit new types of interactions which go beyond the currently common but very static and asymmetric Remote Procedure Call (RPC). Dynamic composition (plug and play) configuration and customisation of services is essential in a pervasive environment as lack of resources to carry out a task may require a partial/complete reconfiguration of services.

Requirement (2) is addressed by using the event notification concept (publish/subscribe) as a core building block of the m3 architecture. Requirement (1) and (3) are addressed by integrating a policy manager inspired by RM-ODP enterprise policy specification and RM-ODP modelling concepts into the m3 architecture [16] . The three basic modelling concepts inspired by RM-ODP [16]

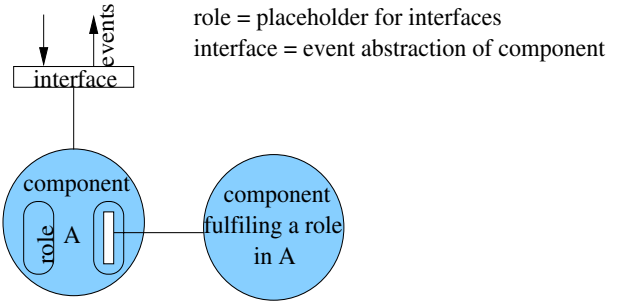

Fig. 2. m3 architecture modelling concepts

are components, their roles and the events to which components can react. These basic modelling concepts are illustrated in Figure 2.

- **Event** is an atomic modelling concept from which interaction protocols such as message passing, RPC, streams and other communication paradigms can be expressed. We assume that interactions between all components are event based.
- **Component** is a core modelling concept. It is a software abstraction. In object-oriented terminology, a component is a set of interacting objects (or even a single object).
- **Roles**: m3 focuses on RM-ODP enterprise specifications which define the purpose, scope and policies for a system in terms of roles played, activities undertaken, and policy statements about the system [16]. An enterprise role is an identifier for a behaviour (e.g. Director). A role is an abstraction and decomposition mechanism. It focuses on the position and responsibilities of a component within an overall system to carry out a task. An example of the Director's task is *supervise_subordinates*.
  A role cannot exist without a component. A role is a placeholder that can be fulfilled by one or more interfaces. Roles provide abstraction required for enterprise modelling.Roles can be added/removed dynamically during the lifetime of a component.

5 m3 Architecture

The modelling concepts in Section 4 are used to define the m3 architecture. The architecture is organised into three layers as shown in Figure 3:

- Coordination layer: a coordination manager that executes a coordination script written in MEADL (Mobile Enterprise Architecture Description Language)
- Dedicated manager layer consisting of three fundamental parts of pervasive and reactive computing environments. They are the Context, Adaptation and Policy managers.

– Distributed Service layer: services such as notification and security. This layer also includes network protocols.

The following subsections describe the functionalities of these three layers. It also gives examples of interaction within the m3 Run Time Environment (m3-RTE).

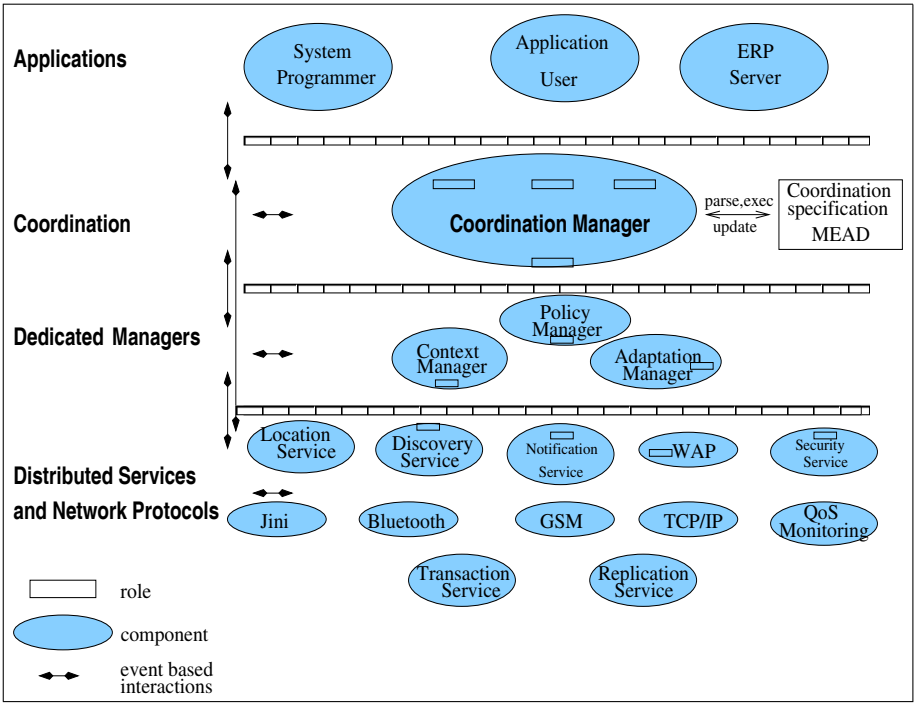

Fig. 3. Internal components of the m3 Architecture

5.1 Coordination Layer

The need for increased programmer productivity and rapid development for dynamic changes within complex systems are the main motivations that led to the development of coordination languages and models. Coordination-based concepts such as Architecture Description Languages (ADL) provide a clean separation between individual software components and their interaction within the overall software organisation. This separation makes large applications more tractable, more configurable, supports global analysis, and enhances reuse of software components.

The coordination layer is the core of the m3 architecture. The coordination focuses on concepts necessary to manage the implications of mobility and reactivity for enterprise-wide activities. For example the coordination layer should

be able to re-configure itself if a policy manager and/or context manager is unreachable or cannot be instantiated in a small device. The coordination consists of two parts (*i*) specifying with MEADL the interaction between enterprise roles and (*ii*) executing MEADL scripts within a coordination manager.

Coordination specification with MEADL. MEADL specifies the coordination of events. It is a coordination script that separates the behaviour specification (functionalities) of components from the communication needed to coordinate such behaviours. MEADL specifications provides means to configure dynamically the interaction between components, to change dynamically the communication paradigm and to provide session management.

We take a pragmatic (not formal description techniques) and minimalist (simple) approach to specify coordination. The script offers two main advantages: (*i*) interaction between components can be changed dynamically, and (*ii*) coordinated components can be added or removed dynamically.

Events belong to roles, and therefore, MEADL specifies the coordination of roles. Roles can be fulfilled by users, ERP(Enterprise Resource Planning) servers, network protocols or dedicated managers. A role's description is the interface of tasks that a component can provide. A role's duty is described in a set of input/output events. The duty of roles is to perform a set of tasks. The interface to tasks corresponds to the interface of components.

The m3 architecture is a reactive architecture and this is reflected in the use of the reaction rule syntax `on event` as first class construct of MEADL.

`on event` $R_j : E_i$ waits for the occurrence of a set of events E_i from a defined set of roles R_j. The value of E_i parameters is checked with a `WHERE` clause. The relevant context is also checked with `in context` or `out context` clauses.

Code 5.1 MEADL EBNF syntax

```
rule    : "on" pexpr [guard] "{" actions "}"
pexpr   : "event" role "(" [ident(","ident)*] | "(" pexpr ")"
          | pexpr op pexpr
guard   : "not"* ( "in" | "out" ) "context" string
actions : [action(";"action)*]
role    : ident ":" ident
op      : "where" | "and" | "or" | "eq" | "neq"
action  : ( "call" | "emit" ) ident "(" [expr ("," expr)*] ")"
```

The MEADL syntax is summarised in Code 5.1. An example of a MEADL sentence is shown in Code 5.2. Code 5.2 specifies a rule that waits for the occurrence of event `http:get(balance, account)` from the role `Director`. If the current context is secure and the balance is more than 10,000 then it emits a new event featuring the new balance then calls a function `foo`.

Code 5.2 Example of MEADL specification

```
ON EVENT Director:http:get(balance, account)
        WHERE (balance > 10,000)
        IN CONTEXT ‘‘secure_environment’’
    {
      EMIT employee_chat_line(‘‘DSTC has’’,balance);
      CALL foo(account, 23);
    }
```

Coordination Manager. The coordination manager is an engine that executes a MEADL script. MEADL scripts are transformed into XML [27]. Then, the coordination manager coordinates the events based on the generated XML specification. The use of XML is a provision for the future use of different coordination mechanisms (e.g. workflow engine) and language independent coordination.

The coordination manager coordinates the other components by sending/receiving events. To allow rules to affect other rules (a form of reflection) and to allow dynamic change of reaction rules, the Coordination Manager offers the following methods:

- `Reaction-Id create-react-rule (spec)` create a reaction rule
- `void delete-react-rule(reaction-Id)` delete a reaction rule
- `void map (event, operations)` : map an event to a set of operations
- `void inhibit (reaction-Id)`: inhibit a reaction rule
- `void reactivate(reaction-Id)`: reactivate an inhibited reaction rule

5.2 Dedicated Managers Layer

The middle layer of the architecture is composed of three dedicated managers which provide the three major functionalities. We identified them as necessary for modelling the enterprise pervasive environment (and thus, for building applications for such environments). The three managers are context, adaptation and policy managers. The Context Manager (CM) provides awareness of the environment and feeds the Adaptation Manager (AM) with context information in order to enable selection of an appropriate adaptation method. The Policy Manager (PM) ensures that the overall goals of the roles involved are maintained despite adaptation and that a change of roles is followed by appropriate adaptation.

The use of these managers is invisible outside the m3 architecture. They are autonomous components with well defined interfaces that interact together to provide the appropriate level of service to applications using the m3-RTE. Only the coordination manager has an impact on the way the three dedicated managers interact.

Context Manager (CM). Components may move to other environments or some features of the current context may change (memory, disk space, user preference etc). Context consists of

> "any information that can be used to characterise the situation of an entity, where an entity can be a person, place, physical, or computational object." [9]

The aim of the CM is to provide context awareness. The design of a context aware system must contain at least three separate but complementary steps. They are:

1. Context description: The syntactical representation of semantic-free information relevant to an entity.
2. Context sensing: Related to gathering contextual information specified in the context description.
3. Context interpretation: Related to understanding perceived information.

The current version of CM focuses on bullet point 3. The CM is responsible for gathering context information and making it available to the other entities. The CM enables consistent addition, deletion and modification of the context (value, attribute and relationship between attributes) and the detection of context changes.

We identified two types of context called local and community context. A component can be aware of its own *local context* (e.g. the characteristics of the device on which it is running, location, role fulfilled by the user), or the contexts of all the components or roles (on the same device or on other devices) it is interacting with. We call the combination of the contexts of all the interacting applications *community context*. A community being the composition of components formed to meet an objective [16]. Awareness of local or community context (or changes in local or community context) can itself be local (i.e., only one application has the awareness) which we call *local awareness*, or community (i.e., all the applications are aware of the context change at roughly the same time either synchronously or asynchronously) which we call *community awareness*.

The CM represents context as value/attribute pairs using RDF (Resource Description Framework) graphs. RDF instances assign values to instances. It is these instances that describe context. RDF enables the definition of schemata which describe the gathered context as well as allowing the use of relationship operators between attributes (e.g cardinality) or constraint properties. A community context is modelled as relationship between different contexts (e.g relationship between context of different roles). The use of RDF enables us to use the work of the W3C CC/PP [5] WG which is defining a vocabulary for describing device capabilities, as well as protocols to transmit these device profiles from the client. The interfaces provided by our context manager are:

- `ContextValue getContext (Id,Path,Callback)`: Get the context associated with Id
- `void addContextListener(Path, Callback)`: Register to be notified when context changes take place. Path restricts the scope of interest.
- `ContextAttr listContextAttr (Id,Path)`: Get the context vocabulary associated with context Id.

- `boolean ExistsContextAttr(Id,Path)`: Checks if the attribute pointed to by Path exist in context Id
- `boolean incontext(component, ctx)` Checks if a component is within a context
- `boolean outcontext(component, ctx)` Checks if a component is outside a context

Adaptation Manager (AM). The AM enables the m3-RTE to react to changes context. The AM makes a decision about adaptation if context changes and installs/invokes an appropriate adaptability method. The AM is able to incorporate various types of adaptation.

Adaptation in response to context changes can be either local (i.e. only one application adapts independently from the others) or community (i.e. all the applications adapt together in some coordinated manner). We call these two forms of adaptation *local adaptation* and *community adaptation* respectively. These six dimensions are summarized below.

	local	community
context	local context	community context
context-awareness	local awareness	community awareness
adaptation	local adaptation	community adaptation

Most work to date has focused on local adaptation of an application in response to local awareness of local context [10]. For instance, there is not much work on other aspects such as local adaptation in response to local awareness of community context (i.e., an application adapts according to its knowledge of the contexts of the other applications on other computers it is interacting with. Our approach to tackle community adaptation is to specify dependencies (chain) of execution between adaptation rules. At this stage our dependency is sequential (A then B) but we plan to have concurrent triggering of adaptation rules. A dependency is modelled as a graph where nodes are adaptation rules. The AM prevents the formation of cycles in the graph as it represents a ping-pong effect.

The AM specification is based on `Event Condition Action` (ECA) rules. `Event` is an event that might trigger `Action`. `Conditions` are conditions related to the event. Events are often generated by the context manager. `Action` is a reference to an adaptation action. Such rules support multimedia (continuous) and operational (discrete) adaptation and decouple the rules from the actual implementation of the adaptation. It also supports the specification of application aware adaptation [20] and application transparent adaptation [24,3] by respectively giving the control and implementation of the (A)ction of an ECA rule to the application or to the m3-RTE (see Section 7.1).

The Adaptation Manager provides the following interfaces:

- `Boolean AddECARule(ECARule r)`: Adds an adaptation rule. The rule r is added to the Adaptability Manager.
- `Boolean RemoveECARule(ECARuleID rid)`: Removes an adaptation rule.

- `ECARuleIDSet FindECARules(Criteria c)`: Retrieves an adaptation rule. A criterion c (e.g. keywords to search in rule descriptions or event descriptions) is used to search for ECA rules. The set of ECARuleIDs whose criteria match c is returned. This set could be empty if there are no rules found.
- `Result Adapt(Event e, Parameter p)`: Invokes the Adaptability Manager which will make an adaptation choice.
- `ECARuleIDSet findECARules (Event e)`: This function queries the rules on which an event e can be possibly executed. The function returns a set of ECARuleID, which can be used to select appropriate rules for execution.
- `Boolean Depend(ECARule a, ECArule b, seq | conc)` create a dependency between the execution of the two rules. The Dependency can be parallel or sequential. Return false if a cycle of dependency is detected. A cycle might generate an infinite loop.
- `result ap-aware(condition, function)` call function each time condition is true, this function is mapped to AddECARule.

Policy Manager (PM). The Reference Model for Open Distributed Processing (RM-ODP) Enterprise Language [16] comprises concepts, rules and structures for the specification of a community, where a community comprises objects (human, software, or combinations thereof) formed to meet an objective. The Enterprise Language focuses on the purpose, scope and policies of the community, where an enterprise policy is a set of rules with a particular purpose related to the community's purpose. We see such an approach as providing useful concepts for high-level modelling of a distributed system in general and pervasive systems in particular. The goals of roles can change due to a lack of resources required to achieve a task, for example. This implies that policies can be altered and changed dynamically depending on the context. The policy manager ensures that policy specification described as *Obligation, Prohibition and Permission* associated with enterprise roles are respected in pervasive environments. An example of policy specification is `Prohibited (Role, Community, Action, Time)`. It specifies that a `Role` in a `Community` is `Prohibited` to do `Action` during `Time`.

The policy manager enables consistent addition, deletion and modification of policies. The Policy Manager is also responsible for maintaining consistency and resolving conflicts between roles objectives. (e.g. a role cannot be Authorised and Prohibited to do an action at the same time.). It also prioritises adaptation rules and context of interest based on objectives.

The Policy Manager provides the following services:

- `Boolean CheckPolicy(CommunityID, Roles,URI)`: Check policy
- `Boolean AddPolicy(CommunityID, Roles,URI)`:Add policy rules
- `Boolean RemovePolicy(CommunityID, Roles,URI)`:Remove policy rules
- `PolicyRuleIDSet RetrievePolicy(CommunityID, Roles,URI, criteria)`: Retrieve policy rules based on the criteria.

5.3 Service Layer

A dynamic and open environment is characterised by the frequent appearance of new services and network protocols. A unified mechanism is required to access services and network protocols. This layer wraps services and network protocols to fit into m3 modelling requirements. The wrapping allows upper layers to interact with the service layer using the event-based paradigm. This layer provides:

- at least the notification and discovery services. All the services fulfill a role, accessed as components using event notification service. The discovery service communicates via the notification service to the rest of the m3-RTE.
- a universal interface to the upper layers that enables the use of communication protocols such as SMTP, HTTP, TCP/IP or security protocols transparently. The protocol independent interface is similar to the Linda tuple space model [15,11]. It has generic interfaces `in(from,data,attr)` and `out(to,data,attr)` to communicate with components.

5.4 Interactions between Components

All components of the m3-RTE have to register to a service discovery server as illustrated in Figure 4. Components use a lookup service operation to get the reference of a registered service and interact with it. The reference is used to join the service. The `join` request will generate a unique channel ID in which peers will interact. The in/out operations are used to exchange data.

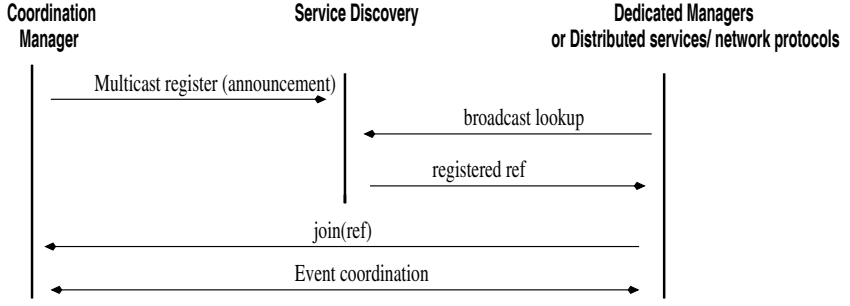

Fig. 4. m3-RTE initialisation

The discovery service registers only active tasks (services). A JINI like lease mechanism is used to achieve this.

Once all the components are initialised and registered, the m3-RTE can receive calls from an application. Figure 5 illustrates an example of a m3-RTE processing of an `http(GET(URL))` request from an application. Note that the coordination manager was programmed as illustrated which is not a mandatory

interaction (e.g we can change the coordination script and avoid using the policy manager). The Coordinator Manager receives the request and forwards it to the PM to check authorisations. If authorisations are granted then the request is sent to the AM. In this script, the AM is constantly updated by the CM about the context of the application. The AM selects an adaptation rule based on context and the adaptation action is executed. The reply is sent directly to the coordination manager which forwards it to the application.

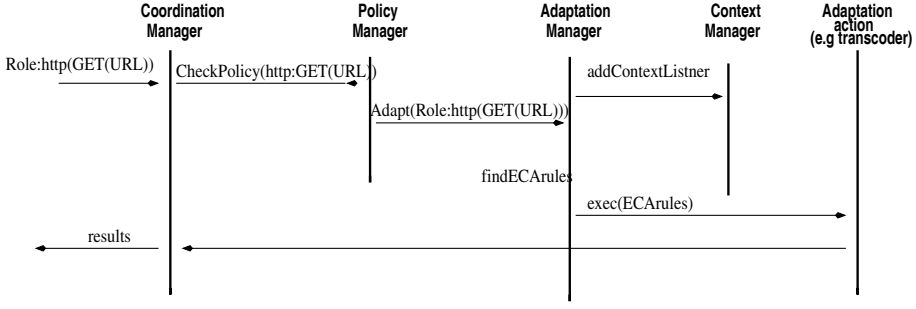

Fig. 5. Interaction between m3-RTE components

The python Code 7.1 relieves the user application (AP) from having to poll the value of the bandwidth. The AM will do it on behalf of the AP. Note that the code 7.1 does not prevent the AM, CM or PM to take actions provided that it has been programmed by the system programmer. For example Application transparent adaptation has to be programmed explicitly in MEADL which coordinates the other three managers.

6 m3-RTE Implementation Issues

This section shows how the Coordination Manager uses the Elvin [6] notification service to implement the execution of MEADL/XML scripts.

6.1 The Use of Elvin

Using point-to-point communication models such as RPC (JAVA-RMI) leads to rather static and non-flexible applications. Event notification reflects the dynamic nature of mobile applications. Interaction components are symmetrically decoupled. Such decoupling and content-based addressing features are the key to the scalability of a mobile platform. Event notification is the building block of the m3-RTE. The concept consists of sending a message to an intermediate system to be forwarded to a set of recipients. The benefits are: reduced dependencies between components and as a result greater flexibility. DSTC Elvin [6]

takes this simple concept and extends it to provide a routing service which is dynamic, scalable, and applicable for a wide variety of communication needs.

Elvin is a notification/messaging service with a difference: rather than messages being directed by applications from one component to another (address base), Elvin messages are routed to one or more locations, based entirely on the content of each message. Elvin uses a client-server architecture for delivering notifications. Clients establish sessions with an Elvin server process and are then able to send notifications for delivery, or to register to receive notifications sent by other components. Clients can act, anonymously, as both producers and consumers of information within the same session. The Elvin subscription language allows us to form complex regular expression.

Elvin is used to exchange events within the m3-RTE. MEADL expressions are parsed and transformed into Elvin subscriptions using Python [21]. Elvin's data is encoded in W3C Simple Object Access Protocol (SOAP 1.1) [25]. Such a choice provides for the use of different communication protocols in the future.

6.2 m3-RTE Prototype

The goal of the m3-RTE prototype is to create a test bed run-time environment for our current and future concepts in pervasive computing. The prototype includes the following managers:

Coordination Manager.

Code 6.1 summarises how the MEADL specification 5.2 is transformed into an XML specification. The XML expression in Code 6.1 is parsed with a Python [21] XML parser. Python Elvin subscription expressions are then generated from the XML parsing.

Context Manager. Currently, the context manager is able to manage device descriptions and location information in an RDF (Resource Description Framework) graph and location information for users and devices. This information has to be gathered from a variety of physical sensors (location devices, e.g. GPS, badges) and logical sensors (extracting/extrapolating location information from operating systems and communication protocols). A location service has been prototyped to gather location information from a variety of sensors, to process this information and convert it to a common format before submitting it to the CM. Our prototype comprises agents processing location data from several location devices and from the Unix and Windows operating systems. Its architecture allows easy incorporation of new sensor agents.

Adaptation Manager. Several adaptability methods have been prototyped including insertion of a variety of filters, migration of objects from resource poor sites to the static network and sending the result of the operation back to the object's client, restriction of an object interface (e.g. allowing access to a subset of operations only) and adaptation of Web content [12].

Policy Manager. The Policy Manager extends context information by adding descriptions of roles and policies associated with roles. Our Policy Manager prototype is in an early stage of development and is currently able to manage

Code 6.1 XML representation of MEADL

```
<rule>
       <trigger>
           <relation rel="and">
               <event name="http:get">
                   <argument ref="#action" />
                   <role="Director" />
               </event>
               <relation rel=&gt>
                    <argument ref="#action" />
                    <literal type="string">10,000</literal>
               </relation>
           </relation>
       </trigger>
       <guard>
           <not>
               <in context="secure_environment" />
           </not>
       </guard>
       <reaction>
           <call function="foo">
               <argument ref="#action" />
           </call>
       </reaction>
 </rule>
```

simple security oriented policies such as Permission and Prohibition. We use concepts from MEADL to specify simple policy rules. The following example shows an authorisation to the role Director to do an action if the Director belongs to the DSTC community and the context of the role is in the Head Quarters.

Code 6.2 Example of Policy specification

```
ON EVENT Director:action
        WHERE (Director.community = 'DSTC')
        IN CONTEXT ``HQ''
    {
     AUTHORIZE(action)
    }
```

7 Examples of Applications Using m3-RTE

The first example demonstrates the ability of m3-RTE to support application aware adaptation according to the terminology used in [20]. The second shows an application "transparent" adaptation.

7.1 Application Aware Adaptation

This section shows an example of the use of interfaces of m3-RTE managers. It shows code that implements the concept of application aware adaptation using m3-RTE in Python. AM and CM modules are imported and the ap-aware (application aware) method of the AM is called with "Callback" as parameter, Callback is a function that belongs to the application. The Callback function will be called once the bandwidth is `CM.bandwidth < 33`. The ap-aware expression is mapped directly onto an Elvin notification expression.

Code 7.1 Application aware adaptation

```
import AM  /* import Adaptation Manager */
import CM  /* import Context Manager context variable such as bandwith */
main()  /* main of the client AP */
  AM.ap-aware(CM.bandwith < 33 ,Callback)

def Callback()
   print ‘‘AP callback adaptation function ’’
```

7.2 Tickertape

This application shows how the m3 architecture adapts the Tickertape user interface rendering and the communication protocol according to the device capabilities and user preferences provided by the CM.

Fig. 6. Unix X Windows

Tickertape is an existing application that gives visual scrolling display of Elvin (event) traffic (see Fig 6). It is a tailorable window that can be used to both produce and consume notifications as desired. It displays notifications that the user subscribers to. In this prototyping, we use it for chat-style messages. Scrolling messages can have a MIME attachment that the user can view with a mouse click.

The demonstration consists of delivering the relevant Tickertape notifications with the appropriate communication protocol to four *existing* applications installed in three devices (Palm, Solaris Ultra10, Nokia7110). Note that this

example is not just a transcoding exercise as communication protocols and components involved in the delivering changes significantly for each adaptation.

We programmed the m3-RTE as follows. The CM has an abstract description of the class of devices such as Palm, Solaris Ultra10, Nokia7110 that contains their capabilities (e.g. WAP 1.0, WAP 1.2, touch-screen, Netscape). The AM has the adaptation rules that select a required component to exchange Elvin messages with the appropriate device (e.g. selection of proxies, WAP gateways). The MEADL script executed by the Coordination Manager forwards the request to the AM and is specified as follows:

Code 7.2 MEADL specification of Tickertape application

```
ON EVENT Project_leader:Ticker-connect(Elvin-server)
        IN CONTEXT ‘‘Palm’’ or ‘‘Nokia7710’’ or ‘‘Solaris’’
    {
      EMIT Adapt(Project_leader:Ticker-connect(Elvin-server));
    }
```

The list of subscribed channels for roles is known by the CM from the user profile. The content of Elvin messages to be forwarded and the communication protocols to be used are chosen by the AM. As a result Tickertape can be adapted in four different ways. When a user uses:

- a Unix (Solaris Ultra 10) workstation, the user sees tickertape message as in Figure 6. It uses X11. Messages and associated MIME type scroll in the bar. The Elvin protocol uses TCP.
- a non WAP enabled palm device, the user sees Figure 7(a). The communication protocol is a phone line connected to a modem and then to the server via HTTP. The Palm user interface cannot scroll messages. The associated MIME types are not transfered. The user has to click the receive button to check the latest update due to the absence of any push mechanism. A proxy component is involved in the interaction. The proxy provides persistence for Elvin by remaining subscribed on behalf of clients even while the client is off-line. When clients reconnect, stored notifications are delivered by the proxy to the clients.
- WAP 1.1 enabled 7710 Nokia Mobile phone, the user sees Figure 7(b). We use Kannel 1.0 (open source) to connect mobile devices to WAP servers. WAP (Wireless Application Protocol) is used to provide services on mobile phones by enabling mobile devices to function as simple web browsers. Kannel 1.0 implements the WAP 1.1 specification gateway to connect a phone line to the Elvin server. The WAP optimised protocols are translated to plain old HTTP by a WAP gateway. At the time we were writing the Nokia 7710 prototype there was no WAP browser that supports the push mechanism. Therefore the user had to press on a receive button to read tickertape messages. Note that MIME types are not sent.

– WAP 1.2 browser, the user sees Figure 7(c). We extend an open source WAP browser (ja2wap) in order to have the first WAP browser supporting the push mechanism compliant with the WAP forum specification. The browser automatically displays tickertape messages and the user does not need to press a button to load messages. MIME type attachments are not sent.

(a) Palm

(b) Nokia 7710 WAP 1.1

(c) Palm- WAP 1.2 - push

Fig. 7. Tickertape on mobile devices

8 Benefits of the m3 Architecture

The described examples allowed testing of some of the m3 architecture goals. The focus of the Tickertape example has been on the "plumbing" required to integrate multiple forms of data delivery to the application level. The m3-RTE prototype emphasises openness and generality of context description and management. The openness of the m3 architecture is demonstrated by the fact that we only added less than ten lines of code into the MEADL and AM to plug in a new component that enables tickertape viewing on different devices[2]. Such conciseness also demonstrates the expressiveness of our coordination language MEADL. Note that the adaptation mechanism can be more complex. For example some sensitive tickertape messages must not be sent to particular roles due to their physical location. Hence, m3-RTE features a universal approach to adaptation which can support a reasonable set of adaptation mechanisms such as discrete, continuous and community adaptation.

One of the main strengths of the coordination language MEADL is that it simple, role-based and can be translated into XML. Furthermore the use of an event based publish/subscribe concept as a core modelling concept allows

[2] This doesn't include the code required to write the actual components that implement the new communication and protocols which is a different engineering effort.

dynamic definition and management of event coordination between the Context Manager, Adaptation Manager, Policy Manager and other component based applications. The use of a publish/subscribe protocol eases the deployment of a distributed version of the m3 architecture. To our knowledge no middleware architecture that integrates and coordinates context, policy and adaptation to support reactive components exists.

9 Conclusion and Future Work

This paper describes an open architecture used as a toolkit to build adaptable enterprise applications for pervasive environments. It leverages heterogeneous devices, existing mobility related standards and protocols. We have identified context, adaptation and policy management as fundamental elements of pervasive computing applications, and shown how our architecture integrates these three notions. The use of a role-based component model and MEADL coordination gives the m3 architecture the ability to provide dynamic plug and play, and yet effectively coordinate enterprise components. The use of Elvin as the main communication paradigm within the m3 architecture enables a loose form of integration of components (or roles), permitting architectural reconfigurability as reactions to the ever-changing contexts of applications or occurrences of (a)synchronous events in a pervasive environment. A different way of building applications is also proposed, where the behaviour of the application is explicitly dependent on separately specified relevant context changes, required adaptations and behaviour policies. An application is therefore constructed by defining (and mixing and matching) these elements rather than by traditional monolithic application building.

As a next step, work is being carried out on both an extension to the prototype and on further adaptive and context-aware applications [13] in order to fully capture and test the concepts introduced by the m3 architecture to support pervasive computing environments. We also are distributing the architecture presented in Figure 3. Each node (or device) will have its own set of dedicated managers (context, policy and adaptation managers) which communicate to coordinate adaptations across several nodes. This permits a set of applications running on possibly different devices to adapt and react cooperatively. Global policies would then be needed which are disseminated to the nodes and integrated with the local policies. We also plan to evaluate the performance of the m3 architecture and compare it with conventional adaptive middleware.

References

1. Acharya, A. Ranganathan, M., Saltz,J. "A language for Resource-Aware Mobile Programs" *Mobile Object Systems: Towards the Programmable Internet, pages 111-130. Springer-Verlag, April 1997. Lecture Notes in Computer Science No. 1222.*
2. Banavar, G., Beck, J., Gluzberg,E., E., Munson, J., Sussman, J. and Zkowski, D. "Challenges: An application Model for Pervasive Computing" *6th Proc annual Intl. Conference on Mobile Computing and Networking MOBICOM 2000, Boston August 2000*

3. Bianchi,G., Campbell, A.T, Liao, R. " On Utility-Fair Adaptive Services in Wireless Networks" *Proc of the 6th Intl Workshop on QoS IEEE/IFIP IWQOS'98 Napa Valley CA, May 1998*
4. Blair,G,. Blair,L.,Issarny, V,. Tuma, P,. Zarras, A,. The Role of Software Architecture in Constraining Adaptation in Component-Based Middleware Platforms. *Middleware 2000 Proc LNCS 1795 - IFIP/ACM NY, USA, April 2000*
5. Composite Capabilities/Preference Profiles CC/PP - W3C - `http://www.w3.org/Mobile/CCPP/`
6. Arnold. D., Segall, B., Boot,J., Bond,A., Lloyd,M. and Kaplan,S Discourse with Disposable Computers: How and why you will talk to your tomatoes, Usenix Workshop on Embedded Systems (ES99), Cambridge Massachusetts, March 1999 also `http://elvin.dstc.edu.au/`
7. Davies, N. , Friday, A.Wade, S. and Blair, G. "A Distributed Systems Platform for Mobile Computing" *ACM Mobile Networks and Applications (MONET), Special Issue on Protocols and Software Paradigms of Mobile Networks, Volume 3, Number 2, August 1998, pp143-156*
8. Demers,A., Petersen, K.,Spreitzer, M., Terry,D., Theimer, M., Welch,B. "The Bayou Architecture: Support for Data Sharing among Mobile Users" *Proceedings of the Workshop on Mobile Computing Systems and Applications, Santa Cruz, California, December 1994, pages 2-7.*
9. Anind K. Dey. "Enabling the Use of Context in Interactive Applications" *Doctoral Consortium paper in the Proceedings of the 2000 Conference on Human Factors in Computing Systems (CHI 2000), The Hague, The Netherlands, April 1-6, 2000, pp. 79-80.*
10. C. Esftratiou, K. Cheverst, N. Davies, A. Friday, An Architecture for the Effective Support of Adaptive Context-Aware Applications, in Proc. of 2nd International Conference on Mobile Data Management, Hong-Kong, January 2001. Lecture Notes in Computer Science, Vol 1987.
11. Eric Freeman, et al JavaSpaces(TM) Principles, Patterns and Practice *The Jini(TM) Technology Series June 1999* also `http://www.sun.com/jini/specs/jsspec.html`
12. K. Henricksen, and J. Indulska., "Adapting the Web Interface: An Adaptive Web Browser", Proceedings Australasian User Interface Conference 2001, *Australian Computer Science Communications, Volume 23, Number 5, 2001.*
13. K. Henricksen, J. Indulska and A. Rakotonirainy "Infrastructure for Pervasive Computing: Challenges", Workshop on Pervasive Computing and Information Logistics at Informatik 2001, Vienna, September 25-28, 2001.
14. J. Indulska, S.W. Loke, A. Rakotonirainy, V. Witana, A.Zaslavsky "An Open Architecture for Pervasive Systems" The Third IFIP WG 6.1 International Working Conference on Distributed Applications and Interoperable Systems September 2001 Krakow.
15. Wyckoff, P., McLaughry,S. W., Lehman, T. J. and Ford,D. A. "TSpaces" *IBM Systems Journal, August 1998* also `http://www.almaden.ibm.com/cs/TSpaces/`
16. Information Technology - Open Distributed Processing - Reference Model - Enterprise Language (ISO/IEC 15414 — ITU-T Recommendation X.911) July 1999
17. Joseph A., Kaashoek F. "Building reliable mobile-aware applications using the Rover toolkit" *MOBICOM '96. Proceedings of the second annual international conference on Mobile computing and networking, pages 117-129'*
18. Kon, F. et al Monitoring, Security, and Dynamic Configuration with dynamicTAO Reflective ORB *Middleware 2000 Proc LNCS 1795 - IFIP/ACM NY, USA, April 2000*

19. Medvidovic N, Taylor "A Framework for Classifying and Comparing Architecture Description Language " *Proc Software engineering Notes, ESEC/FSE'96 - LNCS Vol 22 number 6 November 1997*
20. Noble, B., Satyanarayanan, M., Narayanan, D. Filton J.E, Flinn J.,Walker K. , "Agile Application Aware Adaptation for Mobility" *16th ACM Symposium on Operating System Principles 1997*
21. Python programming Language `http://www.python.org`
22. Renesse,v-R. Birman, K., Hayden,M,., Vaysburd, A,., Karr, D. "Building Adaptive systems using Ensemble" *Cornell University Technical Report, TR97-1638, July 1997.*
23. Rakotonirainy A., Bond A., Indulska,J,. Leonard.D. SCAF: A simple Component Architecture Framework. *Technology of Object-Oriented Languages and systems TOOLS 33 - June 2000 - IEEE Computer Society - Mont St Michel France*
24. Satyanarayanan, M. The Coda Distributed File System *Braam, P. J. Linux Journal, 50 June 1998*
25. Simple Object Access Protocol (SOAP) 1.1 `http://www.w3.org/TR/SOAP/`
26. Sun One brings mobile intelligence to the wireless world `http://www.sun.com/2001-0710/feature/`
27. Extensible Markup Language (XML) 1.0 `http://www.w3.org/XML/`
28. Want, Z. and Garlan D., "Task-Driven Computing". *Technical Report, CMU-CS-00-154, School of Computer Science CMU May 2000*
29. Wireless Application Protocol - WAP Forum Specifications `http://www.wapforum.com/what/technical.htm`