Policy-Driven Binding to Information Resources in Mobility-Enabled Scenarios

Paolo Bellavista[1], Antonio Corradi[1], Rebecca Montanari[1], Cesare Stefanelli[2]

[1]Dipartimento di Elettronica, Informatica e Sistemistica
Università di Bologna
Viale Risorgimento, 2 - 40136 Bologna - Italy
{pbellavista, acorradi, rmontanari}@deis.unibo.it
[2]Dipartimento di Ingegneria
Università di Ferrara
Via Saragat, 1 - 44100 Ferrara - Italy
cstefanelli@ing.unife.it

Abstract. The widespread diffusion of mobile computing and of portable devices with wireless connectivity identifies new challenging scenarios for the Internet provisioning of information services. The possible mobility of users, terminals, and even middleware/service components requires solutions to handle properly the links to information resources in response to the mobile entity migration. Binding decisions may depend on dynamic deployment conditions, e.g., local availability of resources, user preferences, and terminal hardware/software characteristics, and should be determined at service provision time. There is the need for novel middlewares capable of supporting mobility-enabled resource binding and of cleanly separating the application logic from binding strategies. The paper presents a middleware, called SCaLaDE, that supports the accessibility of mobile users/terminals to information services. SCaLaDE provides mobile clients with mobile agent-based proxies that can follow the user/terminal roaming and have their resource references transparently accommodated by the middleware depending on policy-driven binding strategies expressed in a high-level specification language and separated from the application logic. This separation of concerns is crucial to reduce the complexity and leverage the development of mobility-enabled information services.

1. Introduction

The ubiquitous availability of Internet points of attachment and the growing market of mobile devices with wireless connectivity present new challenging scenarios for service provisioning in contexts of mobility. Mobility-enabled services require to re-think and to re-design traditional middleware solutions to accommodate effectively the increased dynamicity stemming from the possible mobility, the temporary disconnection and the wide heterogeneity of client users/terminals [1, 2].

On the one hand, mobility encourages the development and deployment of new classes of information services that can take into account the relative position of users,

M.-S. Chen et al. (Eds.): MDM 2003, LNCS 2574, pp. 212-229, 2003.

service components and available resources. For instance, a city guide assistance service should dynamically retrieve maps, audio descriptions, and detailed textual information about the buildings and restaurants close to the current physical position of the user. In the following, we will call location-dependent the information services that provide results also determined by the physical position of the user/device point of attachment to the network and by the consequent direct accessibility to the locally available set of information resources.

On the other hand, the enlarging market of portable devices is stressing the asymmetry between the resource availability (in terms of memory, computing power, network bandwidth, ...) at the client side and at the server side. Service provisioning to portable devices should consider the strict limitations on their hardware/software characteristics and their wide heterogeneity. For instance, a tourist equipped with a portable device would like to access information about the nearby artworks; already available Web services can provide very detailed multimedia tourist information that should be downscaled to fit the bandwidth and visualization capabilities of the portable device.

All the above requirements call for novel middlewares with increased dynamicity, capable of activating hosts in the Internet infrastructure to support mobility-enabled service provisioning [3, 4]. Few solutions are starting to explore active middleware components that can be deployed at service provision time to act over the fixed network on behalf of users/devices. The middleware components can perform both application-independent operations, e.g., dynamic installation and disconnection support, and application-specific ones, e.g., service configuration and tailoring. Recent research activities are recognizing the relevance of implementing middleware components as mobile entities that follow the client movements to offer the needed support only where and when needed [2, 5]. Mobile middleware components can operate autonomously to carry on service requests even in case of temporary device disconnection, can support location-dependent service provisioning, and can adapt service results to fit specific device characteristics.

It is also crucial to provide middleware solutions to handle properly the links to information resources in response to the migration of mobile users, terminals, or middleware/service components. Anytime a mobile entity changes its location, it is necessary to adopt a suitable strategy to update its references to the needed information resources: a mobile entity could take the needed resources (or their copies) with itself; or it could transform its references to resources in the origin locality into remote resource references when re-connected at the destination locality; or it could re-qualify its old resource bindings to new suitable resources in the new hosting locality. The choice of the most proper re-binding strategy requires the visibility of the location of mobile users/terminals and information resources; in addition, it may depend on dynamic deployment conditions and, therefore, should be decided only at service provision time.

The paper claims the need for novel location-aware middlewares capable of supporting the flexible binding of mobility-enabled services to information resources and of cleanly separating the application logic of information services from the binding strategies. This separation of concerns is crucial to reduce the complexity and leverage the development of mobility-enabled information services in different, statically unknown, usage scenarios. To this purpose, the paper proposes to separately

specify binding strategies via high-level policies, which can be modified, deployed and activated dynamically with no impact on the implementation of service components.

We have designed and implemented a middleware, called SCaLaDE (Services with Context awareness and Location awareness for Data Environments), that supports the accessibility of mobile users/terminals to information services. SCaLaDE provides mobile clients with location-aware proxies, implemented as Mobile Agents (MAs), that can follow the user/terminal roaming and have their resource bindings accommodated after migration depending on location-dependent policies expressed via a declarative policy language. The paper specifically focuses on how different resource binding strategies are supported in SCaLaDE as the result of the dynamic enforcement of different policies.

The rest of the paper is structured as follows. Section 2 presents several usage scenarios that show the need for different re-binding strategies depending on dynamic deployment conditions. Section 3 gives a general SCaLaDE overview, by concentrating on the middleware components for the MA binding management and the policy enforcement. Section 4 shows how SCaLaDE can support the usage scenarios sketched in Section 2 via the enforcement of declarative resource binding policies, by showing the improved flexibility, dynamicity and re-usability deriving from a policy-based approach. Related work, lessons learned and directions of on-going work end the paper.

2. Possible Usage Scenarios for Mobility-Enabled Information Services

Both in the case that a mobile user/terminal accesses resources directly and in the case it exploits the mediation of a mobile proxy hosted in the fixed network infrastructure, mobility-enabled services impose to face the issues of managing the bindings of mobile entities to the needed information resources. In the following, for the sake of simplicity, we will always refer to service architectures where mobile terminals directly access information resources. The same discussion on resource binding applies, with very slight modifications, to novel service architectures where mobile proxies access the resources and roam in the distributed system in response to the movement of the mobile users/terminals they are working for [6].

A mobile terminal MT can refer information resources through various types of binding, each type describing how the MT mobility impacts on its bounded resources. It is possible to identify four possible binding strategies [7]:

- resource movement strategy. In this case, when MT moves, its bounded information resources are moved along with it. Obviously, this type of binding is possible only if the resource transfer is technically/semantically possible, and requires to handle properly the modifications of other possible client bindings to the moved resources;
- copy movement strategy. When MT changes its point of attachment, copies of its bounded resources are instantiated and transferred along with it. This type of binding is permitted only if the resource copy is technically/semantically possible,

and may require to merge the updates and to resolve conflicts in case of concurrent modifications on multiple resource copies;

- remote reference strategy. This strategy requires to modify MT resource bindings after its migration to refer remotely the resources, which are not moved from the source location. Any MT operation on bounded resources requires a network communication with the remote execution environments hosting the information resources;
- rebinding strategy. In this case, the MT movement has to trigger a re-establishment of bindings to equivalent resources available in the new locality of attachment. This strategy can apply not only in case of by-type bindings [8] but also anytime MT is interested in accessing already existing disseminated instances of information resources, whose content possibly depends on the instance location.

The above classification for resource binding types is well recognized in the mobile objects/agents domain where different research activities have proposed similar classifications, with minor differences and variations of the names associated to binding types [8-10].

Let us start to examine the specific issues related to information resource binding in contexts of mobility by presenting some different usage scenarios that point out how the most suitable binding strategy often depends on dynamic deployment conditions.

2.1. Usage Scenario 1: On-Board Resources

Suppose that MT is a fully-equipped laptop with no strict constraints on locally available resources, from free space on the file system to memory resources and computing capabilities. MT can decide to move/copy a data resource DR (or a part of it) on-board and work in a completely local way (resource/copy movement). This enables MT to operate on DR also when MT is disconnected from the network. In particular, in case of multiple clients working with write permissions on their local DR copies, a copy/movement strategy requires to merge the updates and to resolve possible conflicts when DR copies are concurrently modified by multiple clients.

The resource/copy movement strategies can be suitable if the information service has to support disconnected operations or if the costs associated to the network traffic of DR migration are less than the ones due to the sequence of service requests/replies between MT and DR in its original location. However, depending on dynamic deployment conditions, moving/copying DR can be either unsuitable or not absolutely feasible. For instance, if MT is going to move close to the DR position, it can be unsuitable to move/copy DR on-board. Or, if MT is a limited portable device with no storage resources (or MT currently has insufficient free space on its file system) it is not possible to enforce the resource/copy movement strategies.

2.2. Usage Scenario 2: Moving Resources Locally to the Client

If MT is a wireless personal digital assistant or a smart phone with strict limitations on local computing resources and connectivity bandwidth, it should move/copy DR close to its current point of attachment to the network and access DR by maintaining a remote reference to it. By presuming co-location of MT and DR, MT can perform operations on data also in case of network partitioning between the current MT locality and the original DR position; however, this imposes MT to maintain a continuous connection to the network.

Similarly to the previous scenario, the suitability of a remote reference strategy in conjunction with the resource/copy movement towards the current MT locality in place may depend on dynamic deployment conditions. For instance, it is influenced by network-specific deployment conditions such as the possible high heterogeneity in the average network bandwidth available for MT local connections vs. MT remote connections: if MT remote connections are significantly more expensive than local ones and MT is expected to interact several times with DR for a long interval, then moving/copying DR can be preferred. In addition, the binding type decision should take into consideration application-specific expected client behavior, possibly determined by the past history of the user interaction with the service. If one client exhibits a high ratio of writing/reading operations, then the access to DR via a remote reference can be convenient to avoid the overhead needed to resolve possible conflicts among concurrent modifications.

2.3. Usage Scenario 3: Location-Dependent Information Services

Novel location-aware information services should provide mobile clients with results that depend on client position. For instance, a location-aware movie-info service should provide data (title, director, main actors, theatre, starting hour, trailer, place availability, ...) about the movies shown in the theatres in the area where MT is currently located. In this scenario, information services can deploy different instances of the same type of information resources in the different localities, each instance of resource with the specific data related to its locality. The resources can be considered different and type-equivalent local instances (they usually have the same format, or are expressed according to well-know common rules). In this case, the expected binding strategy is to reconnect MT to the locally equivalent information resources anytime MT changes its locality.

This requires not only to sense the MT disconnection/reconnection to the network as in the two previous scenarios, but also to organize the network topology in disjoint localities in order to track inter-locality MT movements. In addition, the rebinding strategy can significantly benefit from naming solutions, such as discovery-based naming services, that are capable of maintaining information (address and simple configuration data) about resource availability with a local visibility scope and by imposing minimal client-side knowledge about the current hosting environment.

Let us observe that the rebinding strategy can also require to choose among different local instances of information resources, available in the MT locality and all compatible with the MT rebinding needs. In fact, the local resource to rebind can be

identified not only on the basis of simple description attributes, e.g., its type identifier, but also depending on even complex application-specific information and terminal profile of characteristics. For instance, a location-aware information service can provide different resource instances, each one with a different version of data. Any copy represents a differently downscaled version suitable for the access of a specific category of device. In the movie-info service example, there could be different versions for any local database instance, one that includes full movie information, one that does not include multimedia trailers, and one with only textual information; in addition, there could be different versions of the same content expressed in different languages. The middleware should rebind MT to the most suitable resource instance depending on MT hardware/software capabilities and on the preferred language specified by the currently logged user. Such a scenario for the rebinding strategy calls for interoperable formats to describe information resources, terminal profiles and user preferences, and requires to evaluate all these metadata to perform the most suitable rebinding at any MT change of locality.

The above scenarios point out that the decision of the most suitable strategy for the MT binding to information resources requires to evaluate different (and possibly complex) system conditions during service provisioning and anytime MT changes its point of attachment in the distributed system. Putting these evaluations within the service application code forces service developers to have a deep knowledge of even low-level characteristics of the deployment environment and can make the development of mobility-enabled information services extremely complex, long and error-prone.

3. SCaLaDE

SCaLaDE is a novel middleware that supports the provisioning of both traditional Web services and new location-dependent ones to wireless portable devices. The design of SCaLaDE is centered on the distributed deployment of active middleware proxies over the fixed network to support mobility-enabled service scenarios. These proxies dynamically extend the Internet infrastructure, where and when needed, to adapt services to user preferences and system conditions and to perform the needed service configuration/management operations suited to a wide variety of highly heterogeneous client devices. In particular, SCaLaDE provides any portable device with a shadow proxy that is designed to discover service components, to possibly negotiate service tailoring to fit device characteristics, to act on behalf of the device in case of disconnection, and to possibly follow the device movements among network localities by maintaining the session state. The adoption of shadow proxies emphasizes our choice of handling location-awareness at the middleware level to simplify the management of location visibility and of location-dependent service adaptation. Exporting location-awareness management up to the application-level could impose a significant extra-burden to service developers.

We have designed and implemented SCaLaDE middleware proxies in terms of Mobile Agents (MAs) to achieve the crucial properties of mobility, asynchronicity and

autonomy. MAs can act as autonomous entities with a large capacity of coordination, able to dynamically move, together with their code and reached execution state, where the needed resources are located, and able to operate asynchronously to their launching users/devices [2]. In addition, the MA programming paradigm is typically location-aware and MAs have visibility of their execution environment to adapt their actions, primarily their migration, to the position of needed resources.

As a distinguishing feature, SCaLaDE supports the dynamic programmability of binding strategies for shadow proxies. This means that binding strategies can be defined and changed even at run time without any impact on proxy implementation. Toward this goal SCaLaDE adopts a policy-based model: choices in the type of binding between proxies and needed resources are expressed through declarative policies, separated from the middleware proxy code. In particular, binding policies define the circumstances that trigger a binding change for an MA-based proxy, e.g., from the remote reference strategy to the resource movement one, the new type of binding to exploit, and the dynamic conditions that must hold in the deployment environment for resource binding adaptation to take place.

3.1. Resource Binding in SCaLaDE

The proposed middleware for portable devices is implemented on top of the Secure and Open Mobile Agent (SOMA) framework that provides a wide range of support facilities for MA applications, from basic facilities, such as agent naming, communication, migration, to advanced facilities to achieve proper levels of agent security and interoperability. In particular, SOMA execution environments for MAs are called places and several places are grouped into domains that usually correspond to a network locality [11].

SCaLaDE shadow proxies are implemented by SOMA agents. With a finer degree of details, a shadow proxy is implemented by one SOMA agent running on a place in the SOMA domain where the portable device is currently located. Several shadow proxies for different devices can execute concurrently on the same place without interference because the SOMA platform provides isolated execution environments with separated security domains for the different agents.

SCaLaDE complements the SOMA environment with the facilities required to support adaptive binding to resources, whereas it exploits the SOMA support facilities for the execution and migration of SCaLaDE middleware proxies. In particular, the new facilities integrated on top of SOMA include a Binder facility responsible for managing the bindings between incoming MA-based proxies and resources, and a set of policy-enforcement facilities enabling Binder to perform data space management operations accordingly to desired binding strategies (see Section 3.2).

The Binder facility mediates proxy access to resources and dynamically adapts the set of proxy bindings accordingly to the desired strategies. In particular, at start up, SCaLaDE proxies can refer only to the Binder facility without any direct access to resources. At the first resource request, proxies receive from Binder a resource descriptor, i.e., an object that has the same methods and constructor interface of the needed resource. Afterwards, SCaLaDE proxies can operate on resources directly via the obtained resource descriptors.

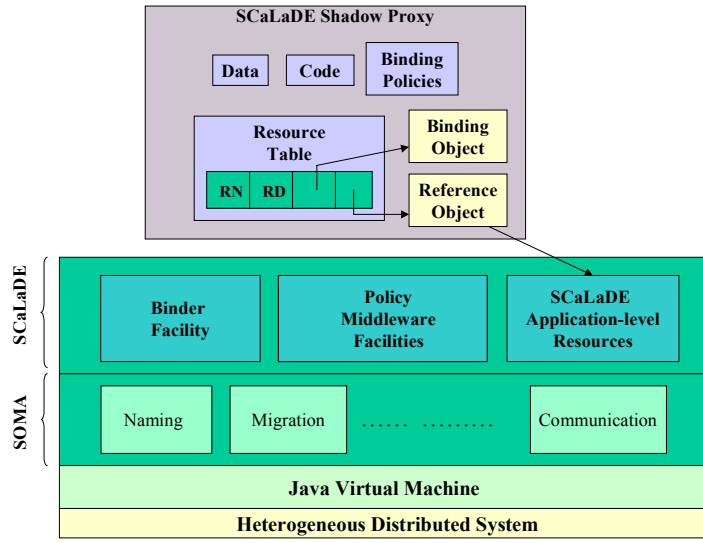

Figure 1. The internal structure of proxy bindings to resources in SCaLaDE.

Currently, SCaLaDE supports policy-driven resource binding only for application-level resources, e.g., database objects, whereas it adopts a default binding strategy for system resources, e.g., files and I/O devices. At proxy migration toward a new node, the proxy bindings to system resources are voided and re-established a the arrival at the new execution environment. In particular, SCaLaDE proxies re-bind to the locally available system resources using Java local references. Our future work will extend the applicability of the policy-driven binding model also to system resources.

Figure 1 depicts the internal structure of proxy bindings to resources. When one SCaLaDE shadow proxy is first instantiated, SCaLaDE creates a resource table to record the list of proxy resource descriptors. In a more detailed view, each table entry associates a resource name RN to the corresponding resource descriptor RD along with a binding and a reference object. The binding object encapsulates the resource binding strategy to apply upon proxy resource usage on the basis of the policies currently enabled for the shadow proxy. SCaLaDE supports all the binding strategies described in Section 2, as detailed in the following. The reference object implements binding mechanisms accordingly to the required resource binding strategy. In particular, the reference object uses Java Remote Method Invocation to refer resources in the case of remote reference or rebinding strategies. For resource movement and copy movement binding strategies, the reference object exploits Java serialization mechanisms to keep valid resource bindings upon proxy migration toward a new place: either the resource object or its copy, respectively, are transmitted as a sequence of bytes to the new execution environment and rebuilt through the deserialization process upon arrival [12].

Let us note that Binder plays a crucial role in the dynamic re-adjustment of resource references upon proxy migration. When a proxy arrives at a new place, the Binder facility of the new execution environment updates, transparently to programmers, the proxy binding object to reflect possibly changes in binding

strategies on the basis of proxy-specific binding policies. In addition, when the new incoming proxy tries to access any resource at the new place, Binder updates the proxy reference object so to modify resource binding implementation accordingly to the possibly new binding strategy. For instance, when a rebinding strategy is set, Binder exploits JINI discovery solutions to obtain information about resource availability and location without any static knowledge of the new hosting environments [13]. If several instances of the same resource type are available, it is the Binder that selects the local resource instance to bind, depending on retrieved user/terminal profiles as detailed in Section 4.

It is worth observing that in the current SCaLaDE implementation the binding decision is evaluated only at the proxy arrival at a new execution environment. This choice represents a trade-off between flexibility and efficiency in binding management. In our opinion, the assumption that proxy-specific bindings do not change while the proxy executes on the same place is reasonable and makes possible an effective implementation.

3.2. The SCaLaDE Policy Middleware

The key feature of SCaLaDE is the distinction between computational and binding aspects of mobility-enabled services. Doing so is an instance of the well-known design principle of separation of concerns. Separation of concerns in SCaLaDE is obtained via the adoption of a policy-based approach. Policies are rules governing choices in the behavior of a system separated from the components in charge of their interpretation [14]. SCaLaDE policies permit to define dynamically the most appropriate binding strategies to apply in order to adapt proxy-resource interactions to the current environment state.

As a consequence of separation of concerns, developers of SCaLaDE proxies do not have to specify at design time the type of binding to apply when the proxy uses resources. The code of MA middleware proxies contains only the directives to retrieve resource descriptors and resource invocation methods when needed. Binding strategies are all defined as policy rules externally to the proxy code; policies can be specified even at run-time with no intervention on the proxy code and are evaluated on the basis of dynamic operating conditions.

SCaLaDE expresses the binding strategies in the Ponder language, which permits the specification of several types of management policies for distributed systems and networks [15]. For instance, Ponder includes authorization policies for access control, obligation policies for proactive management, and organizational policies used to structure responsibility based on organizational models, e.g., role policies. In particular, SCaLaDE uses a subset of Ponder policies, i.e., obligation policies, to specify binding decisions. Obligation policies are declarative event-action-condition rules that define the actions that policy subjects (entities having the authority to initiate a management decision) must perform on target entities when specific events occur if specific pre-conditions hold at event occurrence. The syntax of an obligation policy is shown in Figure 2A.

When applied to binding decisions, the event clause captures a change relevant to binding management operations and the action clause specifies the method that allows

to set the type of binding strategy to activate at event occurrence. Events are related to generic changes in the system and can model any variation in the environment state. For instance, changes in binding strategies can be triggered by variations in the user/device location, by resource method invocations, or by proxy migration.

Figure 2B shows a simple example of a Ponder obligation policy for binding management. Bind1 states that when the policy subject (the MA-based ProxyID shadow proxy) arrives at a new node it should command Binder to set the proxy binding type to the "resource movement" value if the new hosting node has sufficient free space on disk (as observed by the underlying SOMA monitoring facility [16]).

A

```
inst oblig PolicyName "{"
on        event-specification;
subject   subject-Expression;
target    target-Expression;
do        action-list;
when      constraint-Expression ;
"}"
```

B

```
inst oblig Bind1 {
on        Arrival(ProxyID, newNode);
subject   s=ProxyID;
target    t=Binder;
do        t.setAgentBindingType(ProxyID, "resource movement");
when
 MonitoringSystem.getFreeDiskSpace(ProxyID.getLocation())< THRESHOLD;
}
```

Figure 2. Ponder policy syntax (A) and an example of Ponder-based binding policy (B).

The SCaLaDE policy-driven binding model makes necessary the design and development of a policy middleware to support automated binding management accordingly to high-level policy specifications and transparently to proxy developers. SCaLaDE provides a set of middleware services to support the binding policy dynamic specification, modification, check for correctness, installation and enforcement. Policy enforcement consists in detecting changes in the operating environment relevant for binding management, in notifying proxies about event occurrence and in interpreting policy specifications so as to activate low-level binding management actions accordingly to desired binding strategies.

Several tools are available in SCaLaDE to assist users and administrators in the editing of Ponder well-written policies; the same tools can compile Ponder policy specifications and automatically generate lower-level policies in terms of Java objects [17]. Both Ponder policy specifications and correspondent Java objects are published in a distributed LDAP-compliant directory. It is worth noticing that conflicts between policy specifications could arise due to omissions, errors or conflicting requirements of the specifying users. Currently, SCaLaDE faces the problem of conflicts between policy specifications by relying on the solutions provided by the Ponder framework [18].

To support the policy enforcement, SCaLaDE provides the following main services:

- the Monitoring Service (MS) detects modifications in the environment state. MS can observe the state of system/application resources, from the percentage of CPU usage on one place to the allocated heap memory and the network bandwidth consumed by SOMA agent threads. MS exploits the Java Virtual Machine Profiler Interface to enable the observation of the JVM state at the

application level, and platform-dependent monitoring modules via the Java Native Interface to obtain visibility of resource state at the kernel level [16];

- the Event Service (ES) manages and communicates the monitored variations in the whole distributed system. ES typically receives state information from MS, possibly aggregates monitored data in higher level event objects, and notifies events to any entity subscribed for them. Events can be viewed as high-level objects that encapsulate information related to the changes they represent and that can be organized in extensible hierarchies to accommodate application requirements and environment characteristics. As a key feature, ES is designed to take into account MA mobility and to notify events to interested entities, e.g., SCaLaDE proxies, even in case of agent migration;
- the Policy Manager Service (PMS) installs Java-based Ponder policies into policy subjects and enforces them at event occurrence. The PMS component consists of two modules, the Obligation Coordinator (OC) and the Obligation Enforcer (OE). The OC module retrieves new specified policies from the distributed directory and parses them to retrieve relevant information: object events, subjects, targets and actions. Then, it registers the significant events to ES on behalf of policy subjects. At event occurrence, the events are dispatched to all interested policy subjects that trigger binding management actions. It is the OE module that concretely enforces policies on behalf of the policy subjects. In particular, it retrieves the proxy-specific policies, interprets policy specifications, checks policy constraints in the system and if the verification process succeeds, extracts the policy actions and activates their enforcement.

4. SCaLaDE at Work in Different Usage Scenarios

To illustrate how SCaLaDE supports and facilitates the development of mobility-enabled information services, let us re-consider the different usage scenarios sketched in Section 2. In particular, the scenarios will show how it is possible to modify the resource binding strategy of SCaLaDE-based services, depending on requirements specified at service provision time and on dynamic deployment conditions, without any intervention on the code of service components.

As a case study, we have designed and implemented a Mobile News Service (MNS) that permits the reading and browsing of news, automatically retrieved from network servers at regular time intervals. When a user with her mobile terminal MT requests to open an MNS session, a dedicated shadow proxy ProxyID is instantiated. ProxyID runs on MT if MT can host a full Java Virtual Machine and a SOMA place on top of it. Otherwise, ProxyID executes on a host on the fixed network in the same locality where MT is currently attached. In this case, ProxyID automatically follows the MT changes of points of attachment to maintain co-locality with its client [6].

Figure 3A shows an excerpt from the simple, reusable and binding-transparent code of the MNSProxy class. Any shadow proxy executes the init() method at its first instantiation. init() initializes in ProxyID a reference to an information resource called “newspaper”. DataResource.get() produces an invocation of Binder that adds a new line in the ProxyID ResourceTable and returns a resourceID reference to access the

resource. Up to now, we have implemented the DataResource class in SCaLaDE to support HTTP-based access to XML files written according the News Industry Text Format DTD [19]. We work on extending DataResource with other types of data sources, e.g., databases with a JDBC interface. Apart from this initialization, ProxyID repeats a user-specified query on resourceID if the state variable isConnected is set, to visualize freshly obtained results. Other ProxyID threads, not shown in the code excerpt, serve in the user browsing of the results and the insertion of new queries.

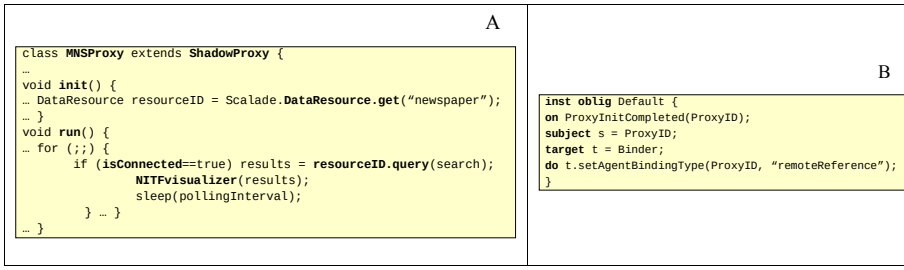

Figure 3. A code excerpt from MNSProxy (A) and `Default` triggered by the ProxyInit-Completed event (B).

The end of the execution of the ProxyID init() method is monitored by SCaLaDE MS to ascertain the notification of a ProxyInitCompleted event. This event triggers the enforcement of a first Default policy, shown in Figure 3B. Default specifies which is the default binding type for information resources: OE, on behalf of ProxyID, makes Binder invoke the setAgentBindingType() method to set to "remoteReference" the binding type for all resources in the ProxyID ResourceTable.

4.1. Usage Scenario 1: Disconnected Operations

Without any modification of the MNSProxy class, it is possible to deploy MNS to support disconnected operations, by specifying the Pol1 policy, shown in Figure 4A, at service deployment time. In any moment, the user can click a button on the MNS GUI to communicate willingness to operate disconnected from the current point of attachment. This generates a disconnectRequest event that ES notifies to ProxyID by triggering the Pol1 enforcement. ProxyID pops up a message window to ask user to wait before disconnecting MT until the resource management process is completed (the current MNS prototype implementation does not handle the physical disconnection of MT before the completion of the resource movement phase). Then, OE forces the isConnected variable to false, to set the binding type to "resourceMove" for any referenced resource in the ProxyID ResourceTable and to possibly update the corresponding reference objects. All the actions specified in Pol1 are performed only if the dynamic condition about the free space available on the MT disk is verified.

updateReferenceObjects()checks the binding type for any reference in the ProxyID ResourceTable and updates accordingly the reference objects. In particular, in case of resource movement of a resource DR, the MT-local Binder coordinates with Binder in the DR locality and requests for the DR serialization and transmission. After DR

reception, the MT-local Binder de-serializes it and puts the (now local) DR reference in ResourceTable. In the current SCaLaDE implementation, the serializing Binder first checks whether DR is already referred by other MNSProxy objects. Only if there are no active references to DR, DR is serialized and moved; otherwise, the binding type is reset to the default "remoteReference" and DR is not accessible while disconnected.

4.2. Usage Scenario 2: Copy Movement to the Client Locality

When MNS is deployed for mobile terminals with very limited local space on disk and in a scenario where their communications with the current locality are extremely cheaper than remote ones, the main goal is the maximization of local interaction with resource copies without disconnected operation support. To enable this, SCaLaDE service providers can simply substitute the Pol2 policy instead of Pol1 (see Figure 4B), without modifying any detail in the MNS implementation. In fact, anytime MT connects to a new point of attachment, MS senses the reconnection and delivers the corresponding connection event to ES. ES processes the reconnection events and, in particular, notifies a domainArrival event only if MT has reconnected to one SCaLaDE domain different from its previous one. The domainArrival event triggers Pol2: OE forces Binder to set the binding type to "copyMove" for any resource in the ProxyID ResourceTable and to set the isConnected variable to true.

Let us stress that Pol2 does not include any Binder action to update the reference objects in the ProxyID ResourceTable. In fact, as already stated in Section 3, Binder automatically performs the update at the first use of one ProxyID resource reference in a new locality, in order not to waste time for re-qualifying resource links not used during the ProxyID execution in one locality. Only at first resource utilization, Binder checks the corresponding resource binding type and, in case of copy movement, coordinate with the remote Binder that is local to the to-copy resource, as described in the previous scenario. In the current SCaLaDE implementation, Binder moves a copy also if the resource is already referred by other MNSProxy instances; this cannot generate inconsistencies in MNS because users can perform only read operations on data resources. To support more general mobility-enabled information services, you need to provide Binder with update merging and conflict solving functions, in order to allow concurrent modifications of distributed copies. We are currently implementing an extended Binder version that supports the consistency restoration schema proposed in [20].

4.3. Usage Scenario 3: Location-Dependent Information Services

A crucial added value for service provisioning in mobile scenarios is the possibility to develop and deploy services that take into account the client position and adapt their results to this location information, e.g., a location-dependent MNS version that offers local news about the current MT locality. To achieve this goal, service developers should usually re-design the service as a location-aware one that directly handles the client changes of location and dynamically discovers the locally available resources. On the contrary, SCaLaDE permits to deploy a location-dependent MNS version

simply by specifying a suitable Pol3 policy, shown in Figure 4C, without changing the MNS code.

A

```
inst oblig Pol1 {
on disconnectRequest(ProxyID);
subject s = ProxyID;
target t1 = Binder, t2 = ProxyID;
do t2.showWaitingMessage() ||
        t1.setIsConnected(ProxyID, false) ->
        t1.setAgentBindingType(ProxyID, “resourceMove”) ->
        t1.updateReferenceObjects(ProxyID) ->
        t2.removeWaitingMessage();
when MonitoringSystem.getFreeDiskSpace(ProxyID.
        getLocation()) > threshold;
}
```

B

```
inst oblig Pol2 {
on domainArrival(ProxyID, newDomain);
subject s = ProxyID;
target t = Binder;
do t.setAgentBindingType(ProxyID, “copyMove”) ||
        t.setIsConnected(ProxyID, true);
}
```

C

```
inst oblig Pol3 {
on domainArrival(ProxyID, newNode);
subject s = ProxyID;
target t = Binder;
do t.setAgentBindingType(ProxyID, “rebind”) ||
        t.setIsConnected(ProxyID, true);
when MonitoringSystem.isDiscoveryAlive() == true;
}
```

Figure 4. Pol1 (A), Pol2 (B), and Pol3 (C) for enabling different MNS provisioning scenarios, i.e., respectively, disconnected operations, local copy movement, and location-dependent news.

The event-related mechanisms are the same as described for Pol2. In this scenario the domainArrival event triggers Pol3 and OE forces Binder to set the binding type to "rebind" for any resource in the ProxyID ResourceTable and to set isConnected. In the example shown in the figure, the actions specified in Pol3 are performed if and only if, at the moment the policy is triggered, there is a discovery server running in the new MT locality.

At the first request for resource usage by ProxyID, due to the binding type set to "re-bind", Binder performs a series of operations very different from the previous usage scenarios. In particular, in this case Binder also considers the access terminal properties (and the user preferences) to rebind MT with the most suitable DR instances in the new locality. This is crucial to permit wide heterogeneity in the hardware/software characteristics of portable devices as they roam between statically unknown hosting environments [6]. First of all, Binder retrieves the terminal/user profiles from the globally available SOMA profile directory [2]. Profiles are expressed in an interoperable XML-based format compliant with the W3C Composite Capabilities/Preferences Profile (CC/PP) specification [21]. Then, for any ResourceTable entry to rebind, Binder interrogates the local discovery service and obtains a list of locally available resources with compatible interfaces. SCaLaDE exploits the JINI discovery protocol to register/de-register resources with the visibility scope of one SCaLaDE domain [13].

If the list of locally available resources includes more items, Binder exploits the profile metadata to choose the resource to rebind. The current SCaLaDE implementation requires the discoverable resources to provide their own CC/PP profile by implementing a getProfile() method. Binder collects resource profiles and compare them with the involved terminal/user profiles; it performs simple parsing and processing operations on the attribute-value pairs contained in the profiles, and increments a score counter anytime a user/terminal profile attribute is compatible with

the corresponding resource one, e.g., "Hardware/DisplayHoriz-Resolution > 320" in the terminal profile and "Hardware/DisplayHorizResolution = 400" in the resource profile. After these comparisons, Binder chooses the resource with the maximum score and updates the corresponding reference object in ResourceTable with the dynamically downloaded Jini-based client stub to the chosen information resource.

Let us finally observe that in this usage scenario the location awareness of the SCaLaDE middleware (and of its Binder component) permits to develop SCaLaDE-based distributed applications that are location-dependent from the point of view of service results but completely location-transparent in their application logic and code. This dramatically simplifies the design and implementation of location-dependent services and can significantly increase the reusability and modularity of service components.

5. Related Work

The effective design and deployment of mobility-enabled services in dynamic environments should consider novel paradigms to provide flexible binding management and its provision-time reconfiguration. While traditional services could be conceived, designed and implemented in quite fixed scenarios of environment conditions, which vary only under exceptional circumstances, novel mobile scenarios clash with static assumptions on available resources and network connections. Network topology can dynamically change, with nodes that are discontinuously connected and users that change location frequently. The conventional approach of hard-coding binding strategies into the service-specific code lacks flexibility and dynamicity. As a consequence, any mobility-enabled service should adapt its execution to all possible operating conditions where it may be deployed.

This section does not provide a general survey on the state-of-the-art solutions for achieving dynamic adaptability of services in contexts of mobility, but focuses on the few proposals that, to the best of our knowledge, explicitly deal with dynamic binding management in mobility-enabled services built with mobile-code technologies. All solutions have in common the principle of separation between binding and computational concerns: this separation is considered as the key design requirement. However, the proposals differ on how to achieve the needed separation of concerns. The approach described in [10] uses reflection for defining customizable application-level binding strategies. In particular, binding strategies are implemented as basic reusable metaobjects that can be attached to any mobile application component. Neither the developers of application components nor the designers of resources are aware of the binding strategies to apply upon component migration. Binding strategies are not embedded within the component or resource code, but implemented by metaobjects. However, the linking between application components and binding strategies is performed at the beginning of the execution, and cannot change at service provisioning time unless an execution re-start. Another interesting approach is proposed by the FarGo system that allows to program various binding relationships between FarGo entities as a separate part of the application [9]. However, similarly to [10], the binding strategies to apply can be associated to entities only at application

launching time. Our proposal has several points in common with the FarGo binding model; the main difference is in the possibility to specify binding strategies at a higher level of abstraction, to modify them even during the application execution, thus enabling dynamic changes in binding management at deployment time, transparently to service developers.

6. Conclusions and On-Going Work

The complexity of service provisioning in mobile scenarios makes more and more relevant the separation of concerns between resource binding strategies and application logic, in order to achieve a high level of dynamicity, flexibility and reusability of middleware/service components. At the state-of-the-art policy-driven middlewares represent a viable and effective solution to realize this separation. In addition, it is crucial that middleware solutions have visibility of location information to take location-dependent support decisions by freeing service developers and application-level service components from all the involved implementation issues.

First experiences in the use of SCaLaDE have shown that our middleware can significantly simplify the design and implementation of services in a wide variety of different mobile usage scenarios. These encouraging results are stimulating further research along different guidelines to improve the current prototype and to develop more complex services on top of it. In particular, we are working on extending SCaLaDE to support flexible binding management even in the case of mobile resources. In the current implementation resource consumers can autonomously migrate across the network, whereas needed resources can be copied/moved only as a result of consumer movements. This implies that service providers cannot relocate resources while shadow proxies are using them.

Another issue that we are investigating is the possibility for SCaLaDE service providers to specify policies with different priorities. When multiple binding policies are simultaneously eligible for enforcement, but only one should be activated in the system, policy priorities may help in choosing the most appropriate binding strategy. For instance, suppose that two policies apply for a resource-limited MT both commanding the MT to move two different resources on-board upon its arrival at a new locality. In the case that only one resource can be moved on-board due to limited disk space,. the choice on which resource to transfer could be driven by policy priorities. The support for precedence between policies requires to extend SCaLaDE to allow both the priority specification and the evaluation of policy precedence at run-time before activation.

In addition, we are interested in a more in depth investigation about the policy conflicts that could arise at run-time when a specific system state results in a conflict. This is still an open and challenging research area with very few solutions. Run-time conflicts are hard to detect in advance because it is extremely difficult to predict all the system possible states.

Acknowledgements

This investigation is supported by the University of Bologna (Funds for Selected Research Topics: "An integrated Infrastructure to Support Secure Services"), by CNR (Funds for Young Researchers: “An Integrated Security Infrastructure for Multimedia Services”) and by MIUR within the framework of the Project “MUSIQUE: Multimedia Ubiquitous Service Infrastructure in a QoS Universal Environment".

References

1. L. Capra , G. Blair, C. Mascolo, W. Emmerich, P. Grace, “Exploiting Reflection in Mobile Computing Middleware”, to appear in *ACM SIGMOBILE Mobile Computing and Communications Review*.
2. P. Bellavista, A. Corradi, C. Stefanelli, "Mobile Agent Middleware for Mobile Computing", *IEEE Computer*, Vol. 34, No. 3, March 2001.
3. H. Chen, A. Joshi, T. Finin. "Dynamic Service Discovery for Mobile Computing: Intelligent Agents Meet Jini in the Aether", *Baltzer Science Journal on Cluster Computing*, Vol. 4, No. 4, March 2001.
4. A. Baggio, G. Ballintijn, M. van Steen, A.S. Tanenbaum, "Efficient Tracking of Mobile Objects in Globe", *The Computer Journal*, Vol. 44, No. 5, 2001.
5. IKV++ Technologies AG, enago Open Service Platform, http://www.ikv.de.
6. P. Bellavista, A. Corradi, C. Stefanelli, "The Ubiquitous Provisioning of Internet Services to Portable Devices", *IEEE Pervasive Computing*, Vol. 1, No. 3, Sept-Oct. 2002.
7. L. Cardelli, "Mobile Computation", in J. Vitek and C. Tschudin (eds.), *Mobile Object Systems: Towards the Programmable Internet*, LNCS 1222, Springer-Verlag, 1997.
8. A. Fuggetta, G. P. Picco, G. Vigna, "Understanding Code Mobility", *IEEE Trans. on Software Engineering*, Vol. 24, No. 5, May 1998.
9. O. Holder, I. Ben-Shaul, H. Gazit, "Dynamic Layout of Distributed Applications in FarGo", *21st Int. Conf. On Software Engineering (ICSE'99)*, ACM Press, USA, 1999.
10. E. Tanter, J. Piquer, "Managing References upon Object Migration: Applying Separation of Concerns", *Int. Conf. Chilean Computer Science Society (SCCC'01)*, IEEE Press, Punta Arenas Chile, November 2001.
11. P. Bellavista, A. Corradi, C. Stefanelli, "Protection and Interoperability for Mobile Agents: A Secure and Open Programming Environment", *IEICE Transactions on Communications*, Vol. E83-B, No. 5, , 2000.
12. W. Grosso, R. Eckstein (eds.), *Java RMI*, O'Reilly, 2001.
13. K. Arnold et al., *Jini Specification*, Addison Wesley Longman, 1999.
14. M. Sloman, “Policy Driven Management For Distributed Systems”, *Journal of Network and Systems Management*, Vol. 2, No. 4, Plenum Press, 1994.
15. N. Damianou, et al., “The Ponder Policy Specification Language”, *Proc. 2nd International Workshop on Policies for Distributed Systems and Networks (Policy 2001)*, LNCS 1995, Springer Verlag, Bristol, UK, 2001.
16. P. Bellavista, A. Corradi, C. Stefanelli How to Monitor and Control Resource Usage in Mobile Agent Systems *Proc. 3rd International Symposium on Distributed Objects & Applications (DOA'01)*, IEEE Press, Rome, Italy, 2001.
17. R. Montanari et al., "A Policy-based Infrastructure for the Dynamic Control of Agent Mobility", *Proc. Third International Workshop on Policies for Distributed Systems and Networks (Policy 2002)*, IEEE Press, Monterey, CA (USA), 2002.

18. E. Lupu, M. Sloman , "Conflicts in Policy-Based Distributed Systems Management", *IEEE Transactions on Software Engineering*, Vol. 25, No. 6, Nov./Dec. 1999.
19. International Press Telecommunications Council, News Industry Text Format, http://www.nitf.org/
20. E. Pitoura, B. Bhargava, "Data Consistency in Intermittently Connected Distributed Systems", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 11, No. 6, 1999.
21. World Wide Web Consortium, Composite Capabilities/Preferences Profile, http://www.w3.org/Mobile/CCPP/