ESPERANTO: a Middleware Platform to Achieve Interoperability in Nomadic Computing Domains

M. Cinque (1), D. Cotroneo(1), C. di Flora (1), A. Migliaccio (1), and S. Russo (1,2)
(1) Dipartimento di Informatica e Sistemistica
Università di Napoli Federico II
Via Claudio 21, 80125- Napoli, Italy
(2) ITEM - Laboratorio nazionale per l’Informatica e la Telematica Multimediali
Consorzio Interuniversitario Nazionale per l’Informatica
Via Diocleziano 328, 80124 - Napoli, Italy
(macinque, cotroneo, diflora, armiglia, sterusso)@unina.it

Abstract

The most challenging issues in Nomadic Computing environments arise from the combination of heterogeneity, dynamism, context-awareness, and mobility. Driven by these issues, this paper presents a new middleware infrastructure, named ESPERANTO, to support the integration of diverse Nomadic Computing domains. This middleware aims to glue the emerging heterogeneous Nomadic Computing technologies and Service Oriented Architectures.

1. Introduction

Advances in wireless technologies and in mobile terminals are leading up to a new computing paradigm that supports the AAA (Anytime Anywhere Access) rule of Nomadic Computing [10]. In Nomadic Computing, communication takes place on very heterogeneous network infrastructures, which consist of one or more wireless domains glued by a fixed infrastructure (the core network). These infrastructures are typically enriched by a set of interacting services, each enabling to access a well-defined set of functionalities. Today, services are often published, discovered, and delivered by means of a Service Oriented Architecture (SOA) [1], consisting of Service Discovery and Service Delivery infrastructures. Hereafter, these network infrastructures along with the adopted SOAs we call Nomadic Computing Domains (NCDs). Several devices – with diverse resources, capabilities, and dimensions – may be dynamically connected to NCDs, and use the services that each domain offers.

For the sake of clarity, let us envision a simple yet meaningful scenario, involving a mobile user in a smart-airport environment (a practical example of a NCD). The user is willing to print a set of documents before departure; some documents are stored onto his Wi-Fi Personal Digital Assistant (PDA), whereas others have to be retrieved through his company’s document management web-service. As he switches the PDA on, its web browser is re-directed to the Airport’s welcome page; upon an e-ticket based authentication, the system provides him with the list of the available services and facilities. The user notices that the system offers more powerful and dynamic services through a Jini-based service infrastructure. Thus, he browses the list of available printing facilities by means of his Jini-browser tool, and he localizes a bluetooth laser color printer within the departure hall he is walking through; he notices that this printing service allows also to print remote web pages, as well as to print documents published on the world wide web as results of a specific Web-service. As the service is requested, the print-service’s graphical client appears on the PDA. The user provides the tool with the local documents to print, as well as with the URI and service parameters to retrieve the remote documents by means of his company’s Web-services. The tool provides him with an Airport Positioning System (APS), which guides him to the printer; moreover, it informs him that the documents will be printed in five minutes.

Although a good deal of research has gone into adapting traditional middleware to mobile environments (as described in Section 2), we claim that time is ripe for an approach shift in middleware development, so as to achieve...
the integration of diverse Nomadic Computing Domains. Indeed, the smart-airport system encompasses diverse Service Oriented Architectures (i.e., Web-Services, Bluetooth, Jini), diverse mobile terminals (e.g., PDAs, smart-phones), diverse wireless short-range technologies (i.e., bluetooth, wi-fi), and several interaction schemes (e.g., synchronous, asynchronous, connected, and disconnected operations). A lot of services reside in the airport system, each belonging to a specific domain and service provider. It is crucial to define proper service filtering mechanisms, since services have to be imported and exported among different domains. Such mechanisms pursue the following objectives: i) to evaluate usability of a certain service (in a domain X) from a domain Y; and ii) to evaluate whether a usable service can be imported in domain Y or not (this is due to the scalability problems of discovery servers). Heterogeneity in its various forms also influences the service delivery infrastructure. For instance, although the user’s PDA is not bluetooth-compliant, he can use this particular printer, thanks to the supporting infrastructures, which allow to deliver a bluetooth service (i.e., the printer) from a Jini-based client application.

This work proposes a new middleware infrastructure, namely ESPERANTO, to support the integration of diverse Nomadic Computing Domains; as this middleware aims to glue the emerging heterogeneous Nomadic Computing technologies and Service Oriented Architectures, we called it ESPERANTO. In particular, we describe the requirements of such a middleware, as well as an innovative solution for the integration of discovery and of delivery facilities for nomadic users. In particular, we propose a novel service filtering strategy which can be used as design guideline for software practitioners who are going to solve similar problems.

The rest of this paper is organized as follows. Section 2 surveys existing middleware-based solutions for Nomadic Computing, and it casts some light on the role of SOAs in NCs. In Section 3 we define the interoperability requirements of nomadic systems. Section 4 outlines the overall architecture of the proposed solution. The Service Discovery and Service Delivery mechanisms are presented in Section 5 and Section 6 respectively. Section 7 concludes the paper and defines our future research directions.

2 Background and Related Research

2.1 Background

2.1.1 Traditional middleware and nomadic computing

Plenty of middleware infrastructures have been recently developed to provide several transparency mechanisms. Location, hardware platforms, operating systems, network protocols, programming languages, and migration transparency are only some examples of the transparency issues that have been addressed. These infrastructures mostly resulted in poorly interoperable solutions that adopt proprietary architectures and specific interaction semantics. Thus, traditional middleware approaches are unsuitable for developing nomadic computing applications. This is due to the following main reasons. First, the execution environment is too static: a host is connected statically to a network with a fixed location. Discovery services are static too and, often, not enough powerful: for instance, CORBA Naming and Trading Services enable to discover CORBA server objects by name, or by a short service description; these services provide applications with quite a static discovery infrastructure, which is not sufficient for mobility, dynamism, and heterogeneity. Moreover, communication paradigms are strongly coupled: remote invocations are implemented by means of synchronous and blocking primitives. This is due to the static nature of network infrastructures that applications are operating over. Finally, applications are insensitive to context: transparency might hide some crucial context-information. Guaranteeing transparency upon mobility is not straightforward; moreover, it is difficult to make an application adaptive without the support of the application itself [6].

2.1.2 Service Oriented Architectures and nomadic computing

The Service Oriented Approach has been adopted as de-facto standard for publishing and delivering services in modern distributed systems. Service-oriented applications are developed as a group of independent and interacting services, which provide users with well-defined interfaces. Service-oriented infrastructures assist applications by means of service discovery and delivery components. Service discovery is based on a set of protocols and mechanisms that enable software/hardware components to discover and look each other up. Service delivery protocols and mechanisms enable software/hardware components to connect to each other, so as to provide meaningful services.

2.2 Related Research

Although an effective middleware solution for nomadic computing has not been developed so far, several ongoing research works are targeted to address specific crucial issues. As far as context-awareness is concerned, several context-sensitive middlewares have been proposed during the last few years [2, 14, 13, 3]: most of them exploit computational reflection as a unique approach to achieve adaptation and reconfiguration. The work in [4] presents a service framework where a mobile device utilizes service sur-
rogates (resources available in the surroundings) for service advertisement, discovery, filtration, synthesis, and migration. These researches show that no standard models to design and develop middlewares for nomadic computing have been designed so far. For instance, most of the mentioned context-aware middlewares combine meta-data and reflection to trigger dynamic service reconfiguration, whereas each adopt proprietary architectures and context semantics [9].

3 Middleware Requirements

Interoperability in its various forms represents the main issue to address, even though a nomadic environment has also to cope with mobility support, and with application adaptation to context changes. Personal devices can change their point of connection (Access Point), upon movements, and can be turned off in order to reduce power consumption. Furthermore, applications can be sensitive to current user position (e.g., in order to guide him to the printer).

Thus, it is crucial that a novel middleware for Nomadic Computing fulfill the following requirements:

- **Guarantee the transparent interoperability of diverse middlewares and SOAs**: As heterogeneity is the trend in mobile computing, nomads should discover and use services irrespective of the particular technologies adopted in a certain domain. In other words, applications of a domain \( X \) must be able to discover and use services belonging to another domain \( Y \neq X \). External services must be transparently discovered and delivered.

- **Support device movements upon both discovery and delivery of services**

- **Support to context-awareness**: Nomadic applications must be sensitive to execution context, as they need to adapt their behavior to several contextual information concerning user, host device, on board sensors, network, and ambient [14]. This context-sensitivity concerns both the service discovery and the service delivery infrastructures.

4 ESPERANTO: Overall Architecture

The conceptual model of the proposed middleware infrastructure is depicted in Figure 1. The interoperability of domains is achieved by Mediators and Agents. Each domain is associated to a mediator component, and to an Agent, which can be either a Domain-specific Agent (DSA) or an Esperanto Agent (EA). Mediator components are in charge of handling connections between devices of different domains. Since applications operate under a broad range of network conditions, they need diverse interaction paradigms, as pointed out by emerging application scenarios [14, 7, 11]. The Mediator component has the following responsibilities: \( i \) to let interaction take place even in the case of temporary disconnections; and \( ii \) to provide one-to-many interaction. Mediators are connected each other via the core network. The DSA is responsible for the integration of discovery protocols. It implements the following two discovery algorithms: a proactive discovery, and a reactive discovery algorithm. The proactive algorithm has the following objectives: \( i \) to import service advertisements from different domains (such as Bluetooth, Jini, and UPnP), by translating/adapting the imported Service Description Record (SDR) into the domain-specific SDR; and \( ii \) to export service advertisements into external domains, by translating/adapting SDRs. Agents also act as local proxies during service delivery, as described in Section 6. Esperanto components reside mostly on the core network; only Esperanto Peers reside on the mobile devices that applications are running on.

As far as device mobility is concerned, both handover and disconnections need to be managed. Handover management requires Esperanto’s mobility support infrastructure to allow handover between different wireless technologies (Wi-Fi, Bluetooth). A device may use different Access Points (APs) to establish wireless connections to the fixed network; moreover, different APs can support different technologies. To address this handover issue, Esperanto provides applications with a low-level wireless communication infrastructure. The interested reader may refer to [5] for further details about our handover management strategy.
5 Service Discovery in Esperanto

The goal of ESPERANTO’s discovery infrastructure is twofold. First, it allows diverse discovery protocols to inter-operate; second, it provides a novel discovery protocol specially suited for nomadic domains.

5.1 Proactive integration of diverse discovery protocols

ESPERANTO integrates discovery protocols proactively: by pro-activity we hereafter refer to Domain Specific Agents’ capacity to dynamically import (and export) services into the domains of their own, before client’s requests. In order to export a certain service, a DSA follows a two-step process. Firstly, it converts SDR from a local representation (e.g., a Bluetooth Service Descriptor) to an ESPERANTO SDR (ESDR); subsequently, it sends the produced ESDR to the target DSAs (responsible for the domains the service has to be imported to), which perform the conversion from the common ESDR semantics to the ones of their own domains. ESDR is used as common language for SDR translations. It is composed of the following three main sections. The Basic Information section contains the service identifier, as well as a set of description attributes (e.g., service name, related key-words). The Service Profile section defines service semantics in terms of well-defined service classes (e.g., operating system, supported wireless technologies).

In order to decide whether a descriptor can be translated or not, we firstly formalize the concept of descriptor translatability, and then we define service visibility.

Preliminary Definitions: A generic SDR can be represented as a list of attributes $SDR = (a_1, a_2, ...a_n)$. Each attribute has a name, a value, and specific semantics (e.g. the name attribute may have a printService value, and “name of service” semantics). Fields can be either mandatory or optional. It is noteworthy that SDR of different domains may have different cardinality, and different mandatory fields as well. Let us consider $SDR_1 = (a_1, a_2, ...a_n) \in D_1$, and $SDR_2 = (b_1, b_2, ...b_m) \in D_2$, where $D_1$ and $D_2$ are two different domains.

Descriptor Translatability: a descriptor $SDR_1 \in D_1$ is translatable into a descriptor $SDR_2 \in D_2$, being $D_1 \neq D_2$, if the following conditions hold: i) all information representing the same semantics can be translated, and ii) the remaining mandatory fields of the target SDR can be artificially filled by the DSA.

Service Visibility: a service $s \in D_1$ is visible in domain $D_2$ if and only if $\exists SDR_2 \in D_2 | SDR_1$ is translatable in $SDR_2$. It is worth noting that if a service $s \in X$ is visible in domain $Y$, and its $Y$-version is visible in another domain $Z$, then $s$ is visible in $Z$ through $Y$ (transitive-visibility property). This property enables to integrate NCDs by using ESPERANTO domains as intermediaries for service advertisement and discovery.

Import operations can significantly increase the logical dimension of a certain domain. Hence, DSAs have to control the dimension of the domains of their own, due to discovery services’ resource limitations (e.g., the number of manageable service records). Visibility rules provide a first step toward the management of domain complexity, even though they do not guarantee that domain resources be always available. Moreover, new import operations can be discarded upon resources unavailability: this can make entire service classes unavailable. Therefore, ESPERANTO adopts a more complex filtering strategy, in order to guarantee a minimum number of service records for each service class. Let $D_{visible}(D_1)$ be the maximum dimension of domain $D_1$, and $C(D_1) = C_j (j = 1...n)$ be the set of service classes provided by $D_1$. Based on these definitions we can further define the following parameters: $D_{residual}(D_1)$ is the residual dimension of $D_1$, $minspace(C_j)$—minimum space required by class $C_j$. If all the classes have their own $minspace$ available, the number of $C_j$ services can exceed $minspace(C_j)$. Thus, we identify Exceeding (Ex) and Not Exceeding (NotEx) classes, depending on the current number of services; we hereafter call $N(D_1)$ the overall number of service classes available in $D_1$, $N_{Ex}(D_1)$ and $N_{NotEx}(D_1)$ the number of exceeding and not-exceeding classes respectively. $Dim_{Ex}(D_1)$ and $Dim_{NotEx}(D_1)$ represent the fraction of $D_1$ assigned to exceeding and not-exceeding classes respectively.

Given a service $s_i$, the filtering rules of the proposed proactive algorithm are defined as follows:

- if $C(s_i)$ is not exceeding, $s_i$ is imported into $LD_1$;
- if $C(s_i)$ is exceeding, $s_i$ is imported only if all the classes have their own $minspace$ available, or equivalently: $D_{residual}(S_1) \geq N_{NotEx}(D_1) \cdot minSpace - Dim_{NotEx}(D_1)$, being $N_{NotEx}(D_1) \cdot minSpace - Dim_{NotEx}(D_1)$ the space available for not-exceeding classes;
- if importing $s_i$ results in $D_{residual}(D_1) < N_{NotEx}(D_1) \cdot minspace - Dim_{NotEx}(D_1)$, $s_i$ is imported if and only if the following conditions apply: a) $s_i$ is usable, and b) the service $s_d$ to discard is either not usable or usable with an usage rate less than $minusage(s_d)$.

Filtering rules are depicted in Figure 2. As for the last rule, usability is the possibility to build service proxies on both client-side and server-side, as defined in Section 6.
versely, i.e., a not-optimal service usage is being performed. Con-
mechanism is too fast with respect to client requests).

reaching a significant usage rate (i.e., the import/export
very frequently, and/or

MaxServices

represented by filtering parameters, i.e., service class an d
service usage;

lies on the following assumptions:

achieving import mechanism stability.

tively used, and service discarding is quite a rare event, th us

estimated through the

import ratio

, defined as follows:

As an example, \( t_i \approx 1 \) means that i) remote domains
behave irregularly, for they announce and discard services
very frequently, and/or ii) services are exported before
reaching a significant usage rate (i.e., the import/export
mechanism is too fast with respect to client requests).
Hence, an unstable mechanism is characterized by \( t_i \approx 1 \),
i.e., a not-optimal service usage is being performed.
Conversely, \( t_i \approx k, 0 < k < 1 \) means that services are effec-
tively used, and service discarding is quite a rare event, thus
achieving import mechanism stability.

We evaluate the stability of the discovery mechanism
through a MATLAB simulation model. This model re-
lies on the following assumptions: a1) each service is
represented by filtering parameters, i.e., service class and
service usage; a2) all the services to import are visible; a3) all the services to import are usable; and a4) remo te domains behavior is regular. We define two kinds of
model parameters, namely internal parameters, which
directly characterize the filtering mechanism, and extern al parameters, which significantly affect the mechanism,
even though they do not belong to it. External paramet ers include the registry dimension, \( \text{Dim} \), the overall num-
ber of classes, \( \text{ClassNo} \), the maximum allocation vec-

tor, \( \text{MaxServices}(i) \) \( (\text{MaxServices}(i)) = n \) means that
each import operation should import at most \( n \) services
of class \( i \), and the service-usage vector, \( \text{UsageRate}(i) \)
\( (\text{UsageRate}(i)) = t_{av} \) means that \( t_{av} \) is the average usage-
time of class-\( i \) services). Internal parameters encompass
the minimum space assigned to each service class, \( \text{MinSpace} \),
the minimum usage threshold, \( \text{MinUsage} \), the frequency of updates to the service-usage informa tion, \( \text{UsageResetFreq} \), and the type of update to perform,
\( \text{UsageResetType} \), i.e., reset usage or divide current usage
by the specified value. Given the inter-filtering time \( T \) and
the quantity of announced services \( Q \), we define the average
volume of services as \( V = T \cdot Q \). However, \( V \) can also be
calculated as follows:

\[
V = \sum_{i=1}^{\text{ClassNo}} \text{MaxServices}(i)
\]

as \( \text{MaxServices}(i) \) elements are uniformly distributed.
Hence, we can vary \( V \) by tuning \( \text{MaxServices} \)’s compo nents. The stability of the import mechanism has been evaluated with respect to the inter-filtering time \( T \). For the sake of simplicity, we assume to import one service each second (i.e., \( Q=1 \)). Figure 3a) shows that, when \( T = 30 \)
or equivalently when \( V = TQ = 30 \), the steady-state im-
port ratio is close to \( 30\% \); if \( T \) is halved, the import mecha
nism gets unstable; this instability is represented by high
import-ratio values, which reach \( 80\% - 90\% \). Hence, Figure
3b) shows how to tune internal parameters so as to
to enhance system stability upon low inter-filtering times;
the reference import-ratio represents an unstable mecha
nism, characterized by \( \text{MinUsage} = 3, \text{UseResetFreq} = 20, \text{UseResetType} = 0 \).
By forcing \( \text{MinUsage} = 2 \), the mechanism behaves more selectively, since services that had been used at least 2 times (upon the last registry re-set operation) cannot be discarded; therefore, the import ra
tio value is lower than the reference one, i.e., the mecha
nism is more stable. We can increase stability also by vary-
ing the characteristics of the registry-reset operations.
Increasing \( \text{UseResetFreq} \) up to 40 can specifically lead to a
more stable mechanism. Initial variations of the import
ratio can be smoothed by decreasing the \( \text{MinUsage} \) par a
ter. The last curve represents the effects of changes in the
\( \text{UseResetType} \) parameter: by dividing usage information
by 4 at each reset operation \( (\text{UseResetType} = 4) \), we
have a more regular curve that asymptotically reaches a
zero value. Hence, in this case the higher the number of
requested operations, the more selective the mechanism.

5.2 Reactive discovery within ESPERANTO do-

mains

Esperanto Agents guarantee discovery protocol reactive-
ness. They explore external Domains upon application’s re-
quests, i.e., they do not perform import operations. How-
ever, in order to publish Esperanto services in external heterogeneous domains, they export such services to non-
Esperanto domains. The characteristics of native Esperanto service discovery protocol are briefly summarized in the fol-
lowing.

- **Service Registry**: each Esperanto domain has its own
  service registry, called Esperanto Explorer (EE), which
  contains local Esperanto services ESDRs, as depicted in Figure 1;

- **Service Advertisement**: each Esperanto Peer (EP) pub-
  lishes the ESDRs of its own through service advertis-
  istments onto the local EE;

- **Service Template**: each EP can discover services by
  sending a specific request to the local EE; this request
  contains a particular ESDR describing the expected
  service, called service template;

- **Protocol reactivity**: two main discovery operations
  are permitted, namely intra-domain discovery (i.e.,
  discovery within a domain), and inter-domain discov-
  ery (i.e., discovery outside the local domain); inter-
  domain discovery requests are delivered to the local
  EA, which is in charge of discovering services among
  different domains.

As for mobility support, the ESDRs of services pub-
lished on the moving device have to be sent from the old-
domain’s EE to the new one. This solution allows de-
vices to exploit the EE to advertise and remove services,
and for intra-domain discovery as well. As for disconnec-
tions, it is crucial that the consistency of service availabil-
ity information be preserved upon disconnections of devices
that provide registered services. Indeed, asynchronous ser-
vices can be discovered despite disconnections of their own
providers, whereas discovering the available synchronous
services upon providers’ disconnections is a challenging
issue. We thus adopted a lease-based strategy. Service reg-
istrations are leased during the advertisement phase; ser-
vices are in charge of subsequently renewing the leases of
their own. This mechanism allows EE to erase the services
whose lease is expired, i.e., the services delivered by dis-
connected or permanently unavailable devices.

As far as context-awareness is concerned, context infor-
mation enables to deliver the right services to the right users
at the right time [12]. Indeed, in traditional discovery proto-
cols, SDRs provide quite a static representation of context-
information; for instance, a "service load" attribute can ex-
press the average service load as calculated at advertisement
time. This static and a-priori description forces users to look
services up among all the available services, until they find
a suitable service. A context-aware service discovery in-
frastucture can filter and present services on behalf of end-
users; therefore, we enriched the ESDR by a set of context
fields (bandwidth, location, load, and availability). Context
attributes are updated by both device-side and core-side Es-
peranto infrastructures.

6 **Service Delivery**

The ESPERANTO delivery infrastructure pursuits the
following objectives:

- **Mobility Support.** The infrastructure has to provide
  i) facilities to let devices communicate within a mo-
  bile environment, in spite of device movements and/or
disconnections; and ii) interaction primitives, so as to
  keep interactions between clients and services sepa-
rate.

- **Context Awareness.** The delivery infrastructure has to
  provide applications with context awareness mechani-
sms. Such mechanisms consist of context descrip-
tions (i.e., the description of contexts, and context
changes on which applications are interested to react) and
adaptation mechanisms (i.e., how the applications
have to adapt their behavior to context changes).

- **To guarantee the interoperability of heterogeneous de-
  livery infrastructures.** For instance, a Jini client has to

![Figure 3. Effects of filtering parameters on the import ratio](image-url)
Figure 4. ESPERANTO context awareness support

(discover and) interact with a Bluetooth service. This is achieved by means of bridging components, which are transparent to clients and services.

In the following, we briefly detail how we addressed the mentioned issues within the ESPERANTO delivery infrastructure.

6.1 Mobility Support

The ESPERANTO delivery infrastructure relies on an interaction paradigm that takes into account mobility management. This infrastructure was inspired by Linda generative communication model; this model allows senders and receivers to not be contemporaneously available, and it makes data exchanges not sensitive to location [8]. Tuple space model supports disconnections and reconnections by decoupling components from each other. We implemented the tuple space model as composition of several sub-spaces, each assigned to a certain domain. Mediator components are in charge i) of providing access to the sub-space associated to the domain of their own, and ii) of allowing applications to access the whole tuple-space, despite device movements, and independently of their current domain. Mediators offer non-blocking primitives to write, read, and take tuples. Similarly to the JavaSpaces service, tuples are managed through a lease mechanism, which makes the space more resilient upon device disconnections.

6.2 Context Awareness

Esperanto addresses dynamic context changes through a specific context-aware adaptation strategy. By context resource, we mean the set of resources (externals and internals) that can influence the behavior of ESPERANTO peers. The adopted context-aware strategy consists of the following activities: i) retrieving and monitoring context resources state; and ii) applying adaptation rules (i.e., how a peer reacts to context changes). ESPERANTO’s support to context-awareness is depicted in Figure 4. In particular, the following two approaches have been adopted: i) peer-driven, i.e. peers monitor resources via the ResourceMonitor class, which presents the low level information concerning resource state, in order to adapt their behavior to context changes; and ii) event-driven, i.e. peers receive context-change events from the ResourceManager class, which is responsible for notifying context changes to the interested subscribers. By using such an event driven strategy, the peer joins the ResourceManager (via the registry method), providing it with an event descriptor which describes the state transition that triggers its adaptation. Adaptation rules are put into a special handler, which modifies application behavior according to the received events.

Figure 5. Service advertisement - exporting phase

6.3 Interoperability

Domain Specific Agents, residing on domains’ edges, enable the interoperability of heterogeneous domains. The role of such agents is twofold: i) upon each registration of a new service, the DSA has to provide a service-side proxy, and to notify service advertisements to all the interested external domains (see Figure 5); ii) upon receiving each service advertisement, the DSA has to create a client-side proxy (see Figure 6). It is worth mentioning that native ESPERANTO devices do not need service proxies to be created. To create service proxies, the Domain Specific Agent produces an XML-based ESPERANTO SADR for each published service within the domain of its own. Information included into the SADR encompass message formats,
Moreover, the selectivity of the filtering mechanism is the lower the inter-filtering time is, the higher the import ratio is. System stability is proportional to the inter-filtering time, evaluated by simulations. Simulation results showed that the effect of several filtering parameters on the import ratio (i.e., on mechanism stability) have been consistent with the Web-Services architecture, which deals with services as collection of functionality, each characterized by input/output messages, and interaction paradigms as well. Client-side proxies can connect to services and use them through service-side proxies; these interactions are based on the tuple-space infrastructure. Interoperability of heterogeneous SOAs is thus achieved by exploiting ESPERANTO’s Mediators.

7 Conclusions

This work presented a middleware platform, named ESPERANTO, to achieve interoperability in nomadic computing infrastructures. The proposed middleware supports integration of diverse Nomadic Computing Domains, from the service discovery and service-delivery viewpoints. The ESPERANTO delivery infrastructure has been designed to provide applications with mobility support, context awareness, and decoupled interaction mechanisms. The ESPERANTO discovery infrastructure has been designed to allow interoperability of existing discovery protocols, and to provide a novel discovery protocol especially tailored to nomadic computing constraints. We achieved interoperability through the implementation of proactive integration of individual discovery protocols, which consists of dynamically importing (and exporting) services among heterogeneous domains. The effect of several filtering parameters on the import ratio (i.e., on mechanism stability) have been evaluated by simulations. Simulation results showed that system stability is proportional to the inter-filtering time: the lower the inter-filtering time is, the higher the import ratio. Moreover, the selectivity of the filtering mechanism is proportional to the number of requested operations.

Our future research will focus on the evaluation and validation of the proposed strategy through massive experiments on the system prototype, including Jini and Bluetooth domains.

References