Experiences Building a Service Execution Node for Distributed IN Systems

Menelaos K. Perdikeas, Fotis G. Chatzipapadopoulos, and Iakovos S. Venieris

National Technical University of Athens, 9 Heroon Polytechniou, 157 73 Athens, Greece
[perdikea|fhatz]@telecom.ntua.gr, ivenieri@cc.ece.ntua.gr

Abstract. We describe a Distributed Intelligent Network architecture. By laying a distributed processing environment a more flexible mapping between functional and physical entities is afforded. Mobile code can in turn be employed to cater for performance or load balancing considerations as well as to increase the overall flexibility and manageability of the system. Changes in the physical locations of the endpoints of a control relationship can be abstracted to higher layer software by virtue of the location transparency properties of the environment. We report on the development of an experimental implementation and we demonstrate how the aforesaid technologies improved its characteristics.

1 Introduction

The Intelligent Network (IN) approach for providing advanced services to end users aims primarily at minimizing changes in network nodes by locating all service-related (and thus likely to change) functionality in dedicated IN-servers, known as 'Service Control Points' [1,2]. These servers are in a sense external to the core network which in this way needs to comprise only a more or less rudimentary switching functionality (Service Switching Points) and the ability to recognize IN call requests and route them to the specialized service control points. The advent of Mobile Agent and Distributed Object Technologies (MAT and DOT) can help unravel the full IN potential. The IN, being a distributed infrastructure has much to gain from these technologies in terms of flexibility, reduced times to develop, test and deploy new services and an overall architecture that is easier to manage, maintain and extend.

The ACTS MARINE project [3] had the objective to investigate enhancements of IN towards a distributed computational environment, which will model the interactions between IN elements as distributed method invocations and will allow the exchange of Service Logic Programs (SLPs) implemented as mobile agents. IN elements in this context refer to those functional entities that correspond to the endpoints of traditional IN flows as they are described in [4].

For comprehensive descriptions of DOT and MAT see [5,6]. This paper is structured as follows. We first introduce the concept of Distributed IN and provide its rationale. We then describe the architecture of a prototypical implementation. Having employed extensively DOT and MAT in this context we present our experiences and conclude with some thoughts on the applicability of these technologies in this and similar contexts.

C. Linnhoff-Popien and H.-G. Hegering (Eds.): USM 2000, LNCS 1890, pp. 310-317, 2000.

2 What is Distributed IN?

In a traditional IN implementation, the Intelligent Network Application Protocol (INAP) information flows are conveyed by means of static peer-to-peer protocols executed at each functional entity. The static nature of the functional entities and of the protocols they employ means that in turn the associations between them are topologically fixed. An IN architecture as defined by [4] is inherently centralised with a small set of service control points and a larger set of service switching points engaged with it in INAP dialogues. The service control points are usually the bottleneck of the entire architecture and their processing capacity and uptime in large extent determine the number of IN calls the entire architecture can handle effectively.

Distributed object technologies can help alleviate that problem by making associations between functional entities less rigid. This is a by-product of the location transparencies that use of DOT introduces in any context.

Once a distributed processing environment has been in place, MAT can in turn be employed to take advantage of object mobility. In MARINE, SLPs are implemented as agents able to migrate to the switch and control its operations locally. This can be justified on performance or load balancing grounds. For instance, having the SLP interact locally with the switch avoids the larger remote communication costs and economizes on network bandwidth. It also relieves a possibly overloaded service control point.

3 The Architecture at Large

Based on the considerations expressed in the previous section, we are augmenting the physical entities of traditional IN with distributed processing and code mobility capabilities. In particular at each computing node that hosts a functional entity, a CORBA ORB and a mobile agent platform are installed. The introduction of CORBA allows a laxer mapping between functional and physical entities, which can in any case change very easily so this mapping is not an intrinsic property of the architecture. In effect, we define a CORBA interface for each endpoint of an INAP information flow and implemented the appropriate CORBA objects for servicing these flows. So, both the Service Control Function (SCF) and the Service Switching Function (SSF) are CORBA objects, peers with each other. Java was used for the implementation of the SCF since the later relied on MAT.

The major physical entities of the system depicted in Figure 1 are:

1. The Service Execution Node (SEN). A workstation hosting – among others – the SCF. A Java-based mobile agent platform (Grasshopper [7]) and a CORBA ORB are installed at the SEN. The SCF agency provides the functionality for SLPs to be introduced, initiated, executed, suspended or withdrawn. The term 'agency' refers to the components that must be installed in a computer node and that provide the necessary runtime environment for mobile agents to execute. SLPs are implemented as mobile agents to take advantage of the mobility characteristics in ways that are described later on in this section. Naturally, not all components of the SCF need be mobile agents. A lot of infrastructure components are stationary agents or simply Java objects having no relationship with MAT except for the fact that they share the same process space. The SCF agency uses the ORB installed at the SEN and so it appears connected on to the CORBA bus of the system. SLP agents in the SEN can migrate to the switch using the agent bus (which is also used when SLPs are initially deployed from a management workstation).
2. The Service Switching and Control Point (SSCP). It hosts a SCF identical with the one at the SEN which caters for SLPs that have migrated locally to the switch. It also hosts the SSF which is for the most part a typical SSF implementation. It is usual for SSFs to be implemented in C++ to respond to stringent performance requirements. Having mobile SLPs implemented in Java communicate with static, C++ code would be a very cumbersome situation if it weren't for CORBA. CORBA's implementation transparency means that language of choice is not important and its location transparency implies that the SSF can behave the same way whether the SLP it is engaged with is locally or remotely located (i.e. located at the SEN's or the SSCP's SCF). Two workstations are shown attached to the switch using bearer signalling. A Broadband Videotelephony and an Interactive Multimedia Retrieval application were installed on each terminal and were used for testing.
3. Finally, there is also the Service Management System node that is used for management, supervision and bootstrapping of the system. The Management System contains the agent repository which hosts a number of service agents (e.g. Interactive Multimedia Retrieval service) that are at bootstrap or reconfiguration time deployed in various agencies of the system. Service agents are originally deployed to the SEN's SCF. The CORBA Naming Service process also appears running in the Management System node but clearly it can run on any other CORBA enabled workstation. The same applies to the Region Registry process which also appears running at that node. The Registry is a Naming Service-like service that is for use within a agent environment and is used in connection with the agent bus.

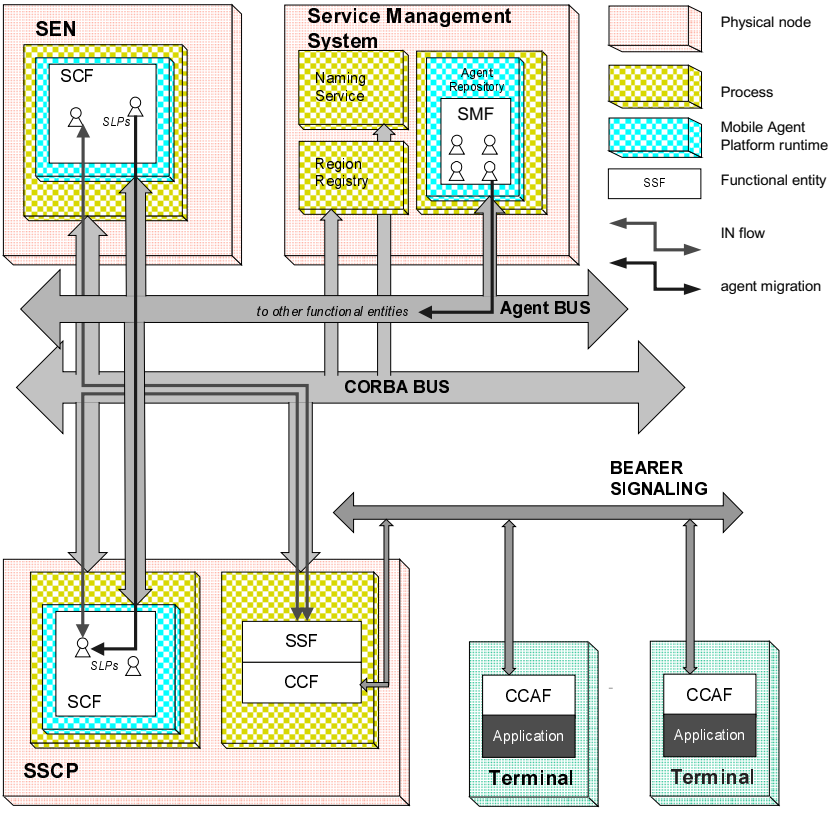

Fig. 1. Reference architecture

With the reference architecture of Figure 1 in mind we can see how automatic reconfiguration of the network's IN flows can take place. Refer to Figure 2 for numbered steps.

Originally all SLPs reside on the SCF located in the SEN and so service provisioning (which involves SLP-switch interaction) occurs between two remotely located physical entities (like in traditional IN). We depict two SSCPs receiving instructions from the same SEN. This corresponds to frame (a) in Figure 2. However, when the load balancing mechanisms of the system detect that a certain service should be served locally (due to either link capacity exceeded or to computational overloading on a node), it uses that deployed service to clone it and create a second prototype. That cloned prototype is then migrated locally to one SSCP (frame (b)) and is used for subsequent sessions to provide services locally. The other SSCP whose load balancing mechanisms presumably did not detect the need for a locally available prototype continues to be served remotely (frame (c)). While the originally deployed prototypes are administered by the Management System, second prototypes' existence is transparent to it. The latter are created and removed by the switch-located SCFs when certain criteria are met. So, at a later point in time we may arrive in a situation where the second SSCP is served locally (i.e. from the SSCP located SCF) and the first one has reverted to remote mode.

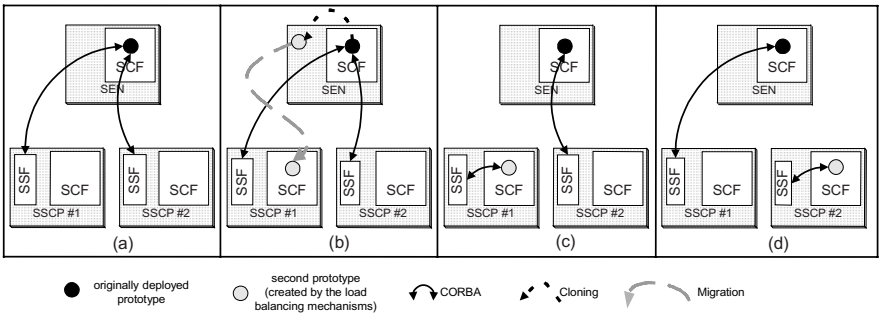

Fig. 2. Automatic reconfiguration of the IN flows of the network

4 Experiences Realising Distributed IN Systems

Reflecting on the design decisions we took while implementing the architecture we described, we can consider as a starting point a typical IN implementation. The first move towards a Distributed IN was the replacement of the SS7 stack with CORBA. This meant that the same IN flows were implemented using CORBA method invocations. The were a number of decisions that had to be taken at this step: the most important was whether the method invocations would be synchronous or asynchronous (blocking or non-blocking).

4.1 Invocation Mechanisms and Threading Issues

Non-blocking method invocations resemble closer the semantics of message passing protocols and thus probably require fewer modifications to existing software. Asynchronous method invocations or message passing mechanisms can be contained within wrapper entities to appear the same to higher layer software. Moreover, since the IN protocol has been defined with message passing mechanisms in mind, asynchronous invocations fit more naturally from a semantic point of view. This is why fewer modifications to existing software are needed after all.

Blocking method invocations are quite different than exchanging messages but they are more natural to use in the context of a distributed processing environment. This is different than saying that they more appropriate with a specific protocol (e.g. INAP) in mind. It is just that, if a distributed processing environment is to provide the illusion of an homogeneous address space, where method invocation are made just as the objects were residing in the same process context, then blocking method calls enhance this view. From a programmatic point of view, the main effect of synchronous method invocations is that they reduce need for threading at sending entities and that the developers don't have to implement synchronisation themselves (because the method calls themselves block until a result can be returned). On the other hand, depending on the architecture and the network of connections between invoking and invoked objects, it may lead to deadlock problems at receiving or intermediate entities where a number of such method invocation chains may pass through.

4.2 Establishing Associations between Objects

Concerning the use of a DOT, another issue we found to be important was the way in which the various CORBA objects of the system (corresponding to the functional entities) would obtain references to each other. Two at least approaches are possible.

One is the use of a simple look-up mechanism where the entities of the system search their peers by means of their names and retrieve an IOR they can then use to directly invoke methods. The Naming Service provides an elegant, standardized and scalable (with federation of Naming Services) solution. Support of nested naming contexts guards against name pollution. Reconfiguring associations is thus readily supported by simply changing the relevant names.

A more structured solution would be to implement a 'configuration server' that would be queried by all CORBA enabled entities of the system. The syntactic form of these queries would be left to the designers of the network to define but it would allow a higher level approach to be implemented, one that would more clearly reflect the idiosyncrasies of the architecture and the roles involved. This server would store in a centralised manner all configuration information about the system facilitating easy monitoring and modifications of these settings.

4.3 Configuring the System

In the MARINE architecture we chose to implemented the Naming Service solution coupled with the use of configuration agents. These agents employ code mobility and are dispatched from a central location to the nodes of the system. One of their uses is to carry configuration settings (e.g. Naming Service names). That reconfigurations should be applicable at all points during the operation of the system and not only at bootstrap time provided the rationale for the use of configuration agents.

In a complex distributed system, one involving many associations between its entities, during bootstrap or reconfiguration time objects can be found to be in invalid states. Catering for such not properly initialized objects can degenerate to a series of ad hoc remedies made on a peer-to-peer basis (e.g. between pairs of communicating objects) and not universally for the entire system. Given that these periods of inconsistent state can be extended (for the time-scales of a telecommunication system) and will occur not only during bootstrap but also during re-configuration or following an erroneous condition, it is easy to lose track of the overall state. A server that centrally stores and administers all configuration information about the system can provide a solution to this problem.

4.4 Applying MAT

Having relied on DOT to provide connectivity between IN functional entities, the next step was to identify ways in which the introduced location transparencies could result in more flexible implementations or even cater for shortcomings of the original centralised approach. Naturally, we considered MAT for this purpose. This presupposed use of Java and installation of special middleware. Since connections between objects were no longer statically defined nor associated with transport protocol addresses, code mobility could be employed to allow a software object to

move from one node to another without terminating its associations. Performance optimisations as well as load balancing considerations could justify such a migration.

It is not the objective of this paper to make a general assessment of MAT but we have found that certain conditions hold which warrant the use of mobile code capabilities in the case of distributed IN. First of all, given the intensity of the IN dialogues held for instance between SCF and SSF, performance gains can indeed be expected. Secondly, each SLP is individual in terms of the finite state machines it maintains, the INAP methods it invokes, their parameters, their sequencing. The abstraction of an agent as an autonomous software object thus fits neatly with the concept of a service logic program. We do not believe that alternate ways to transfer control from SCF to SSF are not possible but we hold that the MAT paradigm blends naturally with the abstractions and models of IN and provides a clean cut and intuitively appealing way of doing things. Also, any solution resting on MAT is more dynamic and flexible than changing the logic of the protocols which can be very cumbersome or even unfeasible. Finally, management's ability to reconfigure the network remotely, without interrupting currently executed services makes the case for MAT even stronger. This can for instance be implemented using agents to be dispatched from a management location to a malfunctioning node to locally interact with it in case remote interfaces have not been defined.

With the particular architecture of MARINE in mind, we have found that the ability for an SLP to migrate to the switch and control its operations locally is a very efficient way to respond to an observed pattern of service requests, a congestion or an anticipated failure or maintenance shut down. We also found the mechanism of cloning existing prototypes very helpful all the more so since it was supported directly by the platform we used. We also employed code mobility to dispatch specialised software components to deal with problems or reconfigurations at remote nodes.

5 Conclusions

We have found that CORBA can very reliably substitute message-based protocols like INAP while incurring only mild performance penalties. Furthermore, there is nothing inherent in the paradigm of CORBA that prevents it for operating in even faster ranges, except perhaps its complexity and the larger number of intermediate layers. Those layers comprise for instance facilities for marshalling and unmarshalling that are absent in more rudimentary transport layer protocols. We believe that this is a profitable trade-off and that CORBA's advantages in wrapping legacy code and enhancing code modifiability (along with making it much more compact) for the most cases compensate handsomely. Moreover, development of specialized, real-time ORBs will probably remove performance-premised objections. Furthermore, and while in out prototype we used a private IP network to interconnect the IN physical elements the consortium behind the CORBA specifications has released a framework that enables conveyance of CORBA over a native SS7 protocol stack.

Concerning MAT, the performance of all presently available platforms would have been debilitating for commercial, real-time applications. Two factors contribute to this poor performance: use of Java and the fact that mobile agents in contrast to CORBA objects operate as threads in the context of a much larger and performance bogging entity (the platform providing the runtime environment). It is difficult to

make an in-depth analysis comparing the performance of our prototype to that of contemporaneous architectures. This is due to the fact that the benefits accruing from the dynamic distribution characteristics of our approach would be more evidenced in large scale installations. However, an educated guess would probably hold the overall performance of our architecture to be inferior to that of commercial deployments. Nevertheless, and while the focus of our approach is not solely on performance, we believe that there is a clear potential for improvement as this is not due to structural reasons but has to be attributed to the present state of the technologies we used.

Performance or load balancing reasons by themselves should not necessarily lead to the introduction of MAT. However, when the potential for a more efficient environment coexists with the ability to adopt a network design that is more natural for a given context (e.g. agents to implement autonomous SLPs) then MAT is a viable alternative.

References

1. Magedanz, T. and Popescu-Zeletin R. (1996) Intelligent Networks – Basic Technology, Standards and Evolution, International Thomson Computer Press, ISBN: 1-85032-293-7, London, June.
2. Venieris, I.S. and Hussmann H. (eds.) (1998) Intelligent Broadband Networks, John Wiley, ISBN: 0-471-98094-3, Chichester
3. MARINE Project (ACTS AC340), http://www.telecom.ntua.gr/~marine/marine.htm
4. ITU-T Recommendations – Intelligent Network, Series Q.12xx
5. OMG, '*Common Object Request Broker Architecture and Specification*', Updated Revision 2.1, November 1997.
6. Menelaos Perdikeas et al., "Mobile agent standards and available platforms", Computer Networks and ISDN systems, Elsevier, Vol 31, pp. 1999-2016
7. http://www.ikv.de/products/grasshopper/index.html