Messaging Infrastructure for Electronic Government: Background, Rationale, Objectives

Elsa Estevez [1]
ece@cs.uns.edu.ar

Tomasz Janowski [2]
tj@iist.unu.edu

Pablo Fillottrani[1]
prf@cs.uns.edu.ar

[1] Department of Computer Science and Engineering, Universidad Nacional del Sur
Bahía Blanca, Argentina
[2]United Nations University International Institute for Software Technology
P. O. Box 3058, Macau

ABSTRACT

Electronic Government applications aim to deliver efficient, seamless, customer-focused services to the public using various distribution channels. One of challenges in building such applications is the requirement to rely upon pre-existing, often heterogeneous government applications. In order to enable communication between such applications, middleware software is required. In this paper we present a research line for formal specification and development of programmable message-oriented middleware, a foundation for infrastructure development for Electronic Government.

Keywords: Software Infrastructure, Message-Oriented Middleware, Electronic Government.

1. INTRODUCTION

In the past, most enterprises in both private and public sectors were focused on the development and use of custom-made software, paying little attention to integration issues. Recently, enterprises are faced with the need to explicitly address the software integration problem. Particularly in the public sector, this problem is widespread due to the accumulation of heterogeneous solutions running on different execution platforms and developed in different programming languages, used to deliver long-lasting public services. In addition, when most government applications are designed to respond to the needs of individual agencies, integration is required to deliver services that seamlessly cross organizational boundaries.

Addressing the integration challenge raises a number of interoperability issues. The basic problem is for the applications running on different execution platforms to be able to exchange messages. This is called technical interoperability. Additionally, semantic interoperability is required, since applications must share the understanding of all messages exchanged between them, which represents a more complex problem. Whatever interoperability level is used, communicating applications should not depend on one another in terms of how and when the messages are produced and consumed internally. Finally, when considering the dependency level between applications, loose coupling is preferred above tight coupling.

There is a significant body of work on solving the above problems. Most approaches rely on middleware [1] which is the level of software providing specific services to user applications [2]. Middleware hides the complexity of communication between these two layers, and provides the functionality needed to bridge the gap between application programs and hardware and software infrastructure [3]. The main objectives of middleware are: (1) to make the development and evolution of distributed systems feasible, easier and cheaper, (2) to divide software into components and coordinate how such components can interoperate, and (3) to enable the integration of components developed for different execution and communication platforms and written in different programming languages.

Our research focuses on the development of a formal model for asynchronous messaging in a Message-Oriented Middleware as a solution to the integration problem for distributed and heterogeneous applications in general and for public sector (Electronic Government) applications in particular. In addition to addressing the interoperability problem, such middleware can provide functions of general interest and reuse by communicating applications.

The rest of the paper is as follows. Section 2 presents the background related to middleware and communication. Section 3 outlines the need for a messaging middleware to support Electronic Government. Section 4 describes the objectives for our research. Section 5 presents related work. Finally, Section 6 draws some conclusions.

2. MIDDLEWARE AND COMMUNICATION

Many middleware solutions have been developed by both software industry and the open source community. Here are four well-known examples, in chronological order [1,2]:

1) *Transaction Processing Monitors* (TPM): An atomic transaction is a set of semantically related operations, which must be processed completely or aborted. TPMs provide an implementation of atomic transactions. Initially, TPMs were designed to support the sharing of resources among multiple concurrent users of a mainframe. Nowadays, they must also deal with multithreading and data consistency.
2) *Remote Procedure Calls* (RPC): An RPC allows the execution of an operation on a remote system by calling a local routine. RPCs are synchronous, providing the benefits of immediate response. Technologies implementing RPCs include: Common Object Request Broker Architecture (CORBA) [4], Remote Method Invocation (RMI) [5], DCOM [6], Sun-RPC [7] and Simple Object Access Protocol (SOAP) [8]. As the main drawback, if the remote system is unavailable, the calling application must somehow be notified about this fact. Therefore, error

handling and recovery logic must be implemented in every application using an RPC-style communication.

3) *Message-Oriented Middleware* (MOM): A message-oriented middleware facilitates the asynchronous exchange of messages between applications. MOM-based technologies are based on a few simple core concepts: a producer application creates and sends messages, consumer applications receive and consume these messages, applications use a messaging client API to communicate with one another through a messaging system, the messaging system maintains message queues for message transfer. Producers and consumers are loosely coupled, communicating through virtual channels managed by the middleware. MOM has been implemented in several products: IBM WebSphere MQ [9] Microsoft Message Queuing Services (MSMQ) [10] WebMethods Enterprise [11] and the messaging services of CORBA.

4) *Object Request Brokers* (ORB): An ORB allows an application to request a service without knowledge about what servers exist to provide this service. The request is sent to a service broker and the response is returned to the requesting application. Several ORB implementations exist, most implement the Common Object Request Broker Architecture (CORBA). Although CORBA provides a great solution for interoperability, it was not widely adopted by the industry particularly because of the lack of industry standards and requirements for the format of messages.

For all types of middleware above there are various ways to categorize the communication mechanism. The basic discriminating factor is whether the sending of a message by a producer and its receipt by a consumer must take place at the same time. There are two possible options:

- *synchronous*: the producer sends a message and waits for a response. Middleware relying on synchronous communications include RPCs, DCOM and CORBA. Unfortunately, synchronous communication leads to tightly coupled applications - each must know the details of how the other wishes to communicate.
- *asynchronous*: the producer sends a message and forgets about it. Asynchronous interactions provide a key design pattern for building loosely coupled applications. MOM is the prime example of middleware relying on asynchronous communication.

Recently, it has been recognized that loosely coupled asynchronous processing is more appropriate for describing interfaces and for interactions between applications and services [1][3]. For instance, while SOAP version 1.2 supports RPC-style invocations, it does not recommend RPC as the basic interaction mechanism. JAX-RPC (Java API for XML-RPC) adopts asynchronous processing for its version 2.0. In addition, languages used for modelling business processes based on web-services, such as WS-BPEL [12] and WS-Choreography [13] are entirely based on asynchronous exchange of data.

3. ELECTRONIC GOVERNMENT NEEDS

Many governments worldwide are making great efforts to advance the state of Electronic Governance. Initially, these efforts were concentrated on the automation of public services, with basic web presence and automatic delivery of simple public services taking priority. More recently, emphasis on service quality and efficiency resulted in more complex requirements: one-stop government, front–office and back-office application integration, delivering services offered seamlessly by several agencies, and others [14].

In turn, these additional requirements represent recurrent problems for which general solutions should be provided as part of the infrastructure for Electronic Government. Many of them could in fact be addressed by a programmable messaging middleware, for instance:

1) Government applications require high maintainability. Frequently changing laws, regulations and procedures cause routine modifications to government systems. Such modifications are not only expensive, perhaps involving a major re-engineering effort, but also risky. A programmable messaging middleware could address this by connecting process-dependent components that implement specific administrative and business rules with legacy systems. With such a middleware in place, changing an administrative procedure may cause only a minor change to one of the middleware component without affecting the legacy components.
2) Maintaining government systems is highly expensive. Such systems follow strict laws and regulations, and constitute key assets for public organizations, very difficult and expensive to replace. Integrating such systems with new technologies is difficult. A programmable middleware enabling communication among heterogeneous applications is a relevant solution, since it can integrate legacy systems by exposing and making available their functionality.
3) Delivering seamless, customer-oriented services [15] requires crossing organizational boundaries between agencies and thinking in terms of customer needs. An important challenge is integration of heterogeneous applications running in several agencies. Again, middleware infrastructure to handle asynchronous communication between different government organizations is a relevant solution to this challenge.

Typical functionality offered by MOMs is rarely sufficient to address all application needs. Besides enabling a simple exchange of messages, messaging applications are required to perform additional functions, such as: validate messages against a certain pre-defined format, transform messages from one format to another, maintain a record of every message exchanged, track the progress of messages, etc.

Some of these functions are provided by commercial products but generally, no fixed number of extension features will be sufficient for all kinds of applications. A general extension mechanism is needed, provided by the middleware infrastructure to address a variety of messaging requirements, both anticipated and unanticipated, faced by communicating applications. In particular, such mechanism would be particularly useful for Electronic Government to address the needs of government system integration.

4. OBJECTIVES AND WORK PLAN

Based on the state-of-the-art and needs for Electronic Government development, our research concentrates on the problem of extending, in a systematic way, a message-oriented middleware with functions and properties required by high-level Electronic Government applications.

The first stage of the work involves research and development of an open-source core messaging middleware supporting basic communication services. These services enable applications to interact with one another by asynchronously exchanging messages over dynamically created channels. They provide the foundation and the first layer of the middleware. Currently, this foundation layer is formally specified in the RAISE Specification Language (RSL) [16] and implemented in Java.

On top of this layer, programmable services, called *extensions*, will be specified, built, enabled and disabled as needed, to provide specific functionality and to address more complex communication needs. Thus, the second and main stage of our work is the definition and specification of different tools allowing the specification, construction and runtime configuration of the messaging middleware.

Different types of extensions are considered: *functional* and *predicative*, *process-specific* and *process-independent*. Through *functional extensions*, an application obtains access to additional operations, for instance the operation to transform messages from a given input format to a desired output format. Through *predicative extensions*, applications will rely on the satisfaction of certain properties, for instance that all messages transferred through a given channel are recorded and can be later retrieved. *Process-specific extensions* (also called vertical) apply to particular communication scenarios, for instance the cooperative processing by several government agencies of an application for issuing a business license. In contrast, *process-independent extensions* (also called horizontal) apply universally to any communication scenario. Message auditing is an example process-independent extension. Both process-specific and process-independent extensions may be functional or predicative.

Some extensions, for instance transformation, act directly upon messages. Others are oriented on channels and how they are used to route messages, for instance a routing extension connecting channels with one another. Other extensions will be focused on applications, for example allowing them to form an alliance, able to send and receive messages on behalf of all the members. Finally, some extensions will focus on the administration of message queues, for instance a queue may require following the earliest-deadline-first policy for delivering messages in transit. The latter is an example of a queue-oriented predicative extension; depending on the need, this extension could be either process-specific or -independent.

In addition, extensions themselves make use of the core communication services provided by the first layer. For instance, a process-specific extension may create channels, subscribe to a channel, send and receive messages, and finally destroy the channel. Extensions acting upon channels encapsulate the state and behaviour of the component channels. Similarly, extensions acting upon members encapsulate the state and behaviour of members. Consequently, managing abstraction for hiding this complexity is the guiding principle for specifying how these extensions will be defined and configured.

One of the first steps in our research is to precisely define each of the typical extension types. By analyzing the different extension examples for each category, we aim to formally specify all required elements, behaviours and templates, allowing defining more extensions in the future. We believe that for implementing all process-specific extensions we can rely on the core services implemented in the first layer, as primitive functions of a language enabling programming this type of extensions.

Figures 1 and 2 depict the main elements of the middleware: *members* sending and receiving *messages* through *channels*. Figure 1 shows three channels: C1, C2 and C3. C1 provides the basic communication services for sending and receiving messages between Member1 and Member2. C2 has configured authentication, a channel-oriented extension for authenticate messages sent and received through it. Similarly C3 has configured the validation and logging extensions. These are examples of functional extensions. Figure 2 illustrates a process-specific extension that enables to combine channels C1, C2, C3, and C4 in order to satisfy a business-process. Member1 sends messages through C1, C2 and C3. After the messages are processed by the receivers, the reply messages are sent through C4. Finally, Member5 receives the message.

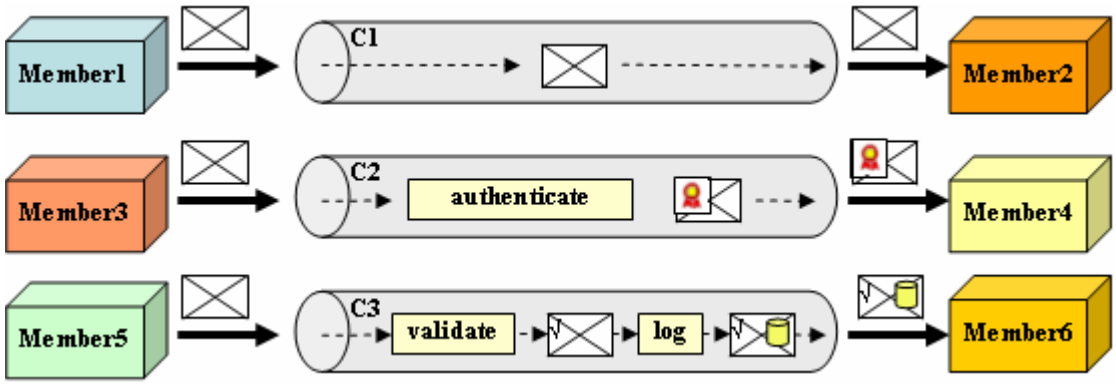

Figure 1: Functional Extensions

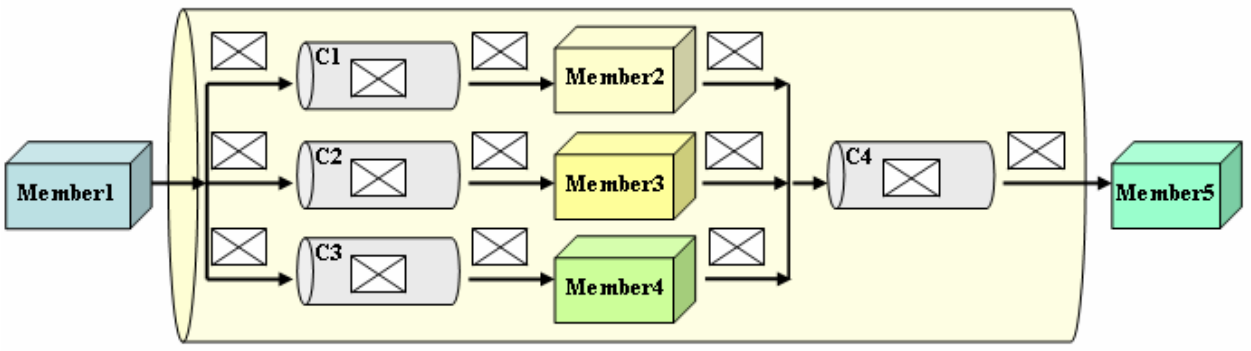

Figure 2: Process-Specific Extensions

More than one extension type may be required simultaneously, and the set of extensions may vary during the application's lifecycle. To address this, we will define a mechanism for run-time configuration of extensions.

Finally, there is a general shortage of formal models to define precise semantics for this type of product, as is common for business applications. We aim to formalize the models built for each extension type, and also to provide design patterns and language for defining future extensions. A formal model will provide the mathematical foundation for specifying and verifying behavioural properties of the extended MOM, and for analyzing if combinations of extensions behave as expected. We plan to explore the compositional behaviour of messaging extensions using process algebra, particularly CSP [17].

5. RELATED WORK

There is a number of formal techniques for integrating components in concurrent and distributed environments. For instance, Pi-calculus [18] models the process of establishing contracts and connectors between components. Pi-calculus is a form of process algebra, developed by Robin Milner to model interactions in concurrent systems. It was adopted to underpin business process specifications by BPMI.org, such as the Business Process Modelling Language (BPML) [19] and the Microsoft's XLANG [20].

We plan to investigate the use of coordination languages such as *Reo* [21] for components composition. *Reo* can describe compositional constructions of connectors that provide the cooperative behaviour of component instances in a component-based system. It is based on a calculus of channels, enabling changes in the topology of connections among components, as well as mobility of components and channels, while preserving the topology. Reo imposes a coordination pattern on the components performing input-output operations through a connector, while remaining totally unaware of the structural and behavioural properties of those components.

The features of channel-based models, such as Reo, are suitable for describing coordination of components. Several channel-based models exist. In particular we would like to explore the use of Dataflow Models, Kahn Networks [22] and Petri-Nets. These can be viewed as specialized channel-based models that incorporate certain basic constructs for primitive coordination. IWIN - Idealized Worker Idealized Manager [23] is an example of this type of models. Manifold [24], a realization of IWIN, is a coordination programming language that supports dynamic reconfiguration of Khan network topologies. The common feature of all these models is the notion of an exogenous channel, located outside components. This concept could be useful for managing middleware extensions.

We also plan to explore the Enterprise Service Bus (ESB) [25] and its relation to our programmable messaging middleware. ESB is an infrastructure combining MOM, web services, transformation and routing. ESB provides a loosely coupled, highly distributed approach to integration. Finally, we plan to investigate the application of semantic web services to the middleware, particularly the use of WSML - Web Services Modelling Language [26] and of WSMO - Web Services Modelling Ontology [27].

6. CONCLUSIONS

This paper presented the background, rationale and objectives of the research program to specify and develop a programmable messaging middleware, with application focus to Electronic Government. First, we outlined the problem of software integration and the concept of middleware as a possible solution to this problem. Second, we described four classical approaches to software middleware: transaction processing monitors, remote procedure calls, message-oriented middleware and object request brokers. Third, we outlined a set of communication requirements related to Electronic Government and how a messaging middleware could address such requirements. Fourth, we describe our research objectives. Finally, we reviewed some work related to this research program.

The main contribution of this work is planned to be a mathematically-based method for formally specifying, implementing, combining and verifying messaging services provided to distributed applications by the underlying communication infrastructure. We also plan to implement a reference messaging middleware to realize and support the above development method. Finally, we plan to apply the method and the reference implementation to build a reliable communication infrastructure for Electronic Government.

7. ACKNOWLEDGEMENTS

We would like to thank Adegboyega Ojo and Gabriel Oteniya for useful comments about this work.

8. REFERENCES

[1] G.Alonso, et al, Web Services, Concepts, Architectures and Applications, Springer, 2004.

[2] R. Schantz, D. Schmidt, Middleware for Distributed Systems: Evolving the Common Structure for Network-centric Applications, Wiley & Sons, 2001.

[3] R. Schantz, D. Schmidt, Research Advances in Middleware for Distributed Systems, World Computer Congress, August 2002.

[4] J. Siegel, CORBA 3, Fundamentals and Programming – Second Edition, Willey, 2000.

[5] W. Grosso, Designing and Building Distributed Applications, Java RMI, O´Reilly, 2001.

[6] T.L.Thai, Learning DCOM, First Edition, O´Reilly, 1999.

[7] Sun Microsystems, RPC Extensions Developers Guide, http://docs.sun.com/app/docs/doc/816-3576

[8] SOAP Specification, http://www.w3.org/TR/soap

[9] IBM, IBM WebSphere, http://www-306.ibm.com/software/integration/wmq/

[10] A. Dickman, Designing Applications with MSMQ: Message Queuing for Developers, Addison Wesley, 1998.

[11] WebMethods Enterprise, WebMethods. http://www.webmethods.com/meta/default/folder/0000005452

[12] WS-BPEL, Web Services Business Process Execution Language, http://www.oasis-pen.org/committees/tc_home.php?wg_abbrev=wsbpel

[13] WS-Coreography Model Overview, http://www.w3.org/TR/2004/WD-ws-chor-model-20040324/

[14] OECD - Organization for Economic Co-operation and Development, The e-Government Imperative, 2003.

[15] R. Stevenson, P. Gibson, Customer and Citizen Focused Public Service Provision, The Scottish Executive Central Research Unit, 2002.

[16] The Raise Language Group, The Raise Specification Language, Prentice Hall, 1992.

[17] C.A.Hoare, Communicating Sequential Processes, 2003.

[18] R. Milner, Communicating and Mobile Systems: The Pi-Calculus, Cambridge University Press, 1999.

[19] BPML - Business Process Modelling Language, http://www.service-rchitecture.com/web-services/articles/business_process_modeling_language_bpml.html

[20] XLang, http://www.nongnu.org/xlang

[21] F. Arbab, F. Mavadatt, Coordination through Channel Composition. Coordination Languages and Models, Proc. Coordination 2002, volume 2315, LNCS, Springer-Verlag, pages 21-38.

[22] R. E. Bryant, D. R. O´Hallaron, Computer Systems: A Programmer´s Perspective (CS:APP), chapter 12, Network Programming, Prentice Hall, 2003.

[23] The IWIM Model for Coordination of Concurrent Activities. Coordination Languages and Models, LNCS 1061, pp.34-56, Springer-Verlag, April 1996.

[24] Manifold version 2: Language Reference Manual, TR: Centruum Voor Wiskunde en Informatica. http://www.cwi.nl/ftp/manifold/refman.ps.Z

[25] D. A. Chappell, Enterprise Service Bus, Theory and Practice, O'Reilly 2004.

[26] WSML - Web Services Modelling Language, http://www.wsmo.org/wsml/

[27] WSMO - Web Services Modelling Ontology, http://www.wsmo.org/