Computer Networks 53 (2009) 1570–1586

Contents lists available at ScienceDirect

Computer Networks

journal homepage: www.elsevier.com/locate/comnet

Transparent autonomization in CORBA

S. Masoud Sadjadi [a,*], Philip K. McKinley [b]

[a] Florida International University, Miami, FL 33199, United States
[b] Michigan State University, East Lansing, MI 48824, United States

ARTICLE INFO

Article history:
Available online 29 December 2008

Keywords:
Transparent shaping
Adaptive middleware
CORBA
Autonomic computing
Self-optimization
Dynamic adaptation
Quality-of-service
Mobile computing
Generic proxy

ABSTRACT

Increasingly, software systems are constructed by integrating and composing multiple existing applications. The resulting complexity increases the need for self-management of the system. However, adding autonomic behavior to composite systems is difficult, especially when the constituent components are heterogeneous and they were not originally designed to support such interactions. Moreover, entangling the code for self-management with the code for the business logic of the original applications may actually increase the complexity of the systems, counter to the desired goal. In this paper, we address autonomization of composite systems that use CORBA, one of the first widely used middleware platforms introduced more than 17 years ago that is still commonly used in numerous systems. We propose a model, called Adaptive CORBA Template (ACT), that enables autonomic behavior to be added to CORBA applications automatically and *transparently*, that is, without requiring any modifications to the code implementing the business logic of the original applications. To do so, ACT uses "generic" interceptors, which are added to CORBA applications at startup time and enable autonomic behavior to be introduced later at runtime. We have developed ACT/J, a prototype of ACT in Java. We describe a case study in which ACT/J is used to introduce three types of autonomic behavior (self-healing, self-optimization, and self-configuration) to a distributed surveillance application.

Published by Elsevier B.V.

1. Introduction

Driven by the Internet revolution and its effects on information technology, the last decade has witnessed proliferation of integration middleware technologies addressing software integration problems [1]. Instead of developing new software systems from scratch, the focus of integration middleware technologies has been on leveraging available software resources by enabling their interoperation [2]. CORBA 2.0 [3], released in 1996, was among the first middleware technologies to address integration issues [4], and since then CORBA has been successfully used in the integration of numerous software systems [5].

A typical CORBA application comprises heterogeneous software components, often developed in different programming languages and targeting different platforms (operating systems, devices, and networks). Indeed, a major goal of CORBA and other middleware platforms is to hide this heterogeneity from the business logic of the integrated applications. While this approach helps developers to integrate their systems more easily, the management of complex CORBA-based systems is challenging, especially as they evolve to accommodate new software and hardware technologies. In particular, managing composite systems involves ensuring non-functional concerns such as quality-of-service, fault tolerance, and security. Unfortunately, these concerns are often directly affected by the underlying technologies and the environments in which the application is deployed.

Autonomic computing [6] promises a general solution to the management problem that relies on complex systems

* Corresponding author. Tel.: +1 305 348 1835.
E-mail addresses: sadjadi@cs.fiu.edu (S.M. Sadjadi), mckinley@cse.msu.edu (P.K. McKinley).
URLs: http://www.cs.fiu.edu/~sadjadi (S.M. Sadjadi), http://www.cse.msu.edu/~mckinley (P.K. McKinley).

1389-1286/$ - see front matter Published by Elsevier B.V.
doi:10.1016/j.comnet.2008.12.012

to manage themselves. Instead of requiring low-level interaction with users or system administrators, self-managing systems would require only high-level human guidance – defined by goals and policies – in order to work as expected. Each autonomic element in the system comprises a *managed element*, implementing the business logic of the system, and an *autonomic manager*, implementing the self-managing behavior of the system. However, self-management concerns (self-healing, self-optimization, self-configuration, and self-protection) tend to crosscut the functional decomposition in the managed elements [7–9]. Consequently, if the code for self-management is entangled with the code for the business logic of the original systems, then the complexity of managing the resulted autonomic system may actually increase, contradicting the purpose of autonomic computing.

This paper describes the *Adaptive CORBA Template* (*ACT*), a framework that enables dynamic addition of autonomic behavior to existing CORBA systems, without modifying the application code. At startup time, ACT turns the constituent software programs into managed elements by transparently inserting generic hooks capable of intercepting all CORBA remote interactions. Next, at run time, these hooks can be used to introduce autonomic managers into the system. An autonomic manager in turn can intercept the requests, replies, and exceptions that pass through the CORBA Core (called ORB, which stands for Object Request Broker), adapting or redirecting them as needed. Effectively, ACT enables *transparent autonomization* (i.e., transparent addition of self-managing behavior) in CORBA applications.

We identify three types of applications that may benefit from such a capability. First, dependable applications are required to operate continuously without interruption; code for handling newly discovered faults and in general self-managing behavior can be added to these applications as they execute. Second, embedded applications are required to provide very small footprints; a minimal autonomic code can be added to the application at compile or startup time, while optional and temporary autonomic code can be swapped in and out as needed during run time. Third, the source code for some legacy CORBA applications may be unavailable, or modifying the source code may be undesirable. Such applications can be autonomized transparently using ACT, without modifying or even recompiling the original application source code.

Various aspects of the ACT framework have been described in earlier conference papers [10,11]; this paper provides a complete picture of the ACT project, presents a more comprehensive architectural solution, and includes additional details of the autonomization process and experimental results for three different types of autonomic behavior: self-healing, self-optimization, and self-configuration. The remainder of this paper is organized as follows. Section 2 provides a background on CORBA and describes the architecture and operation of ACT, as well as a Java prototype, ACT/J. Section 3 presents a case study where we used the ACT prototype to add three different types of autonomic behavior to an existing surveillance application. Section 4 categorizes related research projects, and Section 5 provides concluding remarks.

2. ACT architecture and operation

ACT is intended to support the construction of adaptive CORBA applications from existing CORBA applications transparently, that is, without modifying the original application functionality. ACT should enable CORBA applications to support adaptive behavior at run time without the need to stop, modify, recompile, relink, or restart the applications. Moreover, ACT should introduce nominal overhead to the performance of the existing applications. With these design goals in mind, we developed a two-step process that supports transparent autonomization in existing CORBA applications. In the rest of this section, we provide a brief overview of CORBA and the autonomization process, describe the architecture and internal operation of ACT, and discuss the prototype implementation of ACT in Java.

2.1. CORBA background

The *Common Object Request Broker Architecture* (*CORBA*) [12] is an integration and distribution middleware specification defined by the Object Management Group (OMG) [4]. Fig. 1 depicts a simple client–server CORBA application comprising a client and a server program and their orientation among three system layers: application, middleware, and network.

Let us assume that the client has a valid reference to the CORBA object realized by the servant. For clarity, a broker program is not shown. The *Object Request Broker (ORB)*, the core of CORBA, allows objects to interact transparently with other objects (located locally or remotely). A CORBA object is represented by its interface, is identified by its reference, and is realized in an object-oriented program as a local object called the *servant*. The client calls methods on the servant as if the CORBA object were located in the client address space. The *Interface Definition Language (IDL)* is a language for defining CORBA interfaces. An IDL compiler is used to automatically generate the code for stubs and skeletons. An *IDL stub* represents a servant locally in the client address space and an *IDL skeleton* represents a client locally in the servant address space. IDL stubs and skeletons marshal and unmarshal requests and responses to enable object interactions over a network.

CORBA Portable Request Interceptors provide a transparent mechanism to intercept messages (reified requests, replies, and exceptions) inside the ORBs of a CORBA application. For example, a portable interceptor can be used to forward a particular request to a *different* CORBA object (e.g., forwarded request flows 2, 3, and 4 in Fig. 1). However, to ensure portability, interceptors are not allowed to reply to intercepted requests or to modify the parameters [12]. This restriction limits the ability of request interceptors alone to adapt the behavior of CORBA applications.

2.2. Autonomization process

Fig. 2 illustrates the two-step process to autonomize an existing CORBA application using ACT. As this process is