

Contents

1	Overview	3
1.1	The Binding Framework	4
1.2	The Communication Framework	5
1.3	The Configuration Framework	5
1.4	The Resource Framework	5
1.5	Jeremie: the Java RMI Personality	6
1.6	David: the CORBA Personality	6

JONATHAN

Chapter 1

Overview

Sacha Krakowiak

This document is made available under the terms of the GNU Free Documentation License as published by the Free Software Foundation (<http://www.gnu.org/copyleft/fdl.html>)

Jonathan¹ is a framework for building Object Request Brokers (ORBs), which are central components of middleware platforms. The development of Jonathan was motivated by the lack of openness and flexibility of the currently available middleware systems. In order to facilitate the construction of ORBs adapted to specific runtime constraints or embodying specific resource management policies, Jonathan provides a set of components from which the various pieces of an ORB may be assembled. These components include buffer or thread management policy modules, binding factories,marshallers and unmarshallers, communication protocols, etc. In addition, Jonathan includes configuration tools that facilitate the task of building a system from a set of selected components and allow the developer to keep track of the description of a specific system.

Jonathan is organized in four frameworks implemented in Java, each of which provides the Application Programming Interfaces (APIs) and libraries dedicated to a specific function.

- *Binding.* The binding framework provides tools for managing names (identifiers) and developing binding factories (or extending available ones). Different inter-object binding models may be managed, allowing for example the use of different qualities of service.
- *Communication.* The communication framework defines the interfaces of the components implied in inter-object communications, such as protocols and sessions. It provides tools for composing these pieces to construct new protocols.
- *Resources.* The resource framework defines abstractions for the management of various resources (threads, network connections, buffers), allowing the programmer to implement

¹Dumant, B., Horn, F., Dang Tran, F., and Stefani, J.-B. (1998). Jonathan: an Open Distributed Processing Environment in Java. In Davies, R., Raymond, K., and Seitz, J., editors, *Proceedings of Middleware'98: IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 175-190, The Lake District, UK. Springer-Verlag.

new components to manage these resources, or to reuse or extend existing ones.

- *Configuration.* The configuration framework provides generic tools to create new instances of components, to describe a specific instance of a platform as an assembly of components, and to create such an instance at boot time.

The main components of Jonathan (i.e. the classes that make up the frameworks) have themselves been developed using uniform architectural principles and a common set of patterns and tools (factories, helpers).

Jonathan includes two specific ORBs (“personalities”), which have been developed using the above frameworks. These personalities are Jeremie, an implementation of Java RMI, and David, an implementation of the OMG Common Request Broker Architecture (CORBA).

The next sections give an overview of the frameworks and personalities, which are examined in more detail in the remaining chapters.

1.1 The Binding Framework

Since naming and binding are two closely related functions, we first introduce the main notions related to naming in Jonathan. A *name* is an information that identifies an object (allows it to be distinguished from other objects). A name is only valid in a *naming context*, which specifies a set of associations, or bindings, between names and objects. Such associations may be indirect, i.e. a name may be linked to an object through a chain of bindings, across several naming contexts.

Names may take various forms (e.g. a symbolic name, an index in an array, a physical address, etc.). In Jonathan two specific forms of names are defined: an *identifier*, which designates an object interface in a specified naming context; a *session identifier*, which designates a session in the context of a communication protocol (see Section 1.2).

Binding is the process of interconnecting a set of objects in a computing system. The result of this process, i.e. the association, or link, created between the bound objects, is also called a binding. The purpose of binding is to create an access path through which an object may be reached from another object. Thus a binding associates one or several sources with one or several targets.

The simplest form of binding is the association of a name with the object that it designates (a naming context is a set of bindings). Actual access from an identifier `id` to the object that it designates is carried out through the operation `id.bind()`, which returns the bound object. The object needs to have been previously made available in a naming context `nc` through the operation `nc.export(obj)`.

More elaborate bindings may take place, e.g. between a client and a remote server, or between several parties linked by a group communication protocol. The export-bind pattern is also used in these cases. A complex binding is made up of a set of parts (e.g. a client stub, a communication session and a server stub), which make up a *binding object*. Binding objects are created by binding factories. A *binding factory* is designed for a specific situation (application-level and communication protocols, resource requirements, etc.) and includes factories for all the parts that make up the binding object, as well as coordination code that calls these factories when needed. The notions of binding object and binding factory are inspired by the Open

Distributed Processing (ODP) Reference Model².

The Jonathan binding framework includes interfaces and libraries for managing identifiers and bindings. Specific binding factories are included in the personalities that use Jonathan (Sections 1.5 and 1.6)

1.2 The Communication Framework

The Jonathan communication framework is based on two main notions: protocols and sessions. A session is the abstraction of a communication channel; it is used to send and receive messages between the processes connected to the session. A protocol is a session manager: it acts as a naming and binding context for sessions.

Protocols are usually organized in stacks or acyclic graphs, each protocol using the resources of the lower level protocols, down to the basic communication interface provided by the underlying hardware or operating system. Likewise, sessions are organized in a hierarchy that reflects the protocol hierarchy. Sessions have a “higher” and a “lower” interface, which are respectively used to send messages down and up the session stack or graph. This scheme is similar to that of the *x*-kernel³ framework.

Setting up a session between communicating entities is an instance of binding, for which the export-bind pattern is used. First, a protocol graph is created to represent the layered organization. This protocol graph **exports** a session interface, returning a session identifier that prospective clients may use to obtain a session through a **bind** operation.

At the lowest level, sessions use the communication interface provided by the operating system, in the form of connections (Section 1.4) that encapsulate sockets.

The basic protocols provided are TCP/IP, IP Multicast and a simplified version of RTP (Real Time Protocol).

1.3 The Configuration Framework

The configuration framework deals with the setup of a Jonathan platform at boot time (dynamical re-configuration will be provided in the future). The framework includes tools for the description of a specific configuration in a declarative form (an XML text conforming to a specified DTD), and for generating a platform conforming to this description.

The framework, in effect, defines a composition model in which the components that make up a configuration are specified by a set of properties. An instance of a component, having the specified properties, is created when needed (usually at boot time). The properties include attribute values, and also dependencies between components (e.g. a component may use a specified scheduler or binding factory).

²ITU-T & ISO/IEC, Recommendations X.901, X902, X903 & International Standards 10746-1, 10746-2, 10746-3: “ODP Reference Model: Overview, Foundations, Architecture”, 1995

³N. Hutchinson and L. Peterson. The *x*-kernel: An Architecture for Implementing Network Protocols, *IEEE Transactions on Software Engineering*, 17(1), pp. 64-76, Jan. 1991

1.4 The Resource Framework

The resource framework gives easy access to the management of various types of resources used by ORBs.

- Memory management, in the form of *chunks* (portions of byte arrays) that may be linked to form unbounded buffers, and reused to reduce allocation and garbage collection overhead. Chunks are mainly used for message management. Splitting, combining and duplicating messages is done at low cost through pointer manipulations.
- Activity management, in the form of *jobs* and *schedulers*, which define possibly multi-threaded activities and the policy used to run these activities using the available processing resources.
- Network communication management, in the form of *connections*, which provide low-level communication mechanisms for sending and receiving messages. Connections are implemented using the available communication interface; however, the connection management interface is independent of the communication mechanism. The current implementation is based on sockets.

1.5 Jeremie: the Java RMI Personality

Jonathan allows the use of Java Remote Method Invocation, by providing the compilation tools and the GIOP communication protocol.

Jeremie has some non-standard features:

- a dynamic programming interface, similar to CORBA's DSI;
- support for RTP and IP Multicast bindings;
- context service transmission in the GIOP messages.

Compared to the reference implementation, the support for server activation is still missing.

1.6 David: the CORBA Personality

Jonathan provides a CORBA personality, with the following features.

- CORBA 2.3 compliant IDL compiler;
- IIOP protocol (version 1.0).
- Dynamic Invocation Interface (DII) and Dynamic Skeleton Interface (DSI).