An Agent-based Architecture for Building CORBA Distributed Systems

Fernando Bellas*, Ramón Juanes*, Nieves Rodríguez+ and Ángel Viña*

*Departamento de Electrónica e Sistemas, Universidade da Coruña, A Coruña, CP/15071, Spain.
E-mail: {fer, nieves, avc}@des.fi.udc.es

+Departamento de Electrónica y Sistemas, Universidad Alfonso X El Sabio, Madrid, CP/28691, Spain.
E-mail: rjuanes@uax.es

1. Introduction

Development of distributed applications is a complex task. Object Requests Brokers (ORBs), such as those provided by CORBA, DCOM or Java RMI are making easier the development of large and complex distributed applications. All of these frameworks allow the programmer to invoke methods of remote objects as if they were local. CORBA is standardized by the OMG, the largest software consortium in the world, and provides the most interoperable solution.

Even though ORBs facilitate the development of distributed applications, the programmer still has to deal with a complex framework, which increases the learning curve. This is specially true for users with no previous experience in developing distributed applications. In this brief announcement we outline very briefly an agent-based architecture for building CORBA distributed systems, that tries to facilitate the development of such systems. The architecture is implemented on top of a CORBA ORB, and provides the programmer with high-level abstractions. It is made up of a C++ framework, implementing the core functionality, and an Agent Definition Language, that allows to specify the key components of an agent, hiding the underlying framework from the programmer.

2. The Agent-based architecture

Before explaining the architecture, it is necessary to outline the methodology of analysis we are using for building distributed systems. This methodology, hereinafter the FUN (FUNctional Unit) analysis, proposes an analogy between a system and a human organization. Following the FUN analysis, a system is broken down into a set of agents and FUNs, which provide other agents/FUNs with services. Under this methodology, an agent is an autonomous entity that provides the agent community with services. A FUN is a special class of agent that delegates part of its functionality into a number of cooperating agents, under the FUN’s coordination.

A FUN, in the same way as a human organization, specifies the roles its members have to fulfill. This way, a FUN delegates part of its work into its members. A role describes the part to be played by an agent (that may itself be a FUN) within a FUN in terms of: (1) the role title, (2) the cardinality (the required number of members taking on this role), (3) its joining benefits, and (4) its joining criteria. Joining criteria are specified in terms of services and resources. An agent must satisfy the joining criteria in order to take on a role in a FUN, which will make use of these services and resources in order to carry out its own services. Joining benefits are specified in terms of services, resources and service responsibilities (FUN’s services delegated to the agents playing this role), that an agent gains as a consequence of joining the FUN in this role.

The agent-based architecture, CABLE, is built on top of Orbix 2.2 MT, a CORBA 2.0 compliant ORB, and tries to facilitate the transition from the design to the implementation, by providing the programmer with high-level abstractions, such as, agent and FUN, and therefore giving direct support to the FUN analysis. CABLE is made up of a C++ framework (implementing the core functionality) and an Agent Definition Language (ADL). The purpose of ADL is to hide the complexity of the underlying framework from the programmer, who only needs to learn ADL, a small subset of the CABLE framework and CORBA IDL (it is not necessary for the programmer to learn the API of the underlying CORBA ORB). ADL provides the programmer with a very simple language for specifying the key components of an agent (for example, the services it provides), in a similar way to a template, in which he/she will embed C++ (for example, for the implementation of the services the agent provides). The ADL compiler parses ADL files into C++ files. These files, together with the C++ files provided by the programmer, when linked, provide the agent’s implementation. The ADL compiler also allows to generate code in order to allow a CORBA agent to communicate with a CABLE agent. As CABLE is built on top of a CORBA ORB, a CABLE agent can communicate with a CORBA agent.

Among the key components of a CABLE agent, we can mention: (1) activities (look like Ada tasks), which can be internal or external (their entries can be called from outside the agent; they are also called services); (2) working memory (a placeholder for global data); (3) initialise and (4) finalise section; and (5) the roles for FUN agents, in terms of joining criteria. ADL provides language constructs for specifying these components. The programmer will embed C++ for their implementation, if applicable (for example, for the services). Agent inheritance is also supported. A special agent, the Manufacturer, runs on a machine of the distributed system, and launches and kills agents when necessary. For example, when a FUN agent comes into life, if there are no enough agents running that fulfill its roles, the Manufacturer launches them. When an agent is no longer required, the Manufacturer kills it.

Therefore, we have adopted a language based approach, instead of a framework based approach. Proponents of frameworks argue that frameworks are more open to developers, provide greater flexibility, and obviate the need to learn yet another language. However the type of language based approach adopted by CABLE does offer some advantages over the framework based approach, namely, (1) ease of use (there is no need to learn a complex framework), which is specially true for inexperienced users, and (2) a closer mapping from agent design to implementation (provides high-level abstractions such as agent and FUN).

We are using this architecture in the EUCLID RTP 6.1 European research project, undertaken by the GRACE (Grouping for Research into Advanced C3I for Europe) consortium. The demonstrator being developed is made up of a number of graphical facilities that help military users in the decision taking process. All the facilities, and the demonstrator as a whole, have been analyzed with the FUN approach, and implemented with CABLE.