A Development Framework for Network Management Systems based on Reconfigurable Components

E.Jaén, J.Serrat
Universitat Politècnica de Catalunya
Barcelona
Spain
enric@estos.upc.es, serrat@tsc.upc.es

G.Pavlou
University of Surrey
Guildford, Surrey
United Kingdom
G.Pavlou@eim.surrey.ac.uk

Abstract

This contribution presents a prototype of network management framework based on distributed plug-in component technology. The main innovative aspect is the exhibited capacity to facilitate network management applications to be reconfigured at runtime by changing the communication channels used by the constituent components to communicate. This adds great flexibility because the system functionality is not "closed" at the system design time. The ensemble constitutes a development framework for component-based management applications. An scenario is presented to demonstrate the system in action.

Keywords

Component framework, dynamic management system, events.

1. Introduction

Software applications in any topic area are typically conceived as monolithic black boxes. This means that they are glued parts of code with internal dependencies that cannot be easily separated, observed or changed. This is a serious drawback with respect to code reusability in other applications, modification of its functionality and potential integration with other vendor products. The idea of component-based applications tries to solve the afore-mentioned problems. Thus it tries to open those internal dependencies, yielding modular architectures with public interfaces between constituent modules [1]. Architectures conceived in this way are easily reusable, extensible and compatible with other implementations by replacing pieces of code or adding new ones.

The contribution of this paper is in the area of integration frameworks. Other topics in components where significant contributions have been made include component description languages, repositories to store components, composition tools and composition models. By using the presented framework, the application administrator can at runtime insert or extract components and dynamically define the links between components without stopping the execution. This can be done because components are conceived as groups of CORBA objects with reflection capability, i.e. they are able to make public the events that can emit/receive. For example, a routing algorithm component could be changed with a more optimum one or a component with a poor performance could be duplicated or relocated.

2. The Integration Model

The component framework described hereafter is an enhancement of the approach initially conceived Flowthru project [2]. Figure 1 shows a high level functional architecture. The elements with triangular shape represent the components. Components can be classified as application components and support components (application-independent). Support components capture events and perform actions such as encrypting event payload, monitoring, compressing data, filtering events and so on. The elements with elliptic shape are framework services. The novelty of this contribution is the Integration service.

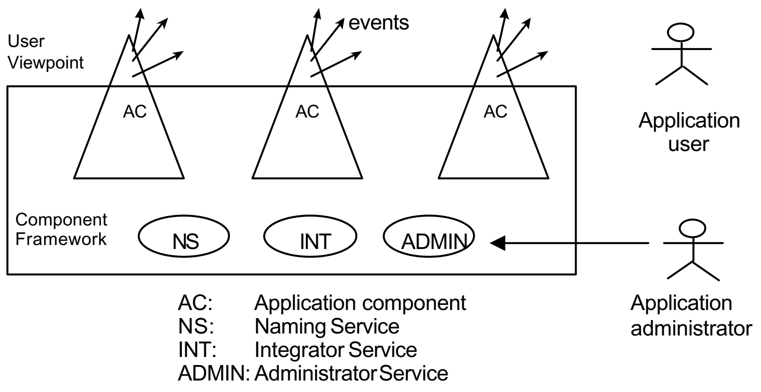

Figure 1: Component framework architecture

In this framework components are CORBA objects. Components can be classified as application components and support components (application-independent). Support components capture events and perform actions such as encrypting event payload, monitoring, compressing data, filtering events and so on.

The Naming Service is a standard CORBA service used to store component references in a directory-based structure so that they can be transparently located under the same context.

The Administrator allows the user (application administrator) to choose the connections that will be established between components. Besides, it lists all the components currently in the system, it shows which events produces/consumes each component, and allows the user to create the event channels.

The Integrator service maintains at runtime the channels established between the components. It receives requests to add/remove components, and subscribe/unsubscribe event channels. Figure 2 depicts the process of inserting a new component in an application.

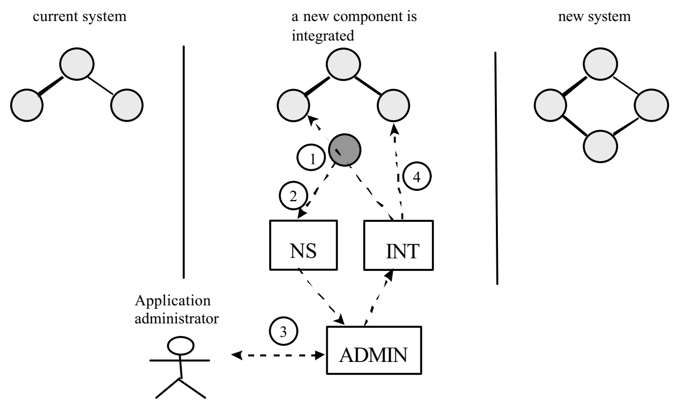

Figure 2: Steps involved in the insertion of a new component in a system

In step 1, the administrator launches the component, which is wrapped inside a CORBA server. In step 2, the component is registered automatically; the component contacts the Naming Service and registers itself under the */component* context. In step 3, the administrator connects manually the component. The establishment of a connection means to create an event channel for each event consumed/produced. Finally, in step 4 the Integrator creates the connections by invoking directly to the components.

3. The Component model

The component model in this framework supports event introspection. An event is modelled as a packet with a header and a body. The header identifies the type of event. The body contains the business information. The body is implemented as a sequence of CORBA type Any, containing the business information.
From a client point of view, a component behaves like a CORBA object. It contains three types of CORBA objects; an object for each event channel consumed an object for each type of event produced and finally an object for the introspection interface. See Figure 3.

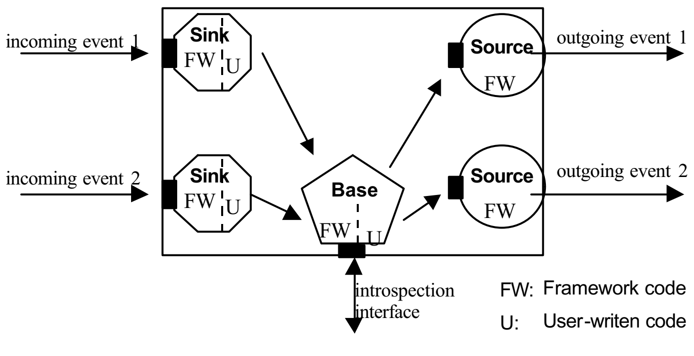

Figure 3: Component Architecture

4. Use Case: ATM configuration system

To test the framework, a prototype of a simulated ATM configuration management application was developed. The application is able to detect new nodes, store them in a repository, establish connections and display them in a GUI. It is decomposed in the following components: Node, Network, Planner, Administrator GUI, Operator GUI, a component collector of events and a generic listener of events. See Figure 4. The operator is able to add new components on line, modify its connections and to replace existing components. See Figure 4. This prototype was implemented using free CORBA platforms [3].

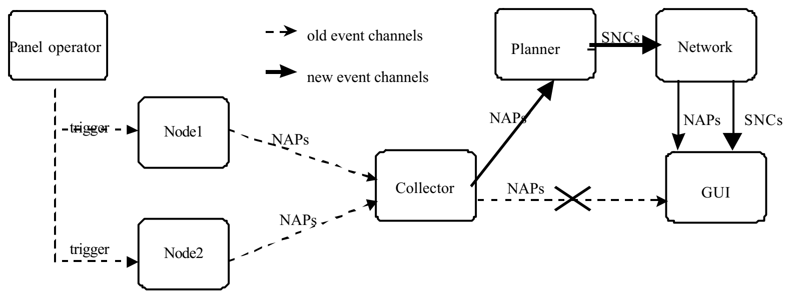

Figure 4. Prototype showing how events can be redirected. All connections are dynamic.

Conclusions

The proposed initial framework has worked fine in applications conceived to stress its capabilities, although improvements are currently under way. For example, it will use the standard CORBA Event Service as the communication service to isolate component crashes, there will be predefined configuration of component connections, a new component installation service will be developed and additional middleware services such as Transaction or Persistence Services will be supported. We also plan to redefine the component model to be fully compliant with the standard CORBA Component Model [4].

Acknowledgement

E.Jaén wishes to thank Dr. Johan Zuidweg for his support in the research that has been the base for this paper.

References

[1] SIGS Component Strategies, February 1999
[2] Flowthru ACTS 335 project, 1998-2000.
[3] MICO and JacORB, freeware CORBA implementation
[4] Corba Component Model, OMG:orbos/ccm-99-02-05.pdf