A PATTERN FOR INTERCONNECTING DISTRIBUTED COMPONENTS

Walid Gaaloul[1], Karim Baïna[1,2], Khalid Benali[1], and Claude Godart[1]
[1] *LORIA - INRIA - CNRS (UMR 7503)*
BP 239, F-54506 Vandœuvre-lès-Nancy Cedex, France
[2] *School of Computer Science and Engineering,*
The University of New South Wales, Sydney NSW 2052, Australia

Keywords: Object Orientation in Internet and Distributed Computing, Internet and Collaborative Computing, Process Design and Organisational Issues, Enterprise-based Component Architectures, Architectural Patterns for Object-based Components and Applications, Distributed Component Cooperation.

Abstract: Nowadays, enterprises express huge needs for mechanisms allowing interconnection of their business components. Due to the weakness of component integration facilities, a large amount of research and development has been made in this area. Nevertheless, developed mechanisms are generally hard-coded, proprietary and lack a high level of abstraction. This paper presents our contribution to the design, the implementation, and the experimentation of an architectural pattern named “Service”. This pattern is able to support interconnection and cooperation between distributed components independently of their specific contexts (workflow processes, database robots, agents, networks nodes, etc.). Our “Service” pattern proposes a generic solution to interconnection and cooperation between components through object oriented structures and scenarios. The essence of the pattern is the ability for ”Service” to provide registration, discovery, negotiation and dynamic API information on behalf of a contained service. Moreover, several alternatives are presented to implement our pattern.

1 PROBLEMATIC

Applications manipulate considerable number of distinct and heterogeneous components. By components we mean all resources needing to be discovered on a network, and to be used through customised access rights (*e.g.* workflow processes, database robots, agents, distributed objects, networks nodes, etc.). These components are often used to cooperate with each other to achieve common goals of the composed applications.

Through our pattern “Service”, we bring a generic solution to the problem of components interconnection and cooperation. We express the genericity underlying the concepts of components interconnection and cooperation which are distinct and heterogeneous as an architectural pattern we name “Service”. This pattern will express a structural schema for cooperation of heterogeneous components which enables reusability of design and architecture concepts.

2 SERVICE PATTERN : STRUCTURE

Our contribution is in the same vain of research works on interconnection and cooperation patterns (Pyarali et al., 2000; Benatallah et al., 2003; Arsanjani, 1999). It is specified through the description of its structure elements, their responsibilities and their manner to collaborate in the same pattern building structure (Buschmann et al., 1995; Schmidt et al., 2000). ”Service” pattern can be used when there is a need of cooperation within an environment of heterogeneous and monolithic components.

”Service” pattern provides a conceptual schema for distributed component to suit needs for dynamic cooperation. It helps to decompose a cooperative system to software bricks that deal with organisational and structural aspects such as component presentation, and with behavioral aspects such as component access rights, component security, etc.

Our pattern “Service” is composed of two structures :

1. a service definition structure which enables to de-

Gaaloul W., Baïna K., Benali K. and Godart C. (2004).
A PATTERN FOR INTERCONNECTING DISTRIBUTED COMPONENTS.
In *Proceedings of the Sixth International Conference on Enterprise Information Systems*, pages 430-434
DOI: 10.5220/0002643404300434

fine, customize the access to components, and manage numerous existing components ;

2. a service discovery and negotiation structure which enables to support operations of either component lookup according to structured properties, or dynamic customisation of component properties and access rights.

2.1 Service definition

UML class diagram of figure 1 represents the role played by a service to abstract and to adapt a component subject in order to customize the access to its resources. Thus, a "Service" is seen as a software entity that encapsulates a component to keep only visible the elements necessary to its interconnection and cooperation with other application components.

Subject

- Defines an object interface, which is specific to a particular component.

Service

- Manages meta-information that keep possible discovery, negotiation and access control to a component;
- Dynamically controls the access to the component it encapsulates;
- Has the capability to dynamically extract the API of its component, in order to ensure the control of a component method invocation or a component event notification;
- Provides a new interface of the component it adapts: (*e.g.*, complementary functionalities related to negotiation and discovery);
- Is able to dynamically extract the adapted component category and properties attributes in order to build a profile enabling service discovery and comparison to other adapted components;

Category

- Enables components "Subject" to be categorised thanks to application specific taxonomy (*e.g.*, object taxonomy, XML hierarchical taxonomy, etc.).

Profile

- Describes meta-information related to the component "Subject" as an application specific set of named-value attributes properties.

APIManager

- Stores information on the component "Subject" in terms of methods "Method" and events "Event";
- Enables the access control to the previously negotiated interface of the component "Subject";
- Method and Event enable the component data, status, and resources management through its primitives invocation and events notification.

ServiceContainer

- Defines a repository of "Service" objects that is necessary to their discovery, negotiation and access;
- Enables global processing to be applied to services of the same category (*e.g.*, mining, verification, transformation, etc.);
- Gathers a view of "Service" objects according to particular criteria.

ConcreteServiceContainer

- Defines a concrete "ServiceContainer" repository (*e.g.*, RequestedServiceContainer : gathering requested services, ProvidedServiceContainer : gathering provided services, WrappedServiceContainer : gathering already negotiated and bound services).

2.2 Service discovery and negotiation

UML class diagram of figure 2 isolates operations related to components discovery (*e.g.*, matching calculus between components and component neighbourhood calculus), and operations related to components negotiation. These operations are dynamic and efficient cooperation tools. On one hand, service definition and discovery aim to structure component service containers and "position" them relatively to each other. On the other hand, service negotiation enables to distinguish between discovered components and to select those that propose most interesting cooperation contract comparing to other neighbour components.

2.2.1 Service discovery

Locator

- Manages the localisation of the objects "Service";
- Enables to discover realisations of the object "Subject" according to a given "LocationStrategy";
- Depends on the object "Matcher" that ensures the matching calculus necessary to every service localisation.

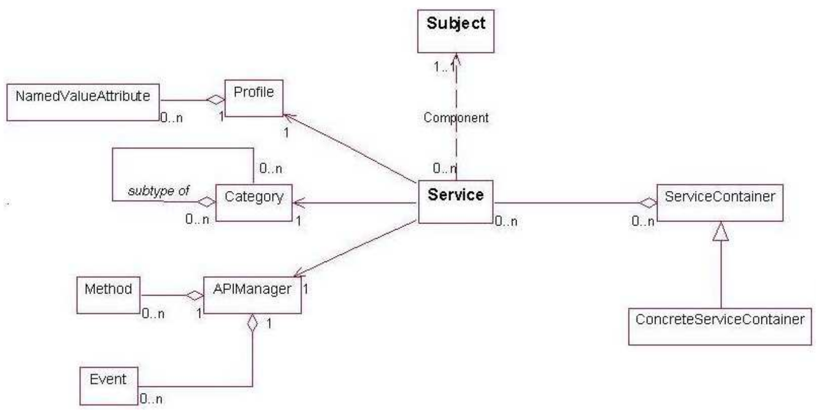

Figure 1: Service definition structure

LocationStrategy

- Dynamically manages service discovery algorithms classes (according to methods, events, profile, category, etc.);
- Declares a common interface to every service discovery algorithm;
- Enables discovery algorithms to be independent and interchangeable by choosing dynamically an algorithms implemented in one of "Concrete-LocationStrategies";

Matcher

- Manages the matching calculus for the objects "Service" ;
- Is composed of strategies "MatchingStrategy" for matching calculus of objects "Service".

MatchingStrategy

- Declares a common interface to every service matching calculus algorithm;
- Enables the management of service matching calculus algorithms and their refinement;
- Enables service matching calculus algorithms to be chosen at runtime from one of "ConcreteMatching-Strategies";

2.2.2 Service negotiation

Negotiator

- Manages the negotiation of "Service" objects according to a given "NegotiationStrategy";
- Enables negotiation of interface access of the object "Subject";
- Knows the set of negotiations concerning the object "Service";
- Enables the application of global processing to negotiations of the same type (*e.g.*, analyse, tactical application, coordination, etc.);

NegotiationStrategy

- Declares a common interface to every service negotiation algorithm;
- Enables service negotiation algorithm to be chosen at runtime from one of "ConcreteNegotiation-Strategies";

3 IMPLEMENTATION ISSUES AND SERVICE APPLICATION EXPERIMENTS

3.1 Implementation issues

Among numerous possible solutions for implementing the facilities of "Service" pattern, we summarise some of them in table 1.

3.2 Service pattern applied to Internet printers

As an example of known service pattern use, we will detail Internet printing service through IPP (*Internet Printing Protocol*) . As described by the RFC (IETF, 2000a), Internet printing service possesses the following properties :

Table 1: Service pattern implementation technologies

Facility	Solution 1	Solution 2	Solution 3	Solution 4	Solution 5
Services	CORBA Servants	CCM Components	Web Services		
Containers	CORBA Trading Service		WebDAV Name Spaces	RDF Containers	UDDI Repository
Profiles	CORBA Trading Service Properties		WebDAV Properties	RDF Properties	XML Files
Methods	IDL Interface	CIDL Interface	WSDL Interface		
Events	CORBA Event Service	CCM Events	Message Oriented Middleware (MOM)		
Interface visibility	CORBA Security Service		ad-hoc facilities		
Discovery	CORBA Trading Services or CORBA Naming Service		WebDAV Searching and Locating (DASL)	RDF-enabled search engine	UDDI
Negotiation	OMG Negotiation Facility, Trading Partner Agreement (TPA) or ad-hoc facilities				

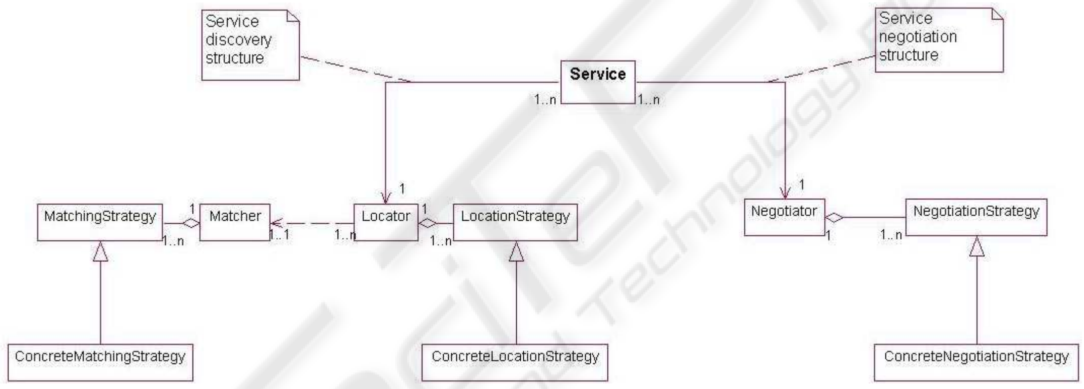

Figure 2: Service discovery and negotiation structure

- A *profile* : (printer-name, printer-uri-supported, printer-location, printer-state, color-supported, multiple-document-jobs-supported, document-format-supported, printer-resolution, etc.) (IETF, 2000b);
- *Operations* : (Print-Job, Get-Jobs, Pause-Printer, Purge-Jobs, Get-Printer-Attributes, etc.) (IETF, 2000b);
- *Events* : (printer-is-accepting-jobs, job-impression-completed, job-state, etc.);
- Requirements for *discovery mechanisms* within a networks of printers : according to printer-name, or according references stored in printer-urisupported attribute which represent the set of all supported URI identifier by the printer, or according to information stored in printer-location attribute which is a character string of 127 bytes describing the printer location in natural language -*e.g.*, "Room 123A, 1st floor, building A"-), or according to values of other attributes (IETF, 2000b);
- Requirements for *negotiation mechanisms* of either printer supported content format or other printer attributes values (IETF, 2000b; IETF, 2001);
- And requirements for printer *access privileges and roles mechanisms* : printing, jobs management, supervision, etc. (IETF, 2000b).

REFERENCES

Arsanjani, A. (1999). Service Provider: A Domain Pattern and its Business Framework Implementation. In *Pattern Languages of Programs (PLoP'99)*.

Benatallah, B., Dumas, M., Fauvet, M.-C., and Rabhi, F. A. (2003). *Patterns and Skeletons for Parallel and Distributed Computing*, chapter Towards Patterns of Web Services Composition, pages 265–296. Springer-Verlag.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., and Stal, M. (1995). *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns*. Wiley, John and Sons.

IETF (2000a). *Internet Printing Protocol - IPP/1.1: Model and Semantics*. IETF (Internet Enegineering Task Force), www.ietf.org/rfc/rfc2911.txt.

IETF (2000b). *Internet Printing Protocol (IPP) : Requirements for Job, Printer, and Device Administrative Operations*. IETF (Internet Enegineering Task Force), www.ietf.org/rfc/rfc3239.txt.

IETF (2001). *Internet Printing Protocol (IPP) : The 'indp' Delivery Method for Event Notifications and Protocol/1.0, draft*. IETF (Internet Enegineering Task Force), www.ietf.org/internet-drafts/draft-ietf-ipp-indp-method-06.txt.

Pyarali, I., O'Ryan, C., and Schmidt, D. (2000). Patterns for efficient, predictable, scalable, and flexible dispatching components. In *Proceedings of 7th Pattern Languages of Programs Conference (PLoP'00)*, Monticello, Illinois, USA.

Schmidt, D., Stal, M., Rohnert, H., and Buschmann, F. (2000). *Pattern-Oriented Distributed System Architectures, 1st edition*. John Wiley & Sons.