1. Introduction

It is well known in the research community, that today’s middleware solutions like the Common Object Request Broker Architecture (CORBA) implementations or Microsoft’s Distributed Component Object Model (DCOM) are not suited for distributed multimedia systems. For example, there is no explicit representation of communication (explicit bindings) and no QoS support as required by multimedia applications. Furthermore, such platforms do not provide the required levels of adaptation and configurability that is needed to accommodate the diversity of modern applications.

In this position paper, we present the architecture of the reflective multimedia object request broker MULTE-ORB, and indicate our ongoing and future research activities.

2. Binding framework

The main programming concepts for multimedia provided by MULTE-ORB are explicit stream bindings, stream interfaces, flows, and associated type checking rules [1]. A binding type identifies the type of stream interfaces which can participate in the binding, the roles they play, and the way behavior at the various stream interfaces are linked. A stream interface consists of a collection of source and/or sink media flows.

Reflection is achieved by reifying the composition of bindings making it available for inspection and adaptation through meta object protocols of the binding and its components (see Figure 1). This is similar to open bindings, as introduced in [3].

- **Binding factory (BF):** a distributed entity responsible for the creation of bindings of a certain type.
- **Binding mutator (BM):** a distributed entity responsible for the reconfiguration of bindings. Reconfiguration actions include adding, removing or replacing components at different points of a binding.
- **Configuration:** the set of components that constitute the internals of binding objects. A configuration is represented by an object graph depicting the identities and types of the binding components, and their interconnections.
- **Binding protocol (BP):** is the protocol used by BFs to create new bindings, and by BMs for reconfiguration.

2.1. Binding protocols

Binding protocols guide the communication and coordination among the BF replicas involved in a binding creation. By defining distinct binding protocols, we define different ways in which bindings may be created, and allow the creation of bindings with different characteristics (e.g., multi-party, with QoS guarantees). In addition, different binding protocols may be optimised for different binding types. Therefore, prior to requesting the creation of a binding, the user may set up policies that configure the BFs so that they behave according to the desired binding protocol.

2.2. Adaptation and reconfiguration through reflection

Adaptability is achieved through use of the meta-object protocol of the binding, which allows any component of the binding to be reified in terms of meta-objects. This meta-object can be used to inspect and manipulate the internal structure of the binding through operations for getting a view of the whole or part of the object graph, as well as to insert, remove or replace individual components. These operations form the building blocks for the complex reconfiguration actions performed by binding mutators.

The resource meta-model defines operations that can be used to adapt resources allocated to components.
Quality of Service (QoS) management is supported as a set of components with given roles such as monitors for monitoring the behaviour of the system and binding controllers that encapsulate (user-defined) policies for reconfiguration and adaptation. Binding mutators are used to implement these policies.

3. Engineering

The engineering of open bindings in MULTE-ORB is based on a flexible protocol framework. A flexible protocol system allows dynamic selection, configuration and reconfiguration of protocol modules to dynamically shape the functionality of a protocol to satisfy specific application requirements and/or adapt to changing service properties of the underlying network. This might even include filter modules to resolve incompatibilities among stream flow endpoints and/or to scale stream flows due to different network technologies in intermediate networks. We use the Da CaPo (Dynamic Configuration of Protocols) system [4] to build the MULTE-ORB [2].

3.1. Overview of Da CaPo

Da CaPo splits communication systems into three layers denoted A, C, and T. End-systems communicate via the transport infrastructure (layer T), representing the available communication infrastructure with end-to-end connectivity. In other words, T services are generic and can be used to establish primitive bindings, like IP or ATM bindings. In layer C, the end-to-end communication support adds functionality to T services such that at the AC-interface, services are provided to run distributed applications (layer A). Layer C is decomposed into protocol functions instead of sublayers. Each protocol function encapsulates a typical fine granular protocol task like error detection, acknowledgment, flow control, encryption/decryption, etc. Protocol functions are implemented by modules. The configuration of modules to a particular protocol is specified by a module graph.

The main focus of the Da CaPo prototype is on the relationship of functionality and QoS of end-to-end protocols as well as the corresponding resource utilization. Applications specify in a service request their QoS requirements in form of an objective function. On the basis of this specification, the most appropriate modules from a functional and resource utilization point of view are selected [4]. Furthermore, it is ensured that sufficient resources are available to support the requested QoS without decreasing the QoS of already established connections (i.e., admission control within layer C).

In order to establish a connection/binding between endpoints and to perform a reconfiguration of an established connection/binding, the module graph is exchanged between the corresponding connection manager instances of Da CaPo.

3.2. Integration of Da CaPo in COOL

We have integrated Da CaPo into the CORBA implementation COOL such that the MULTE-ORB is able to negotiate QoS and that it is able to use optimized protocol configurations instead of TCP/IP.

Our current prototype implementation for Da CaPo in COOL is accompanied with an extended version of IIOP called QoS-IIOP, or QIOP. QIOP encapsulates QoS information from application level IDL interfaces and conveys this information down to the transport layer and performs at the peer system the reverse operation. Da CaPo uses this information for configuration of protocols. This prototype is able to implement binding objects that support QoS and real-time requirements.

3.3. Relationships between Open Bindings and MULTE-ORB engineering

Protocol graphs are abstract descriptions of the functionality of a service and can be directly used to describe a binding type. By instantiating each protocol function with a module, we create a module graph that implements a binding object. The combination of a module graph, the runtime environment, and the signaling protocol corresponds to a binding factory. The module graph corresponds to the meta-object of the binding reifying its internal structure. Table 1 relates the most important concepts of open bindings and MULTE-ORB engineering.

<table>
<thead>
<tr>
<th>Open binding</th>
<th>MULTE-ORB engineering</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Instance of) binding object</td>
<td>Instantiated protocol configuration (module graph)</td>
</tr>
<tr>
<td>Binding factory</td>
<td>Runtime environment, signaling protocol, and module graph</td>
</tr>
<tr>
<td>Binding type</td>
<td>Protocol graph</td>
</tr>
<tr>
<td>Role</td>
<td>Sending/receiving peer</td>
</tr>
<tr>
<td>Interfaces</td>
<td>A modules</td>
</tr>
</tbody>
</table>

4. References


Conference on Configurable Distributed Systems (ICCDS '98), Annapolis, Maryland, USA, May 1998.