Documentation

Software architecture

The background and the main design principles of the DIALOG architecture are described in various scientific articles. From the communication point of view, DIALOG is essentially a peer-to-peer (P2P) architecture. As for other P2P systems, all DIALOG nodes are equal, i.e. they can both send and receive messages. Therefore any node in the network can for instance both ask for information and provide information for the items that it has information about, as shown in Figure 1. Other major design principles adopted from P2P are low installation overhead, equality between parties and scalability.

Figure 1. P2P communication between DIALOG nodes. Any node can both send and receive messages. JDBC: Java Database Connectivity.

The DIALOG node

The main components or objects of a DIALOG node and their functionality are illustrated in Figure 2. The only external function (or method) of a DIALOG node is the "receive" function that takes a message object of type DialogMessage as its only parameter and returns an object of the same type. Three "receiver" types that contain this function exist for the moment, which support different messaging protocols, i.e. SOAP (SoapReceiveImpl), RMI (RMIReceiveImpl) and HTTP/POST messaging (DialogHTMLProductAgent) used by standard CGI- or servlet-type web applications. Which messaging protocol is used depends on how the DIALOG node is started. Any number of messaging rotocols can be running concurrently as long as they can be separated by port number, path names or other standard URL components.

The "ReceiveImpl" object is created at startup. It also has a "receive" function that is called by the "receiver" object when a message is received. If the received message is synchronous (i.e. time-to-live, TTL, is zero), then the message is passed to all "agent" objects that have registered for receiving messages indicated by the "type" field of the message. If the received message is asynchronous, it is buffered in the "receive" buffer and given to the appropriate agent(s) by a thread. Agents can also be described by the notion of "plug-in", i.e. a software component that can be used for extending the functionality of the "core" system at any time. Agent objects are therefore a way of implementing a "plug-in" functionality. For simplicity, we will use the "agent" concept for the rest of this text.

Figure 2. Illustration as UML object diagram of main internal components of a DIALOG node and their functionality.

The DIALOG agent

All processing (or "business logic") is performed by agent objects. By default, agents are initialized with references to an object of type DialogNode (e.g. DServer), that gives agents access to Receivelmpl and Sendimpl objects and a reference to a JDBC database object if a database is configured. This allows agents to easily implement basic functionality, but it does not prevent them from implementing any kind of advanced functionality, such as accessing the file system, using own databases or calling web applications.

A DIALOG agent consists of one or a set of Java classes. In practice, most agents consist of one single class that is a subclass of the abstract class "fi.hut.dialog.agents.DialogAgent". Agent classes usually implement both the server-side logic and a GUI that can be used in "client" applications. This approach has the advantage of combining all the application-specific processing into one single component (i.e. the DIALOG agent) that is independent of the rest of the system. It is also possible to separate "client" and "server" logic if that is desired but the size of a typical DIALOG agent is only 10-20 kbytes (this is the case for currently implemented default agents) so a separation is usually not interesting in practice. The "client" functionality could also be something different than a GUI, e.g. a component that handles RFID-readers or similar devices.

Every agent has a reference to the "Sendimpl" object, whose function "send" is called with a DialogMessage object as parameter. The "send" method also returns a DialogMessage object that either contains the requested information (e.g. for a successful synchronous "read" request) or some other status information about the sending of the message. If the message to send is synchronous, it is sent directly to the appropriate "sender" object for delivery and the result is returned if successful. If the message to send is asynchronous, it is buffered and sent by a sending thread using the appropriate "sender" object. The "sender" object to use is determined either from the type of the message (configurable) or the protocol part of the destination URI/URL.

Adding new Sender/Receiver objects

Other Sender/Receiver interfaces than the standard RMI/SOAP/HTTP can be supported by DIALOG nodes. New Receiver implementations mainly need to wrap incoming messages into a DialogMessage object and then hand it over to the Receivelmpl object, where the actual treatment is taken care of by the appropriate agent depending on the message type. New Sender types can be added and used based on the type of the message being sent, as configured in the file "sendermappings.txt". These receiver and sender mappings make it possible to profit from the default DIALOG messaging, buffering and other functions.