opensphere

COMMUNICATION PROTOCOLS

Index

OPENSHPERE - Developing large projects in distributed environments is never a simple task. Being dependent from other teams makes it hard or sometimes even impossible to develop and test parts of the project under one's responsibility. Opensphere can simulate system components which aren't available yet, allowing to progress with development on schedule and independently from other teams. The built-in testing framework enables executing regular regression test runs making sure the product is thoroughly tested before the delivery.

opensphere

Chapter 1.
Introduction

IT infrastructure usually comprises a number of subsystems each serving a different purpose. Some of these subsystems may have the ability to operate standalone but to use their potential to the fullest they need the ability to communicate with other system components.

Every industry has its own set of protocols which are tailored to very specific requirements and thus have different levels of complexity. Simplest interface which used to be utilized in automotive industry used as little as two wires which surprisingly was enough to implement basic fault tolerance. IT industry is much more demanding when it comes to performance and features.

Strictly regarding protocol, communication is handled in the network layer of the OSI model, and the most popular third layer protocols which fulfill those requirements are TCP/IP stack protocols. Connecting subsystems using CAT5 cables or setting up a wireless connection isn't quite enough to establish communication between system components. This requires implementing means to send and receive data in the application layer.

Such message-based communication enables integration of heterogeneous environments, reduces system bottlenecks and significantly increases scalability. System architects have an option to either use a wide range of available standardized message exchange protocols or develop their own, proprietary solution. Designing our own protocol is a challenging task as it requires not only modeling messages exchange but also addressing potential performance issues. Drawbacks of the design aren't always clear at first and they may become apparent in future after it is integrated in existing IT environment. Having an in-house developed interface also requires providing a detailed interface specification. That's why it's more reasonable to use a popular, commonly used protocol rather than develop our own solution. Unless of course there is no solution available which addresses all our needs.

opensphere

Chapter 2.
Software components message exchange – JMS and Tibco Rendezvous

Enterprise messaging solutions use lightweight entities consisting of header and body to enable communication between different system components. The header contains information necessary for routing and identification while the body contains the data being exchanged between the applications. Java Message Service is an API for accessing enterprise message exchange systems from applications written in Java. Basically it's a set of interfaces and associated semantics enabling communication in a distributed heterogeneous system.

JMS implements two currently dominant messaging styles: point-to-point (PTP) and publish-and-subscribe (Pub/Sub). Both can be used in the same application but since each requires a separate domain and customized sets of interfaces, it's wise to use either one or the other.

The PTP model uses «message queues» which are created administratively according to the current needs. Since queue management requires resources, it is not uncommon to have only one message queue handling all the communication. Clients have the ability to send and receive messages to queue and browse messages in the queue without removing them. A message queue object is usually created by the administrator and it's always available to store the data whether the consumer client is active or not. This way the client doesn't have to implement logic ensuring that the messages which have been addressed to it weren't deleted while he was inactive.

The Pub/Sub model introduces «topic» which are responsible for gathering and distributing messages. This model differentiates between two types of subscribers: non-durable, in case of which messages addressed to it can be deleted without being delivered when the client is inactive; and durable which ensure that when inactive the messages are kept in the «topic» and can be collected once the client becomes active.

JMS applications are platform independent which means that they can run on any hardware and operating system supporting java.

Tibco Rendezvous is a similar messaging solution implementing the same models. The main difference between JMS and Tibco is that the latter is available not only for Java but also for C, C++, C# and Perl, which makes it implementable in pretty much any environment.

The other difference is that Tibco is a complete and mature solution which offers quality of service and, depending on the need, it ensures reliable or guaranteed delivery. In order to eliminate bottlenecks

opensphere

and single point of failure, Tibco utilizes distributed daemon-based peer-to-peer architecture. Companies have an option to use external daemons to manage messages exchange or implement messages management directly in applications. The latter requires more effort but in return it gives the option for ultra low-latency. This makes Tibco a very good solution in mission-critical systems.

Apart from the Pub/Sub and PTP models, Tibco offers request/ reply model making it more universal.

Chapter 3.
Communication over the Internet – Simple Object Access Protocol (SOAP)

SOAP enables messages exchange between applications, which is typically realized with Remote Procedures Calls (RPC), over the HTTP protocol. The main advantage of the SOAP over the RPC is that the HTTP traffic generated by the first one is something acceptable in pretty much every IT environment while the traffic generated by the latter will most likely be blocked by the firewall or proxy server. This makes implementation of SOAP-based communication much easier compared to the network reconfiguration hassle required by CORBA or a similar middleware protocol. While the HTTP protocol is most commonly used for transferring SOAP messages, these can also be transferred over any other transport protocol such as SMTP, TCP or JMS.

A SOAP message is nothing more than an XML document which consists of:

- an Envelope which is a root XML element identifying given XML document as a SOAP message,
- a Header which contains application specific data such as authentication or payment information,
- a Body containing the actual content exchanged between applications,
- an optional fault element containing errors and status information.

Implementing SOAP either as an internal part of the application or as a web service is much less complicated than any other middleware message exchange protocol, but it has its flaws. Messages expressed in a verbose XML format can be quite large and parsing, and processing such documents requires resources and time. This makes SOAP considerably slower compared to competitive middleware protocols.

SOAP protocol is considered a standard for Web services development thus many vendors provide SOAP APIs which allow connecting their products with system components delivered by other third party solutions providers.

opensphere

Chapter 4.
Interfacing databases – Java Database Connectivity (JDBC)

JDBC is an API for Java applications which enables querying and updating data stored in SQL relational databases or other tabular data sources such as spreadsheets or flat files. JDBC allows for executing not only CREATE, UPDATE, INSERT, DELETE and SELECT SQL statements but also invoking stored procedures through a JDBC connection. INSERT, UPDATE and DELETE statements do not return any data other than information on how many rows were affected after executing the query. SELECT statement returns a JDBC row result set with data retrieved from the database and JDBC API provides methods allowing for browsing the results set.

SQL statements may be sent to the database server and executed right away or can be cached for increased efficiency (batch updates). Either way the SQL requests from the Java program are converted to a protocol that is understood by the database server. This conversion is handled by the JDBC driver specifically for the given type of the DBMS (Database Management System). This way an application which was natively written to connect to a MySQL server can be easily adapted to communicate with a PostgreSQL database as it's only a matter of using a different JDBC driver.

JDBC API provides a JDBC Driver Manager which paired with the mechanism for dynamic loading of Java packages allows for accessing different types of database servers from the same application. Thanks to a JDBC-to-ODBC bridge, JDBC API also supports connections to ODBC (Open Database Connectivity) data sources making it a complete solution. That and a very strong support from a vast number of vendors is what makes the JDBC an industry standard.

opensphere

Chapter 5.
Conclusions

Given the variety of communication protocols, system integration is a very challenging task. Making sure that each and every component operates and communicates with other parts of the system according to the specification requires a lot of attention to detail. System integrators crave tools which would make their work easier.

centeractive's Opensphere helps modeling communication between system components during the development phase and supports integration specialists and system testers throughout the integration process.

The built-in Message Detector allows the identification of given types of Tibco Rendezvous or JMS messages exchanged between the system's components. Detected messages can be easily filtered, edited, stored and re-sent allowing the system's communication behavior to be thoroughly tested and identifying potential problems before the system goes live.

Opensphere can also simulate any type of system objects which comes in very handy during development of components which are supposed to interact with other components that aren't available yet. This allows the development of components independently according to the contracted schedule. Add to that Opensphere's automated testing framework and you have a complete solution for system integration.

opensphere