Using JAVA to Provide a Reliable Distributed WEB

András Benczúr and Balázs Csizmazia

Large computer networks like the Internet are heterogeneous with many diverse hardware components running a more diverse spectrum of system software and user-level applications. It is partly because many organizations make their decisions on the basis of the costs of the full solutions of their problems without giving much attention on factors e.g. who provides the solutions or what kind of references has the vendor providing the solution. Another cause of this heterogeneity is the fact that already purchased systems may be too costly to replace, and so they are not likely to be replaced even if they are not based on up-to-date technologies. Of course this heterogeneity has its advantages, too: it allows to utilize the best technologies for each given problem field. Other characteristics of such large networks are that the underlying network protocols (like the TCP/IP, OSI families of protocols) are unreliable, and they cannot tolerate any partial or total failures of the utilized network media. Nowadays there are several distributed applications and there are even more applications under construction needing fault-tolerance, so that these applications can be relied upon in mission-critical settings too, even if they are built on non fault-tolerant networking technologies. Such are application areas e.g. for medicine, aircraft control, or trading applications built entirely on Internet technologies, and needing predictable behavior despite system failures (we have in mind here mainly TCP/IP-based network technologies with so-called halting failures where a component simply stops working, or a network link breaks between active network components). A nice example of this kind of setting is the WWW, which is used worldwide for accessing information and utilizes many components that can fail independently. Another WWW-related important concept is Java a new tool of distributed application development for the WWW by making the client side of the WEB more interactive by a code written in a portable high-level programming language. But it lacks a distributed object infrastructure, like the Arjuna or the COSS/TS service.

In this paper we describe a new tool providing a reusable reliable distributed infrastructure upon which reliable web-applications can be built easily. The tool is constructed in an object-oriented manner allowing an application to bracket a series of method invocations by begin/end transaction markers. If some of the method invocations fail during a transaction, the transaction manager tool will roll it back by undoing the effects of all previous method calls of the transaction. This tool is implemented in the Java language and allows an easy integration of the distributed Java objects implementing a transaction participation marker interface (also non-Java objects can be involved in transactions by providing a simple Java wrapper or other gateways).

Our tool uses the generally available Java remote object access facilities leveraging the programmer from the error-prone task of writing communication codes. In this paper we present its implementation utilizing distributed transaction processing protocols, and compare two well-known distributed transaction manager protocols, the 2PC and 3PC ones. Although 3PC is a non-blocking protocol, it doesn´t tolerate network link break failures typical for present day Internet technologies. This non-blocking protocol is more complex to program (the complexity arises from the fact that the 3PC protocol requires a fault-tolerant coordinator election protocol whose implementation is not straightforward on asynchronous network technologies, like the Internet). The tool provides, among the transaction manager service, a simple concurrency control service that can be used to implement distributed resource locking (also missing from the Java programmers tools).

Finally we are going to discuss the 3PC protocol´s applicability in faulty asynchronous networks, and to identify some areas where its added complexity can be tolerated, because of its additional non-blocking capabilities, and we will compare it with other possible solutions to this problem, focusing on their applicability on the WEB.