Time Independent Invocation in Java CMS[13]

Attila Ulbert and Markus Hof

Distributed object middleware, e.g. CORBA and Java RMI, hides the network details required in order to access remote objects and to invoke their methods. It offers an easy to use framework to develop distributed object oriented applications. However, many applications require - to some extent - access to some of these details. Many of these requirements can be satisfied with the help of appropriate invocation semantics. Therefore, many middleware platforms extend their definitions to include additional semantics, e.g. interceptors in COOL or filters in Orbix. This lead also to modifications to standards as, e.g. CORBA with its messaging service, Enterprise Java Beans with transactional semantics.

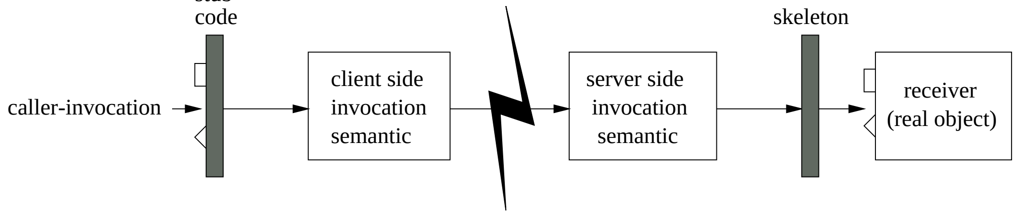

Figure 10: Client and server-side semantics

In our project we added a semantic for time independent invocations (TII) to the composable message semantics (CMS) framework. The CMS framework offers an alternative to Java RMI as a platform for remote method invocation. It allows the creation and composition of arbitrary new invocation semantics, e.g. replication, synchronization, asynchronous as well as arbitrary compositions of them. An invocation semantic consists of a server and a client part (see Figure 10). Both of these again are composed of smaller building blocks that are combined with the Decorator pattern. Every object can have its individual set of semantics. It is even possible to define more than one set of semantics for one and the same object. This allows the definition of multiple views on an object.

Other systems already define time independent invocations, e.g. the CORBA messaging service. A time independent invocation can be viewed as a fault-tolerant asynchronous invocation that is able to receive the results of the execution of the remote method at an arbitrary later point in time. This technique can help to the design and may simplify the implementation.

We have specified the implemented invocation semantics formally, as the informal specifications are often ambiguous. Our formal TII description makes the specification unambiguous and eases the design of the implementation. The complete implementation can be achieved by extending a simplified TII called oneway-TII (a fault-tolerant asynchronous invocation with parameters specifying the level of fault-tolerance).

With our CMS framework it is quite simple to use new invocation semantics. After the server instantiates and exports the object with the assigned server-side semantics, the client imports the object and assigns the desired client-side semantics. Afterwards, the methods of the imported object can be invoked as if they were methods of a local object. The following example depicts the assignment of a TII semantic to the method "set" and an invocation of that method that uses the newly assigned semantics.

Whenever the method "set" is called, the system transparently applies the assigned semantics and guarantees a fault-tolerant behaviour on the level specified by the two parameters of the TIInvocation constructor. In order to use the complete TII, we have to either supply a call-back method or a polling object. With the help of one of these two techniques we can access the results of the remote method invocation and can verify whether the remote invocation was successful or not.

```
Test testObj = new Test();

ClassInfo ci  = new ClassInfo(testObj);  // get Metainformation of class Test
```

[13]This research work was supported by Grant Nr. FKFP 0206/97

```
MethodInfo mi = ci.getMethod("set");
CallerInvocation inv = new TIInvocation(23, 100); // semantic with 23 retries
                                                  // every 100 milliseconds
mi.setCallerInvocation(inv);  // assign semantic to method "set"
testObj = (Test)Remote.get(adr, ci, "testObj");

testObj.set(....); // call methods as usual
```