Centralisation of replicas responses for exception concertation in Dimaxx
ANR FACOMA - Technical note - January 2010

Christophe Dony, Selma Kchir, M. El Jouhari, Chouki Tibermacine
LIRMM - CNRS UMR 5506 - Univ. Montpellier II - Montpellier (France)
{dony, tibermacin}@lirmm.fr, selma_kchir@hotmail.com

Christelle Urtado, Sylvain Vauttier
LGI2P - Ecole des Mines d'Alès - Nîmes (France)
{Christelle.Urtado, Sylvain.Vauttier}@mines-ales.fr

February 4, 2010

1 Introduction

This technical note describes the technical solution we propose, within the DARX middleware, to centralise responses to requests computed by replicas of a given replicated agent within that agent context and before any response is returned to the client. This solution will allow for (1) trapping any responses coming from replicas including exceptions they could throw and (2) for concerting them. Concertation will be the next stage of the implementation, it will for example consist in either transmitting a normal response to the client or in throwing a concerted exception or in waiting for responses for other replicas, or in passive replication mode to elect a new leader and to retry the request after the current one has signaled a replica specific exception.

2 Analysis of the DARX framework : Replicas responses to clients

When a message is sent by an agent A to a replicated agent B, the `ActiveReplicationStrategy` or the `PassiveReplicationStrategy` object that rules replication for agent B (and is associated to the leader for agent B) treats the received message object. Messages are either treated by the *deliverAsyncMessage()* method or by the *deliverSyncMessage()* method, depending on the chosen communication mechanism.

In case of synchronous message invocation, the *deliverSyncMessage()* method associated to agent B collects the results that are produced by the treatment of the message for all replicas of the replication group. These return values might be normal return values or exceptions, indifferenty. This collection process acts as a synchronization point. After all return values have been locally collected, the return value for the message treatment from the leader of agent B is returned to the message-sender A (the agent who requested the treatment of the message). This return value results from a decision by the leader of agent B. In case of synchronous message invocation, we could then easilly extend the `ActiveReplicationStrategy` class to manage exceptions. Unfortunately, Dimax agents do not use these synchronous communication features.

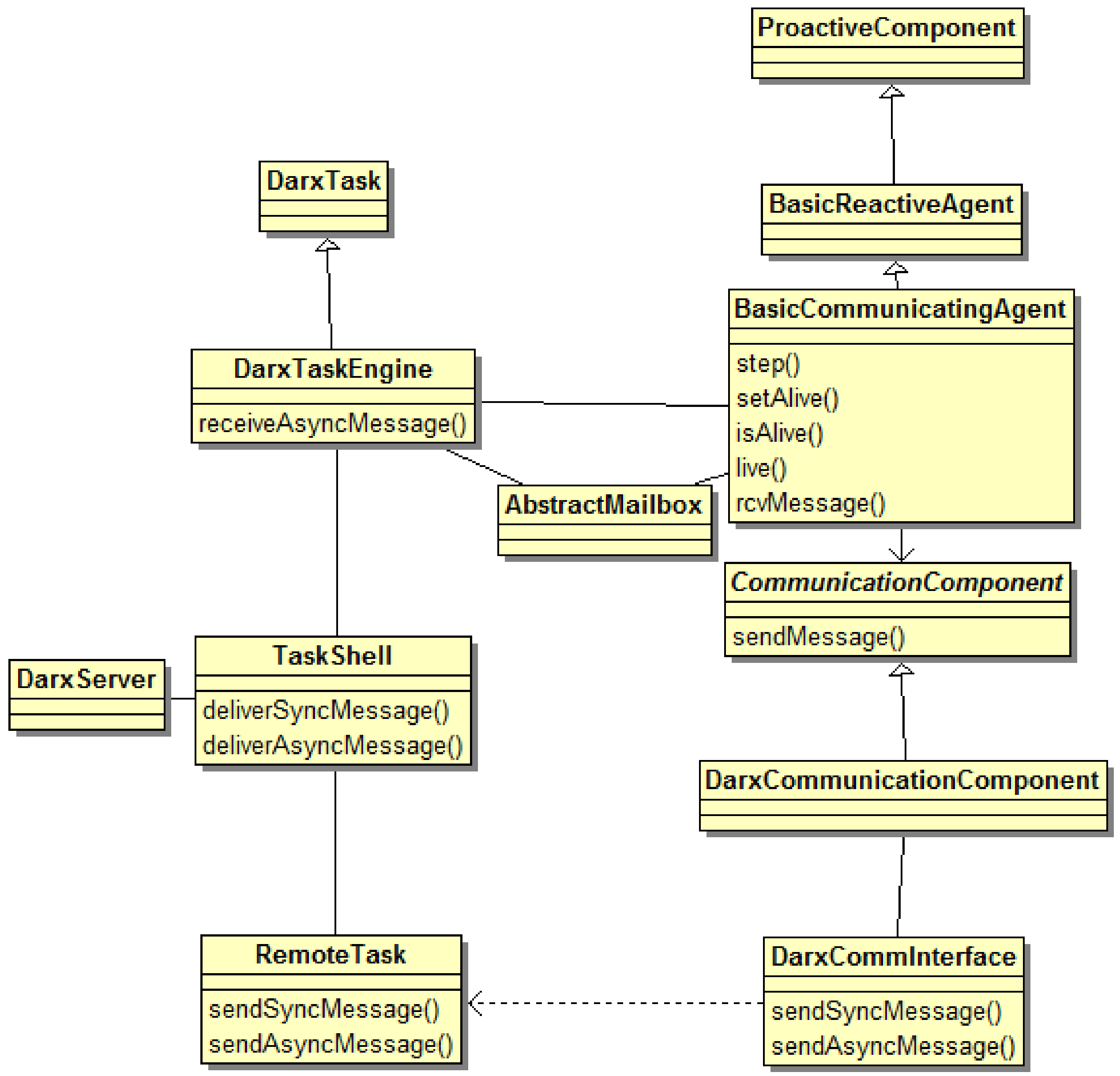

Figure 1: Darx Class diagram

Let us now have a look at the equivalent mechanism for asynchronous messages. As for synchronous messages, the *deliverAsyncMessage()* method broadcasts the received message object to the replicas of agent B (locally to the replication group). What strongly differs is that the *deliverAsyncMessage()* method does not collect responses. Each replica directly sends its response to the message-sender. This response mechanism is both asynchronous and concurrent. As a consequence, the message-sending agent A receives redundant response messages that come from the various replicas of agent B. The message-sending agent has a filtering mechanisms that enables to handle such situations. To do so, messages are identified with serial numbers. As all replicas of a given agent have the same behaviour, they all send identical messages. This means that agent A will receive several messages from the replicas of agent B that all have the same serial number. Darx, as a middleware, is then able to memorize the serial numbers of all the messages it receives to detect redundant (or obsolete) messages and ignore them (*acceptMsg()* method of the *TaskShell* class).

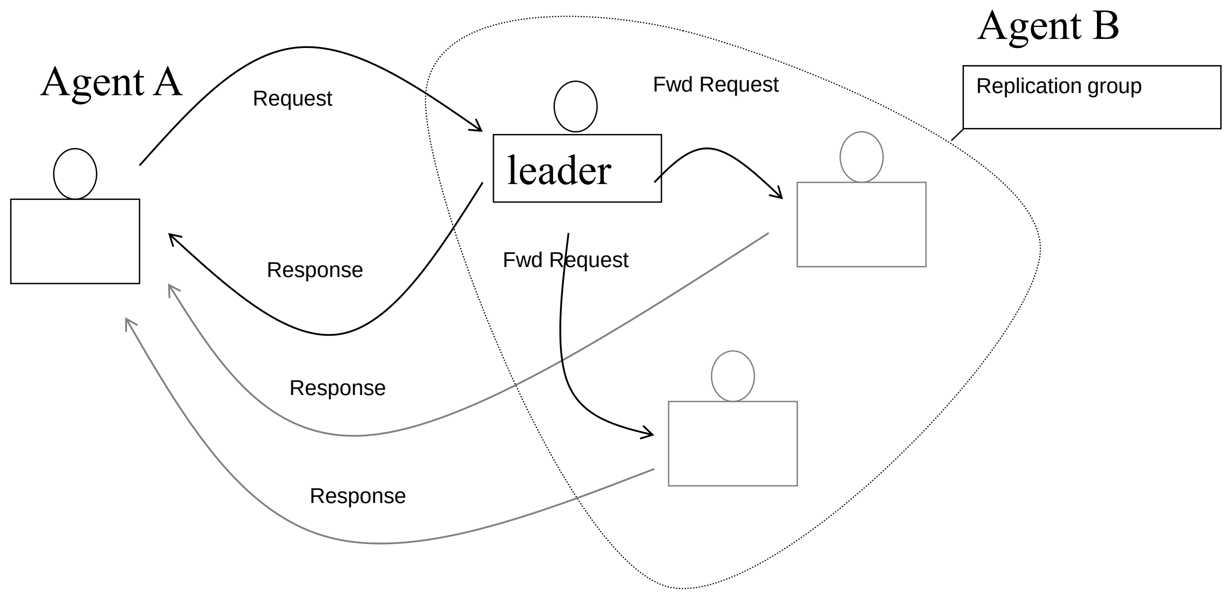

Figure 2: Overview of the DARX message-sending mechanism

To sum up, the DARX replicated middleware initially (see Fig. 2) ensures that response messages from replicas, that are all sent back to the client agent, are filtered at their entry into the client message box.

The fact that the collection and treatment of response messages is assumed by the message-sender is not suitable for exception handling (as it would neither be for a more cognitive treatment of responses in general). Indeed, such message-sender localization implies that a part of agent B's behavior is contained in this filtering mechanism located in agent A. Such externalization of agent's behavior violates the agent autonomy principle and is not compatible with local exception handling decision-making. Such decision-making relative to exception management (such as exception concertation, for example) is considered to be a part of an agent's behavior and thus should be localized into the agent. To overcome this problem, we propose to handle responses that come from replicas in the message-receiver. This will make the message-receiver responsible of all its behaviour.

3 A solution to trap replicas responses at the leader level

If we want to achieve responses handling at the server side (at the level of the agent who computes the reponses), it is necessary to add a response management mechanism for asynchronous messages in the class `ActiveReplicationStrategy`. It is then necessary to generally distinguish request messages from response messages (they are indistinct in Darx) by creating explicit subclasses of `MessageXX` that we call `SageRequestMessage` and `SageResponseMessage` as shown in figure 3 .

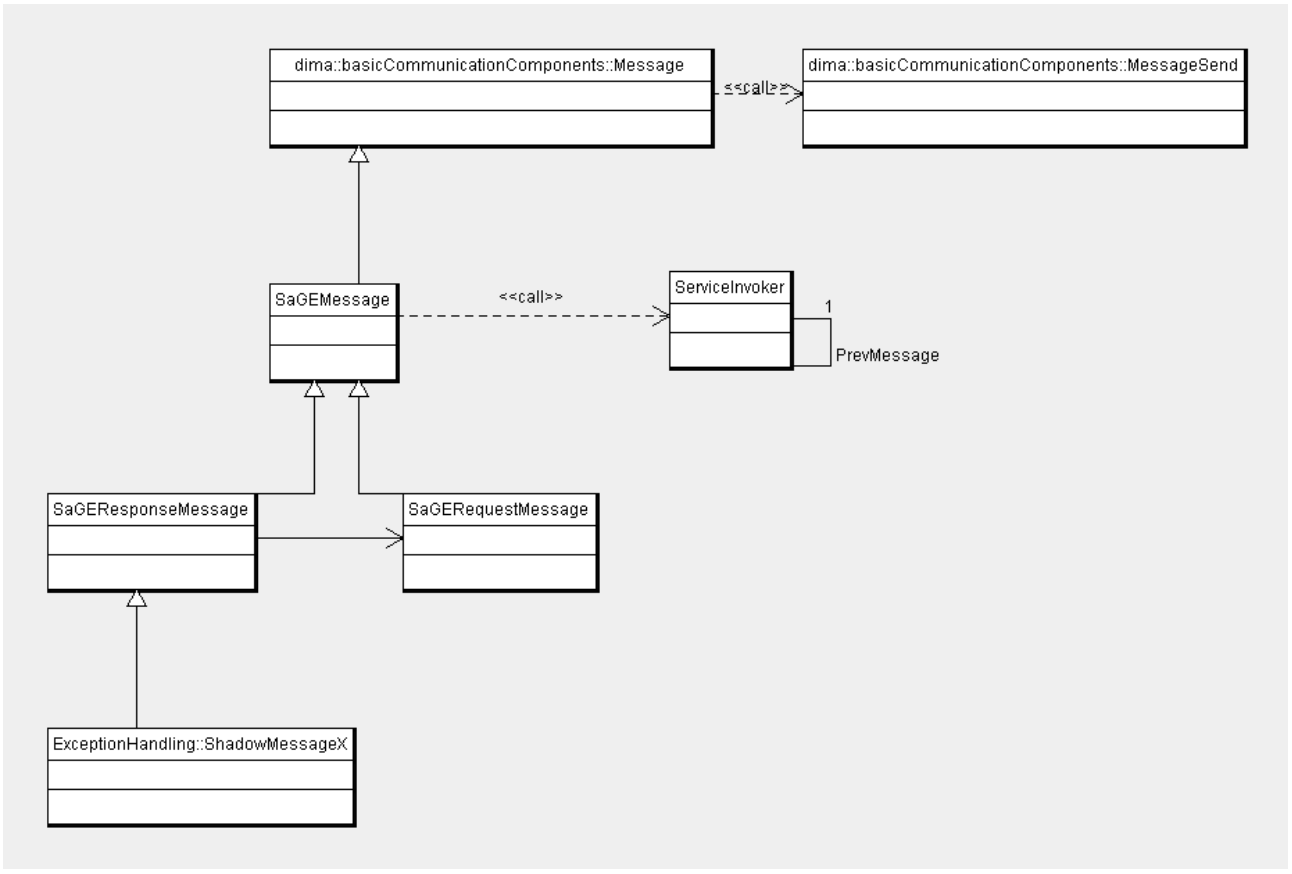
dima::basicCommunicationComponents::Message
<<call>>
dima::basicCommunicationComponents::MessageSend
SaGEMessage
<<call>>
ServiceInvoker
1
PrevMessage
SaGEResponseMessage
SaGERequestMessage
ExceptionHandling::ShadowMessageX

Figure 3: Overview of the modified message sending mechanism

Finally, it is necessary to specialize the Darx message sending primitive so that it processes specifically the response messages. We thus have defined `DARXCommunicationComponentX`, a subclass of `DARXCommunicationComponent`, and have redefined the method *sendMessage()* so that messages are handled as follows (see Fig. 1).

- If the message is a request, it is sent directly.
- If the message is a response, two cases are possible :
  - If the response is sent by a replica that is not the leader, the response must be delegated to the leader to allow its monitoring and/or treatment.

    The simplest solution to achieve that delegation is for the replica to send its response message to its leader instead of to its client. This message will be received and handled by the leader. To implement this, we propose a new kind of response messages, called *shadow messages* , instances of a new subclass of `SageResponseMessage` called `ShadowResponseMessageX`. Shadow messages are transmitted to the leader of the replication group and encapsulate the response message sent by the replica.

    To summarize and in other words, when a replica that is not the leader wants to send a response, the response is encapsuled into a `ShadowResponseMessageX` containing the replica address, which is in turn sent to the leader in a standard way. The message is thus intercepted by DARX, at the leader level, via the method *sendMessage()* of the class `DarxCommunicationComponentX` The `shadow response message` will then be handled in the method *deliverAsyncMessage()* of `ActiveReplicationStrategy` of the leader. At this point we introduce a method called `handleAsyncResponse` to handle all cases of synchronous responses including exceptions.

– If the response is sent by the leader, it will directly invoke the method *handleAsyncResponse()* of the `ActiveReplicationStrategy` associated to the leader and the same *handleAsyncResponse* method will be invoked.

The method *handleAsyncResponse()* concretely implements our mechanism for exception handling. All responses computed by replicas of an agent, either normal responses or exceptions (an exception is a particular case of response), are sent back to their leader agent as shadow messages. The leader is then able to record, filter, dismiss or concert responses so as to send a unique response to the client (if needed). Such a capability prepares the leader to being active in managing the exception handling issues for the whole replication group.

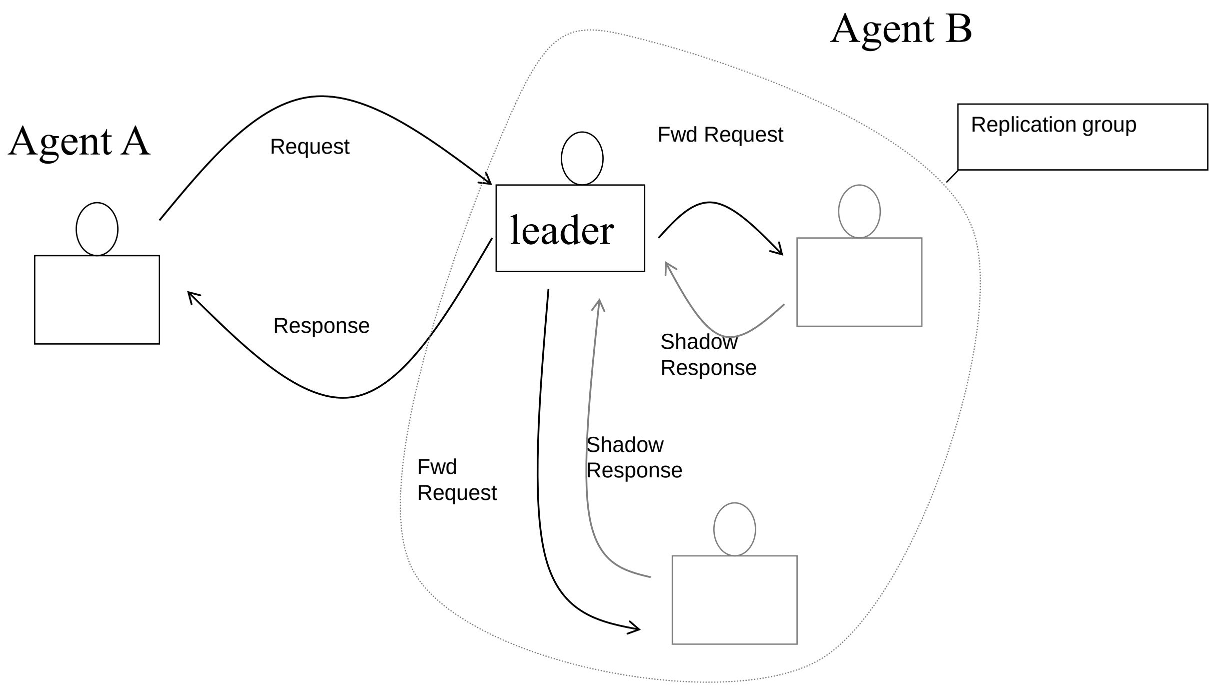

Figure 4: Overview of the modified message sending mechanism

4 Exceptional Response in presence of replication

```
public void HandlerSearch(String serviceName, Exception e){
  /* local search */

  Method[] meths = this.getClass().getMethods();
  for (Method m : meths){
    if (m.getAnnotation(serviceHandler.class) != null)
    { if (m.getAnnotation(serviceHandler.class).serviceName().toString().equals(serviceName))
        if (m.getParameterTypes()[0].equals(e.getClass())) {
          //Service handler found
          runHandler(this, m, e);
          return;}
    }
  }

  for (Method m : meths){
    if (m.getAnnotation(agentHandler.class) != null)
    { if (m.getParameterTypes()[0].equals(e.getClass())) {
        //Agent handler found
        runHandler(this, m, e);
        return;}
    }
  }

  //delegating the exception to the replication manager
  sendExceptionMessage(serviceName, e);
}
```

Local search part of handler search including delegation to the replication manager.

This section explains how exceptions are signaled within replicated agents. More precisely it explains how, if no handler can be found at the replica's level, the signaling process transfers the control to the leader for concertation. The Figure 5), proposes an activity diagram that synthezizes the solution we propose.

Our solution is embedded in the global solution for signaling that we briefly recall here and which is partially illustrated by the above code listing. When an exception is signaled, within a service of an agent, the execution of that service is suspended and handler is search, first locally in the agent context (what we have called *localSearch(e)*) and if no handler is found there, the signaling process, or handler search process, is asynchronously delegated at the caller level (what we have called *callerSearch*). The caller, or client of the current agent, is the agent that has sent the request that has entailed the execution of the signaling service. If a handler is found, it is executed and its execution definitely terminates the faulty service, and more generally the tree of services that have been interrupted by the signaling mechanism; we apply a termination model of exception handling.

The new point for exception signaling, in presence of replication, is that before initiating the caller's search, the control should be transfered to the replication manager for exception concertation. As for standard responses coming from replicas, transmitting exception signaling (exceptional responses) from one replica to the replication manager will be achieved via the DARX middleware, by sending a message to the object representing the current replication strategy.

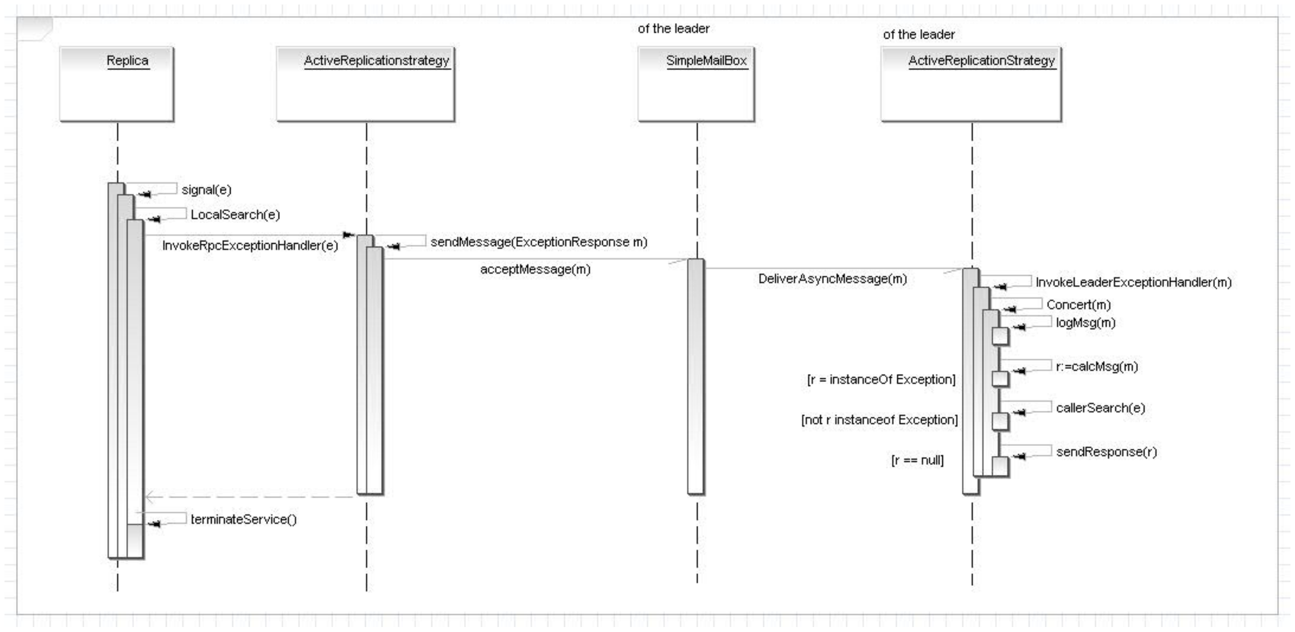

Figure 5: Exceptional response sequence diagram

It the faulty replica is not the leader of the group, the message *InvokeReplicationExceptionHandler(e)* is sent (see Figure 5), and in the corresponding method, the signaling is propagated to the leader: a message of kind *ExceptionResponse* is placed in the mailbox of the leader (via the method *AcceptMsg(DarxMessage m)* of the class `TaskShell`, this method checks the validity of a received message and if accepted, its serial number is recorded for later checks). The message is then sent to the leader using the method *DeliverAsyncMessage(m)* of the class *taskShell*. Finally the object representing the replication strategy of the leader is invoked via the method *InvokeLeaderExceptionHandler(m)*. This method is in fact called each time a normal/exceptional response is received and is responsible for organising the concertation thanks to the replication manager resolution (or concertation) function.

The resolution function of the leader is itself executed each time an exception handler is searched for; let us recall some basic possible cases (mode advances cases have been sketched in [1]).

- The resolution function returns a normal response.
- All replicas have signaled the same exception. The resolution function propagates this exception to the client. It is considered that this is the concerted response for the replica's group, the execution of the requested service of all replicas is stopped. "Caller search" part of the signaling algorithm (see [1] is executed.
- Most of other replicas do not have answered yet. The exception is logged, the resolution function returns null and the handler search stops.

References

[1] Christophe Dony, Chouki Tibermacine, Christelle Urtado, and Sylvain Vauttier. Specification of an exception handling system for a replicated agent environment. In WEH '08: Proceedings of the 4th international workshop on Exception handling - Atlanta, Georgia , pages 24–31. ACM Digital Library, nov. 2008.