AN ARCHITECTURE FOR FAULT-TOLERANT HLA-BASED SIMULATION

Clemens Berchtold* Marco Hezel

*ITIS e.V., email: berch@informatik.unibw-muenchen.de
Institut für Technische Informatik
Universität der Bundeswehr München, D-85577 Neubiberg, Germany

Abstract

In this paper we present a new federate concept to support fault tolerance (FT) in HLA based simulations, the so-called R-Fed (Replica Federate). The R-Fed may serve as a concept for an architecture for a FT HLA-based simulation. The FT mechanism inside the R-Fed is based on replication. The R-Fed contains two parts, the FT components and the replicas, which are the replicated original HLA federates. The FT components form the FT mechanism and are described in detail. The replicas need modifications so that the messages are not directly sent to the RTI but go a roundabout way over the FT mechanism instead. For this purpose new Federate Ambassadors and a new RTI Ambassador (FTAmb-RTI) are integrated. Furthermore FT methods appropriate for the R-Fed are proposed. Finally some scenarios of a FT simulation run are explained to illustrate the entire concept of the R-Fed.

Keywords: Distributed simulation, fault tolerance, High Level Architecture (HLA).

1 INTRODUCTION

The High Level Architecture (HLA) (see e.g. [DMSO, Kuhl et al. 2000]) is defined as the standard for distributed military simulation. But also outside military simulation the HLA has become popular. In consequence the HLA is used as well in safety critical simulations (military simulations) as in time and cost intensive simulations (simulations in industry and scientific simulations). A failure or crash of one single component of the distributed simulation lead to the loss of the overall function of the whole simulation system. Therefore mechanisms for fault tolerance are important for increasing safety or decreasing costs. But currently, the HLA does not include a formal failure model [Dahmann 1999].

In this work we propose an architecture for HLA-based simulations which is able to tolerate not only *fail-stop* but also so-called *Byzantine failures*. In a fail-stop failure a process crashes, but before it produces correct results. A situation in which incorrect results are produced is called a Byzantine failure. In our architecture the fault tolerance mechanism for Byzantine failures is based on replication. The replicas are compared and if there is a sufficient great difference between the replicas a Byzantine failure is detected. The replicas may be different implementations of the federate. For example a wrong model implementation can be detected in this way. Thus in addition this architecture may be used for verification of simulation models too.

In the literature there are only a few related works concerning fault tolerance and distributed simulation. A structured view of fault tolerance in parallel and distributed simulation and possible solution approaches are given in [Lüthi and Berchtold 2000]. *Fail-safe PVM*, an enhancement to PVM (Parallel Virtual Machine) with regard to fault tolerance, is reported in [León et al. 1993]. In [OMG 2000] the fault tolerant CORBA (Common Object Request Broker Architecture) specification is described.

The paper is organized as follows. Section 2 characterizes the R-Fed, our concept for the architecture for FT HLA-based simulation. In Section 3 we pick out FT methods that may be appropriate for the R-Fed. Scenarios of a FT simulation run to illustrate the function of the R-Fed are depicted in Section 4. And finally conclusions are drawn in section 5.

2 ARCHITECTURE

We developed a new federate concept for FT, the *R-Fed* (*Replica Federate*). The architecture of the R-Fed is shown in figure 1. A FT mechanism based on replication is integrated into the R-Fed. The R-Fed consists of the two main parts

- Replicas
- FT components

The replicas (Rep_1, Rep_2,..., Rep_n) are the (various) implementations of the original federate and may be located on different nodes in the distributed simulation environment. The FT components consist of the FT Manager, the Property Manager, the Compare Unit and the Fault Detector and have to be implemented on one single node. We are now going to

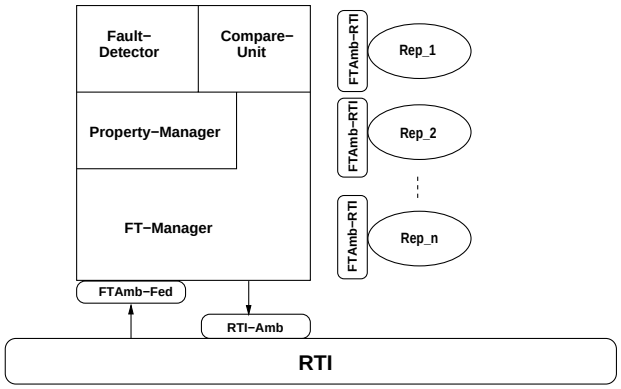

Figure 1: R-Fed

explain these two main parts of the R-Fed in more detail.

2.1 Replica

The modified federate called replica and its modifications are drawn in figure 2. Besides the standard federate implementation and the Federate Ambassador, save/restore functions and the FTAmb-RTI (see later) are added. The replica must be able to save and restore its simulation state. For this purpose commands similiar to the standard HLA commands `federate_save()` and `federate_restore()` may be used.

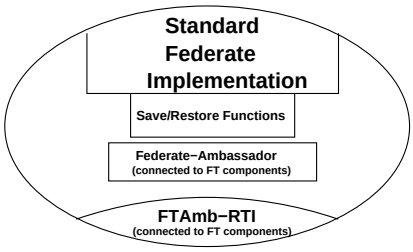

Figure 2: Replica

FTAmb-RTI In a normal HLA simulation the federates send the results (attribute changes, interactions) directly to the RTI via the RTI-Ambassador. The RTI services are called up by the federates via the RTI-Ambassador. In our R-Fed concept the results of the replicas have to be sent to the FT mechanism before. But this needs a modified implementation of the RTI-Ambassador for the replica. This new modified implementation is an interface to the FT mechanism and we named it *FTAmb-RTI*. The FTAmb-RTI has the following functions:

- Send results of the replicas to the FT Manager
- Receive and forward function calls
- Wait for error messages from the RTI

As the pendant to the FTAmb-RTI a new Federate Ambassador (*FTAmb-Fed*) is implemented in the R-Fed to forward the messages of the RTI to the FT Manager.

Note: The communication between the FT Manager and its replicas must not be done by the means of the RTI, but may be done directly by the TCP/IP protocol for example. Therefore a replica has to know how to reach its FT-Manager, i.e. replicas need the IP address and the port where the communication takes place. This information is transmitted through the FTAmb-RTI too.

2.2 FT Components

The fault tolerance components form the fault tolerance mechanism of the R-Fed. As mentioned above these components have to lie on one single node. In this subsection we describe the components and list their functions.

Compare Unit The Compare Unit is generally responsible for the comparison and documentation of the messages of the replicas. The Compare Unit receives directly from the FT Manager the messages and the results of the replicas by the means of the implementation of the FTAmb-RTI. Then the functions in detail are

- Write results in an internal data structure
- Watche time outs
- Compare results of the replicas
- Call the Fault Detector

The time outs criteria are defined by the time intervals within the replicas have to answer to the broadcast of the FT Manager. The time outs criteria are determined by the Property Manager. The Compare Unit gets the value of the time interval from the Property Manager and registers if the message of one replica is within or out of this time limit. This registration is written in the internal data structure. The results of the replicas are compared bit-wise (for example). Again, this comparison is put into the internal data structure, which is then given to the Fault Detector.

Fault Detector The Compare Unit calls the Fault Detector by sending the comparative results. Then the Fault Detector makes decisions about

- Which replicas are correct or faulty
- Further FT procedure

Finally the decision is sent to the FT Manager. A restart of a replica is an example of a further FT procedure. The rules for this decision making are looked up in the configuration file of the Property Manager

(see next paragraph). An example for such a rule may be: (Say) m replicas of the (say) n replicas ($m \leq n$) have to be identical to declare that the m replicas are correct.

Property Manager The parameters characterizing the FT mechanism are determined by the Property Manager. These parameters are stored in a configuration file. The configuration file contains among others

- Number of the replicas to be executed
- Number of the various implementations
- Failure criteria and FT methods
- How many replicas have to be restarted after a failure
- Where are the checkpoints put

Various implementations of the federate are needed if the R-Fed is used for verification of the simulation system. The information which replica represents which implementation is also part of the configuration file of the Property Manager.

Additionally the Property Manager is responsible for the determination of the time outs, i.e. the Property Manager calculates the time interval within the results have to be sent by the replicas. A method for the calculation of this time interval can be found in [Echtle 1990].

FT Manager The FT Manager is the central unit in the R-Fed that controls the FT mechanism. The main functions of the FT Manager are

- Intermediate station between RTI and replicas
- Contain information about the replicas
- Ability to communicate with other FT Managers

The Fault Detector transmits the correct results of the replicas and the information which replica is correct to the FT Manager. The FT Manager forwards these correct results to the RTI. On the other hand the FT Manager broadcasts the messages of the RTI to all replicas. For that function the FT Manager needs the following informations about the replicas

- Number of replicas
- PID, node, IP address, port of the replica for communication

Furthermore the FT Manager is capable to (1) start, (2) terminate, (3) save and (4) restore the replicas on its node and to (5) migrate a replica to another node. As mentioned earlier the replicas may be located on nodes different to the "FT Manager's node". Therefore the FT Manager must be able to start, terminate, save and restore its replicas on a node of other FT Managers.

3 FT METHODS

In this section we give a short overview of FT methods appropriate for the concept of R-Fed. We distinguish between FT methods for replica failure and for FT Manager failure. The replicas may crash or produce incorrect results whereas the FT Manager may only crash. In consequence the FT methods are different. The mechanism to tolerate replica failures is based on replication and the mechanism to react on crashes of a FT Manager may base on checkpointing.

3.1 Failure of Replica

First we examine the possible FT methods to react on Byzantine Failures, i.e. some of the components in the distributed system produce incorrect results. An appropriate FT mechanism is the replication of the federates resulting in a number of replicas. The problem to decide which replicas are "good" or "bad" is generally known as *the Byzantine Generals Problem* [Lamport et al. 1982]. In the R-Fed the good replicas may be identified by *majority voting*. So if at least half of the replicas' results are identical they are assumed to be correct. It has to be mentioned that at least $2N+1$ good replicas can tolerate N bad replicas.

Methods of replication can be divided into two main schemes, the passive and the active scheme (see for example [Défago et al. 1998]). In the passive variation a primary replica sends periodically updates to the backup replicas. In the active scheme all replicas act concurrently. Thus for the R-Fed concept only the active scheme is relevant.

To detect crashes of the replicas a method of time outs may be applied. If a replica does not answer within a predefined time interval it is declared to be dead.

3.2 Failure of FT Manager

FT Managers only commit fail-stop failures, e.g. if the node where the FT Manager lies crashes. In [Becker 1991] crash detection techniques are described where the processes are kept under surveillance. A process is chosen to be the crash detection manager and the other processes have to send periodically *heartbeats* to the crash detection manager. In the R-Fed a FT Manager has to be chosen to play the role of the crash detection manager. If the crash detection manager fails a new one has to be chosen. Thus the primary issue in this crash detection technique is an election problem (for which algorithms to solve are given in [Becker 1991]).

If a FT Manager has crashed it must be recovered. For this purpose checkpointing and rollback recovery techniques may be applied. An overview of such techniques can be found in [Elnozahy et al. 1996]. To recover the failed FT Manager the elected crash detec-

tion manager has to restart the failed FT Manager and has to bring the federation to a consistent checkpoint. This can be done through a *remote procedure call* (RPC). But it has to be emphasized that in this case the HLA rules are insured, because the communication between the federates must be done solely via the RTI.

4 FT SIMULATION RUN

Here we explain some scenarios of a FT simulation run based on the architecture discussed in section 2. We describe three situations to illustrate the FT mechanism of the R-Fed. First we show the two directions of the communication between the RTI and the replicas and finally the FT reaction in case of a failure is explained.

4.1 Communication Between RTI and Replicas

Replicas → RTI The transmission of messages from the replicas to the RTI is drawn in figure 3. The

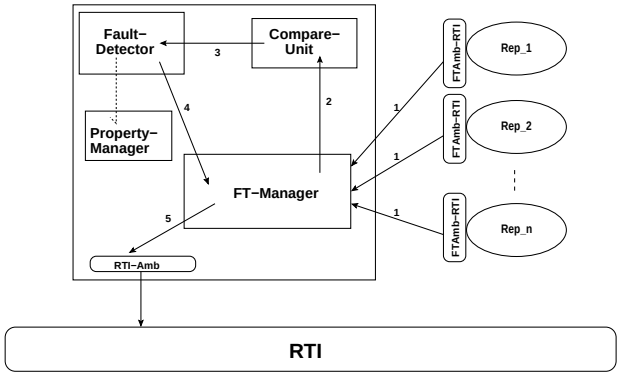

Figure 3: Message Transmission Replicas → RTI

arrows indicate message transfer or function calls and the dotted line between the Fault Detector and Property Manager means "need information from". The numbers close by the arrows are the sequence of the steps of the FT mechanism.

First, the results of the replicas are sent via the FT Manager to the Compare Unit. For this purpose the FTAmb-RTI is responsible that the results are not directly routed to the RTI but over the FT mechanism. The FT Manager can be seen as the corresponding part of the FT mechanism because the communication interfaces are integrated in the FT Manager.

After the results of the replicas are sent to the Compare Unit, the Compare Unit compares the results (for example bit-wise) and registers an eventual time out. This documentation (comparison and time outs) is stored in an internal data structure which is given to the Fault Detector. The Compare Unit calls up the Fault Detector. Next, the Fault Detector evaluates this documentation. The criteria for this evaluation are found in the configuration file of the Property Manager. Thus the Fault Detector looks up the parameters for the FT methods in the Property Manager before it makes the decision whether a failure is detected or not. Furthermore the Fault Detector adjudges due to the configuration file in the Property Manager what to do next, e.g. is a restart of a replica necessary or not.

This decision (failure detection and further FT actions) is given to the FT Manager. If FT procedures have to be done (if a failure is detected) the FT Manager initiates the appropriate FT actions, e.g. terminates a replica. And of course finally, the FT Manager forwards the correct results to the RTI via the RTI-Ambassador.

RTI → Replicas The transmission of messages from the RTI to the replicas is shown in figure 4. In

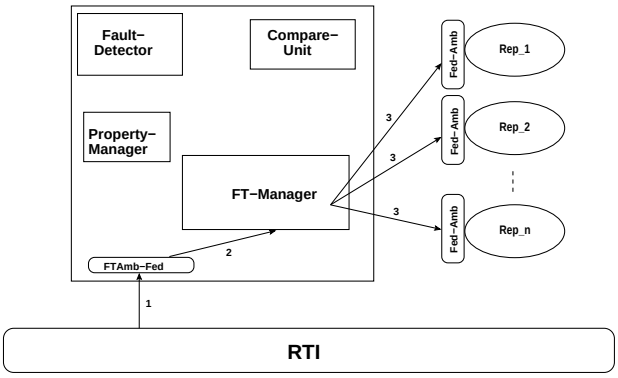

Figure 4: Message Transmission RTI → Replicas

contrast to the other direction of the message transfer (replicas → RTI) in this case the FTAmb-Fed plays the role of the router instead of the FTAmb-RTI. The function calls of the RTI must not directly be sent to the replicas but to the FT Manager previously. The FT Manager broadcasts the RTI messages to all its replicas. The FT Manager is capable for this broadcast because it knows how to reach each replica, i.e. it knows the number, PID, node, IP address and port of each replica. The forwarded RTI messages are taken by the Federate Ambassador (connected to the FT components) of the replica.

4.2 Failure

In this subsection a scenario when a replica fails is shown (cf. figure 5). Let's assume that the FT Manager and its replicas lie on different nodes. In the figure the replicas Rep_1 and Rep_2 belong to FT Manager(1). FT Manager(1) is on node(1) and its replicas on node(2). FT Manager(2) is on node(2) too. Let's assume further that the replica Rep_1 is incorrect.

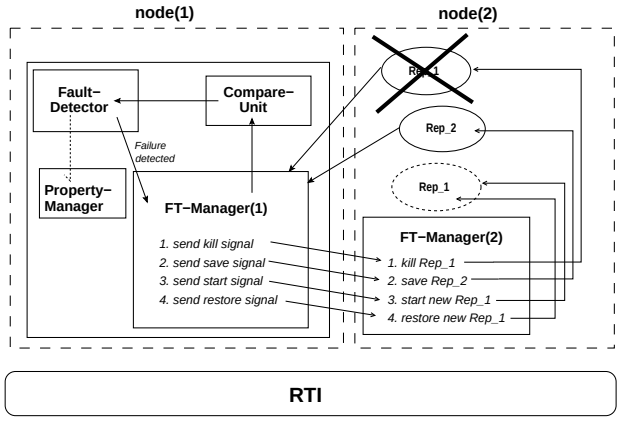

Figure 5: Failure and Restart

The results of the replicas are directly sent by the TCP/IP protocol through the appropriate ports to FT Manager(1). The results are compared and consequently the Fault Detector detects a failure due to the failure criteria listed in the Property Manager. Moreover the Fault Detector determines that a restart of Rep_1 is necessary (again due to the configuration file in the Property Manager). These informations (failure of Rep_1 and required restart) are given to FT Manager(1).

Now FT Manager(1) establishes contact with FT Manager(2). FT Manager(1) sends a (1.) kill, (2.) save, (3.) start and (4.) restore signal to FT Manager(2). Having received the signals FT Manager(2) does the following four steps in a sequel. First it kills Rep_1. Then it saves the simulation state of Rep_2, starts a new Rep_1 and restores the new Rep_1 with the saved simulation state. In the following the two replicas Rep_1 and Rep_2 simulate concurrently again.

5 CONCLUSION

There is no formal failure model in the HLA yet. We developed a new federate concept, the R-Fed, to support fault tolerance. In the R-Fed as well so called Byzantine failures as fail-stop failures can be tolerated. The FT mechanism is based on replication. Several (various) implementations, called replicas, of the original federate are integrated in the R-Fed. The R-Fed consists of the FT components and the replicas.

The aim of the new concept was to maintain the replica in its HLA-usual implementation and to build a FT mechanism around. But the replica needs some (little) modifications. These modifications are a new Federate Ambassador, a new RTI Ambassador (FTAmb-RTI) and save/restore functions. The new ambassadors are responsible that the results of the replica go a roundabout way over the FT mechanism.

This paper gives a short description of our concept of the R-Fed. The R-Fed is the first attempt for an architecture that supports fault tolerance within an HLA-based simulation. A more detailed description of the R-Fed and approaches for solutions to implement the R-Fed are worked out in [Hezel 2000]. However the technical implementation of the R-Fed, experiments and practical experience are subjects for future work.

References

[Becker 1991] T. Becker. *Keeping Processes under Surveillance.* In IEEE 10^{th} Symposium on Reliable Distributed Systems, Pisa, Italy, Sept 30 - Oct 2, 1991, pp. 198-205, IEEE Computer Society Press.

[Dahmann 1999] J.S. Dahmann. *The High Level Architecture and Beyond: Technology Challenges.* Proceedings of the 13^{th} Workshop on Parallel and Distributed Simulation (PADS'99), Atlanta, Georgia, May 1-4 1999, pp. 64-70, IEEE Computer Society Press.

[Défago et al. 1998] X. Défago, A. Schiper, N. Sergent. *Semi-Passive Replication.* In IEEE 17^{th} Symposium on Reliable Distributed Systems, West Lafayette, Indiana, October 20-23 1998, pp. 43-50, IEEE Computer Society Press.

[DMSO] The HLA web page of the U.S. Defense Modeling and Simulation Office, 2000. `http://hla.dmso.mil`.

[Echtle 1990] K. Echtle. *Fehlertoleranzverfahren.* Springer-Verlag Berlin, 1990. (in German).

[Elnozahy et al. 1996] E. Elnozahy, D. Johnson, Y. Wang. *A Survey of Rollback Recovery Protocols in Message Passing Systems.* Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1996.

[Hezel 2000] M. Hezel. *Fehlertoleranz mit der High Level Architecture (HLA).* Studienarbeit UniBwM-IT 17/00, Institut für Technische Informatik, Fakultät für Informatik, Universität der Bundeswehr München, Germany, 2000. (in German).

[Kuhl et al. 2000] F. Kuhl, R. Weatherly, J. Dahmann. *Creating Computer Simulation Systems: An Introduction to the High Level Architecture.* Prentice-Hall PTR. 2000.

[Lamport et al. 1982] L. Lamport, R. Shostak, M. Pease. *The Byzantine Generals Problem.* ACM Trans. on Programming Languages and Systems, Vol. 4, No. 3, July 1982, pp.382-401.

[León et al. 1993] J. León, A.L. Fisher, P. Steenkiste. *Fail-Safe PVM: A Portable Package for Distributed Programming with Transparent Recovery.* Technical Report CMU-CS-93-124, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, 1993.

[Lüthi and Berchtold 2000] J. Lüthi, C. Berchtold. *Concepts for Dependable Distributed Discrete Event Simulation.* Proceedings of the 14^{th} European Simulation Multi-Conference 2000 (ESM2000), Ghent, Belgium, May 23-26 2000, pp. 59-66, SCS Europe.

[OMG 2000] Object Management Group. *Fault Tolerant CORBA Specification, V 1.0.* OMG Document: ptc/2000-04-04, 2000.
`ftp://ftp.omg.org/pub/docs/ptc/00-04-04.pdf`