Concepts for Dependable Distributed Discrete Event Simulation

Johannes Lüthi
Institut für Technische Informatik
Universität der Bundeswehr München
D-85577 Neubiberg, GERMANY
luethi@informatik.unibw-muenchen.de
www.informatik.unibw-muenchen.de/inst4/luethi

Clemens Berchtold
Institut für Technik Intelligenter Systeme (ITIS e.V.)
an der Universität der Bundeswehr München
D-85577 Neubiberg, GERMANY
berch@informatik.unibw-muenchen.de
www.informatik.unibw-muenchen.de/inst4/berchtold

KEYWORDS

Distributed Simulation, Discrete Event Simulation, Dependable Systems, Fault tolerance, HLA

ABSTRACT

In many situations, parallel and distributed simulation is a well-suited approach to overcome performance as well as capacity limitations of complex simulation models. However, if distributed simulation runs take hours or days for termination or if distributed simulation is applied to safety critical domains such as e.g. military decision making, it is not acceptable that a failure in a single process, processing element, or communication link stops the whole simulation experiment. Fault tolerance mechanisms which are popular in general purpose distributed computing are not common in the special case of distributed discrete event simulation. In this work, we structure the necessary dependability requirements and point out approaches how they can be met. Two directions are discussed: the specialization of fault tolerance mechanisms from general purpose distributed computing to distributed simulation and the adaptation of special mechanisms of distributed simulation to support fault tolerance. Special attention is also paid to approaches increasing the dependability of implementations of the High Level Architecture (HLA) and to possibilities to extend the HLA to support fault tolerance mechanisms for HLA federates and federations.

1 INTRODUCTION

Today, simulation as a tool for planning, decision making, scientific analysis, education and training is ubiquitous. Many simulation models require long computation times and need high amounts of memory, leading to restrictions with respect to the size or with respect to the level of detail of the model. Distributed simulation is a means to overcome such limitations. However, most existing architectures for distributed simulation do not include mechanisms for fault tolerance. I.e., in conventional distributed simulation architectures, the failure of a single process, processing element or communication link may stop the whole simulation experiment. Depending on the application domain this may not only be tiresome, but missing fault tolerance may also lead to low cost effectiveness of distributed simulation or even to dangerous and life-critical situations. Consider for example distributed simulation as a decision tool in military operations. In that application domain, highly dependable simulation systems are required. If a simulation run fails, because the distributed simulation architecture lacks fault tolerance, this could reduce the amount of available information for life-critical decisions. Or as another example, consider scientific simulation experiments that may take several days even on a distributed environment. If a failure in one of the processing elements causes a halt of the whole simulation experiment, this may increase the computational cost of the experiments excessively.

Fault tolerance (FT) in general distributed computing as well as several special application domains such as e.g. distributed databases is a well-known and popular issue. However, in the domain of distributed discrete event simulation, only little effort has been made in this direction. Exceptions include the work of Agrawal and Agre about replicated objects in time warp [Agrawal and Agre, 1992], where simulation objects are replicated to improve performance as well as FT of time warp simulations. Another optimistic fault tolerant simulation approach is presented by Damani and Garg [Damani and Garg, 1998], based on checkpoints on stable storage and extension of the anti message mechanism to handle failure recovery. However, these approaches present solutions for specialized situations, and as far as we know, no general discussion of the problems and possible solutions arising from combination of distributed simulation and FT mechanisms has been presented yet. For example, both works only consider data recovery, whereas fault detection and load redistribution due to failures is not discussed. Furthermore, also the *High Level Architecture (HLA)* for distributed simulation, currently becoming popular especially in military simulation (but not restricted to that domain), does not include a formal failure model [Dahmann, 1999]. This is true in two respects: firstly, some processes of current implementations of the HLA's *Run Time Infrastructure (RTI)* are centralized and are thus bottlenecks with respect to FT. Secondly, the HLA itself offers no support for FT of simulation federates of an HLA federation.

In this paper, we structure the basic FT requirements (i.e. fault detection, data recovery, load redistribution) and discuss how (a) existing fault tolerance mechanisms from general distributed computing can be applied to the special case of distributed simulation, and (b) existing mechanisms in distributed simulation can be adapted to support FT. However, this paper presents a structured view of the problem and possible solution approaches. Application experiences of these approaches are not included in this work. In a recently started project at the institution of the authors an HLA-based prototype implementation of a FT distributed simulation is developed. This prototype will be accompanied by performability models to allow experimental as well as analytical evaluations of various approaches. Results in this directions will be reported in future works.

The paper is organized as follows. In Section 2, the general framework of our discussion is laid out. In the sequel, the three main requirements for FT are discussed one by one: in Section 3, issues of fault detection are considered, Section 4 discusses possibilities to allow recovery from failures, and eventually necessary redistribution of load in case of permanent failures is considered in Section 5. Special attention is paid to the HLA and how it can be improved with respect to FT. These issues are taken up in Section 6. Section 7 presents a summary of the paper including plans for future work.

2 GENERAL FRAMEWORK

We restrict our considerations to discrete event-driven simulation models which are distributed in the lines of a logical process architecture (see for example [Ferscha, 1996]) with message based communication and asynchronous simulation time.

2.1 Logical Process Simulation

The distributed logical process architecture which serves as a bases for our discussion is depicted in Figure 1. In this logical process architecture, the simulation model is partitioned into spatial regions. A logical process (LP) consists of a model region (R), a simulation engine (SE), and a logical process communication interface (LPCI). LPs communicate via messages which are handled by the LP's LPCI. The LPs are mapped on processing elements (PE). Such PEs may be processors of a parallel architecture, nodes in a LAN cluster of workstations, or even computers communicating over the internet. On every PE, a PE communication system (PECS) handles the message transfer between its LPs. Message transfer between LPs located on different PEs (external messages) is handled via the PE communication interface (PECI), which is provided by the operating system of each PE. The communication among PEs is provided by an inter-PE-communication system (IPECS), such as e.g. the communication system of a hypercube operating system, a LAN, or the internet.

We refer to the process of finding appropriate subregions of the model as *partitioning.* The distribution of LPs to PEs is referred to as *mapping.* A key issue in distributed discrete event simulation (DDES) with asynchronous virtual times is the need for appropriate simulation protocols to guarantee the absence of causality violations, and thus guarantee that DDES produces the same results as sequential simulation. Since a description of synchronization protocols such as the conservative [Chandy and Misra, 1979] or the optimistic [Jefferson, 1985] approaches would exceed the scope of this contribution, in the sequel we suppose that the reader is familiar with these issues. A detailed introduction can be found for example in [Ferscha, 1996].

2.2 Fault Types

Different types of failures may require different types of reactions and thus different FT requirements. We distinguish between so-called *Byzantine failures* and *fail-stop failures.* In a fail-stop failure, a process stops producing results. However, as long as it produces results they may be assumed to be correct. A situation in which incorrect results are produced is called a Byzantine failure. We restrict our analysis to the case of fail-stop failures. Byzantine failures would require additional failure detection and recovery mechanisms such as e.g. replication with distributed voting.

In this work, three basic types of fail-stop failures are considered: (a) failures of a logical process, (b) processing element failures, and (c) communication link failures. In the case of a PE failure, all LPs located on that PE are stopped. In the case of a communication link failure, the affected PEs may produce results, but messages cannot be communicated to external LPs. Failures of type (b) and (c) may either be temporary or permanent. Furthermore, failures of any type may be controlled (e.g. the shutdown of a workstation) and thus be known in advance or uncontrolled.

2.3 Dependability Requirements

With a few exceptions (e.g. [Agrawal and Agre, 1992], [Damani and Garg, 1998]), DDES architectures do not provide support for FT. Three mechanisms have to be provided to allow appropriate reaction on various types of failures.

Fault detection: Appropriate reaction to failures is only possible if such failures are recognized including their type within reasonable time.

Data recovery: In the case of a failure, local data may be lost. Mechanisms that allow the recovery of such data have to be provided.

Load redistribution: In many cases (such as e.g. the non-temporary breakdown of a PE), LPs have to be redistributed or remapped among the PEs after a failure.

In general purpose distributed computing, FT mechanisms are commonly used. Two strategies to adapt DDES to fault tolerance are available and will be discussed:

1. Application of general purpose FT methods to the special case of DDES.
2. Adaptation of techniques and mechanisms of DDES protocols such that they additionally support FT.

Examples where the second strategy may be used include the extension of optimistic checkpointing techniques to support data recovery or the use of dynamic load balancing mechanisms to manage load redistribution after failures. In both cases, the main problems in DDES such as communication overhead or memory consumption have to be given special consideration. In the

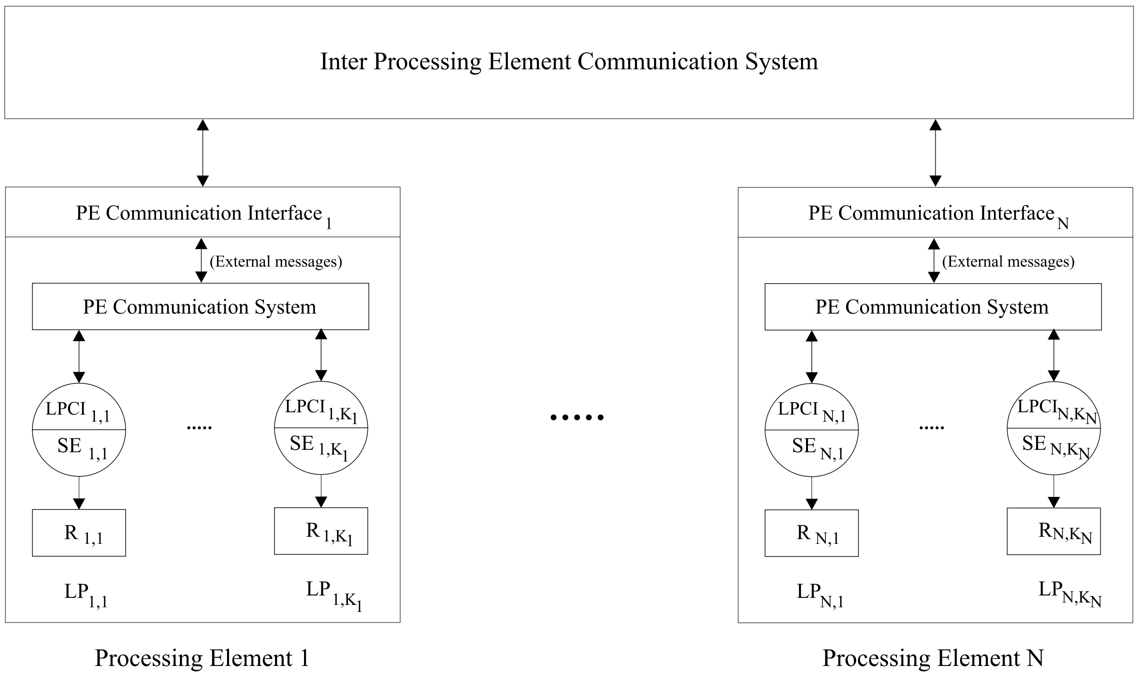

Figure 1: Distributed logical process simulation architecture.

following sections, the three basic dependability requirements and how they may be met in the special case of DDES are discussed in more detail.

3 FAULT DETECTION

The first dependability requirement we focus on is fault detection. If a failure is neglected this may lead to disastrous consequences as discussed in Section 1. Some effects of failures when there is no failure detection are shown in [Chabridon and Gelenbe, 1995]. To allow appropriate reaction to failures, mechanisms have to be implemented which can detect failures of various types.

3.1 Fault Detection Techniques

In distributed systems, FT and consequently fault detection mechanisms are commonly used. A DDES can be seen as a special distributed system and therefore it may be helpful to have a look at general fault detection techniques in distributed systems. In the sequel, we pick up some techniques which are useful for fault detection in the distributed logical process architecture for DDES.

A traditional method to cope with the problem of fault detection in distributed systems is that the processes send *heartbeats* to a process monitor [Avritzer and Weyuker, 1996]. If a heartbeat is not acknowledged within a specified time the process is said to be dead.

Becker proposes two methods for fault detection, where the processes are kept under surveillance [Becker, 1991]. These are:

- Central crash detection,
- Distributed crash detection.

In the central crash detection scheme, a crash detection manager observes the other processes with the help of periodically multicasting messages. If the crash detection manager fails, a new one has to be chosen. The basic idea in the distributed crash detection scheme is to build a virtual token ring. A token is given from one process to another. A crash is detected when the expected token does not arrive within a specified time interval. If the token is lost (and maybe the virtual ring is broken) a ring manager has to be elected to reorganize the virtual ring and to recreate a new token. In both techniques the primary issue is an election problem. In the central crash detection approach, eventually a crash detection manager has to be elected, whereas in the distributed crash detection scheme, a ring manager (to build a virtual ring where each process proves its neighbor) has to be elected. Two algorithms to solve these election problems can be found in [Becker, 1991].

Monitoring techniques can be used to prevent failures by identifying processes that appear to crash with high probability in the near future. Using the so-called *rejuvenation* mechanism [Huang et al., 1995], such processes can be restarted in a controlled way avoiding failures of the affected processes or PEs. For example, such an identification approach based on *fault signatures* is presented by Avritzer and Weyuker [Avritzer and Weyuker, 1996].

3.2 Fault Detection in DDES

In DDES, LPs play the role of the process monitor, crash detection manager or ring manager. Messages similar to NULL-messages, known from the conservative synchronization protocol, can serve as heartbeats. Let us consider what conditions must be fulfilled to detect the three basic types of fail-stop failures reflected in Section 2.2.

Failures of logical processes: We distinguish between detection of LP failures (a) on the same PE and (b) on different PEs. In the first case, both versions, the centralized and the distributed one, can be applied. In the latter case every PE may select a representative which then communicates with the representative of each other PE. An evident choice for the representative is the crash detection manager or the ring manager of each PE. If a representative detects a LP failure on its PE, it broadcasts a message to the other representatives. All the representatives may again select a centralized crash detection manager or may build a virtual token ring. Note that for this mechanism a reliable communication system is assumed.

Processing element failures: Again, LPs acting as representatives of a PE communicate with each other. If a heartbeat of such a LP is not acknowledged, the PE is decided to be dead. The election of a representative within a PE, for example the crash manager or ring manager, guarantees that at least one LP is working on a non-faulty PE. Consequently the representative will indicate whether the PE is alive or not. A reliable communication system is required here, too.

Communication link failures: To detect communication link failures more effort has to be made than to detect LP or PE failures. We restrict our considerations to communication link failures in the IPECS. Communication link failures in the PECS are left to be tolerated by the operating system of the PE. The difficulty stems from the situation that the reason for a missing heartbeat from a representative may either be a PE failure or the heartbeat may be lost via the communication system. Thus a representative from PE PE_i (denoted by $LP_i^{\mathcal{R}}$) has to do the following. If it does not receive the heartbeat of the observed representative $LP_k^{\mathcal{R}}$ it has to send a message to a third representative $LP_l^{\mathcal{R}}$ asking for a heartbeat from $LP_k^{\mathcal{R}}$ to $LP_l^{\mathcal{R}}$. If $LP_l^{\mathcal{R}}$ receives a heartbeat from $LP_k^{\mathcal{R}}$ it is sure that a communication link failure between PE_i and PE_k occurred. Otherwise this procedure has to continue until all links between PE_k and the other PEs are tested. Since all these communication links are declared to be failed, the PE PE_k is no longer reachable or more realistically the PE is dead.

4 DATA RECOVERY

Two techniques to allow recovery of simulation data are *checkpointing* and *replication*. Checkpointing techniques require the availability of *stable storage*. Stable storage is assumed to survive processor or communication failures and thus it is always accessible. Stable storage is an abstraction and may be a stable disk or the volatile memory of another LP. If in the latter case the recovery data (checkpoints and logged messages) of an LP LP_i are sent to the memory of just one other LP, only one failure (with the loss of the data of LP_i as a consequence) can be tolerated. To tolerate an arbitrary number of failures, either a reliable storage system has to be used or the recovery data have to be stored on the local memory of each other LP. For the considerations of Section 4.1 it is assumed that stable storage is available and therefore the recovery data of an LP are accessible at any point in time. However, implementation of stable storage induces high additional computation as well as communication overhead.

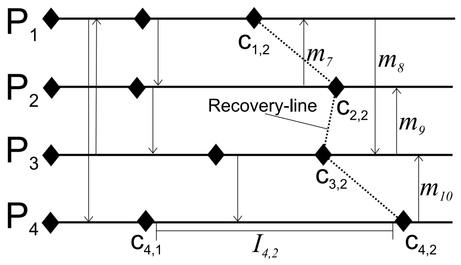

Figure 2: Global checkpoint and recovery line.

4.1 Recovery Techniques

Checkpointing. First we give some definitions. A *stable checkpoint* is a snapshot of the state of a process, saved on nonvolatile storage to survive process failures [Wang, 1995]. In this work we use the term stable checkpoint to distinguish checkpoints made for FT purposes from checkpoints made to allow rollbacks in time warp simulation. We emphasize the fact that a stable checkpoint survives process failures. We denote $c_{i,x}$ as the x^{th} stable checkpoint (x is called the *checkpoint index*) of the process P_i (i is the *process id*). $I_{i,x}$ is defined as the *stable checkpoint interval* between the stable checkpoints $c_{i,x-1}$ and $c_{i,x}$. A *local* checkpoint is a stable checkpoint of one specified process, whereas a *global* checkpoint is a set of local checkpoints, one of each process, such that the system forms a consistent state. A global checkpoint is said to be consistent if no message is sent after a local checkpoint $c_{i,x}$ and received before another local checkpoint $c_{j,y}$ ($c_{i,x}$ and $c_{j,y}$ belong to the global checkpoint) [Wang, 1995]. This implies that after a failure a distributed application can be restarted from this global checkpoint [Baldoni et al., 1997]. The most recent global checkpoint is called the *recovery line* (see Figure 2, arrows indicate messages). However, the determination of a recovery line is in itself a non-trivial task. An example for an approach to that problem which is based on a rollback-dependency graph is proposed in [Wang, 1995].

Elnozahy et al. give a survey of checkpointing techniques where three classes are proposed, namely *uncoordinated*, *coordinated*, and *communication induced* checkpointing [Elnozahy et al., 1996]. In uncoordinated checkpointing techniques, the processes take the (local) checkpoints independently. The main problem of uncoordinated checkpointing techniques is the possibility of the so-called *domino effect* (see for example [Upadhyaya and Ranganathan, 1994] or [Elnozahy et al., 1996]). Unfortunately, in the case of the domino effect, a huge amount of the calculations may be lost. A solution to overcome the domino effect is that the LPs take their checkpoints in coordination with the other LPs with the aim of a global checkpoint. The drawbacks of coordinated checkpointing are the loss of the LP autonomy and a higher communication overhead. Another way to avoid the domino effect is *communication induced* checkpointing. One method in communication induced checkpointing is the *index based* variation (see for example [Baldoni et al., 1997]), where consistency between checkpoints of the same index is guaranteed.

Instead of solely taking checkpoints, another technique for data recovery in distributed systems is the *message logging technique*. In this case the processes store messages and *non-deterministic* events in addition to the checkpoints such that it is possible to replay the calculations since the last checkpoint. Therefore this method has to be used in the case of interactive distributed simulation because user interactions are non-deterministic events. Furthermore, this method is not subject to the domino effect. Message logging techniques can be divided into three categories: *pessimistic*, *optimistic* and *causal* [Alvisi and Marzullo, 1998].

Replication. Replication is the maintenance of online copies of data and other resources. It is a key to achieve good performance, high availability and FT in distributed systems [Coulouris et al., 1994]. The main problem in replication techniques is keeping the multiple replicas in a consistent state.

It can be distinguished between *active* and *passive* replication [Shi and Belford, 1989]. In the passive replication scheme, also called the *primary-backup approach*, a primary replica executes the computations and then sends state copies to the backup replicas. If the primary replica fails, one of the backup replicas has to take over. In the active replication scheme all replicas act independently and perform the computations concurrently.

Another replication technique, especially popular in the domain of reliable storage systems, which may be applied to DDES is the RAID (Redundant Arrays of Inexpensive Disks) scheme. An introduction to RAID can be found in [Patterson et al., 1989]. The main disadvantage of these techniques is that such strategies can tolerate only one failure. If two processes fail at the same time, the system can not recover the lost data. Therefore after a failure the data have to be reclaimed before another failure occurs. In [Plank, 1996], coordinated checkpointing with the help of RAID techniques is proposed. Two RAID strategies are applied, *disk mirroring* and *parity disk* (level 1 and level 5 RAID, respectively). The reason to use level 5 RAID is to create a parity checkpoint which is defined as the bitwise exclusive-or of each local checkpoint. If a LP fails, its checkpoint can be constructed from this parity checkpoint. Therefore the overhead in the resource usage, caused by checkpoints and replications, can be reduced with the help of RAID strategies.

4.2 Recovery Techniques in DDES

In DDES, the data structures for conservative and optimistic simulation are different. To transfer ideas from general checkpointing techniques to DDES, it must be considered what data is necessary for checkpointing. In a conservative protocol a stable checkpoint of an LP consists of the state variables, the event list and the virtual time of an LP. In an optimistic protocol additionally the input and output queues of the sent and received messages have to be stored to guarantee the capability to roll back. Note that in time warp local checkpoints are taken anyway. Therefore the only effort that has to be made with respect to FT is to periodically transfer checkpoints to stable storage. In the coordinated checkpointing scheme the LPs have to coordinate the checkpoints to build a global checkpoint, i.e., a consistent state. If a LP LP_i fails, the LPs depending on LP_i have to roll back to the most recent available global state. Another advantage of the coordinated approach is that all simulation data before the recovery line are no longer important and can be discarded. In that sense, the determination of a recovery line in coordinated checkpointing is analogous to the computation of global virtual time (GVT) in DDES. Thus, GVT computation or approximation techniques such as e.g. presented in [Tomlinson and Garg, 1993] and [D'Souza et al., 1994] can be used for the combined computation of both, GVT as well as recovery lines.

To apply message logging protocols in DDES, the LPs store the incoming and outgoing messages on stable storage. Pessimistic and causal logging in DDES immediately lead to an overhead, because in DDES message activity is usually high [Damani and Garg, 1998].

A significant amount of research has been done to determine optimal checkpoint intervals in optimistic DDES. An overview of checkpointing considerations for optimistic DDES including an experimental comparison can be found in [Fleischmann and Wilsey, 1995]. Analytical models that compute optimal checkpointing intervals (see for example the comparison of copy state saving and incremental state saving in [Cleary et al., 1994] or the adaptive approach presented in [Rönngren and Ayani, 1994], where the length of the checkpointing interval is determined at runtime) have to be extended by considerations about stable checkpoints. This implies a major change in such models since (a) failures introduce an additional cost factor and (b) checkpointing to stable storage introduces another type of checkpoint. Thus, models must not only determine an optimal checkpointing interval, but a checkpointing strategy which considers an optimal mix of both, local volatile checkpoints as well as stable checkpoints has to be found.

Replication of LPs is another idea in DDES to achieve FT. In the passive scheme only the primary LP executes the events and sends the actual state to its back-up replicas. To tolerate a PE crash, the back-up LPs must lie on another PE. When the back-up LPs are informed of the failed primary replica, one of the backup LPs takes over the role of the primary LP. In the active scheme, several LPs, perhaps again on different PEs, will execute the events independently. This approach is only applicable if it can be guaranteed that the active replicas produce exactly the same simulation results. Active replication needs more processing power, whereas in passive replication the message activity is high. Thus, if in a DDES the message overhead is critical, as in the conservative protocol, the active replication would be more preferable. On the other hand, if memory and processing power is the primary bottleneck, as typically in time warp simulations, passive replication should be considered. RAID techniques such as described by [Plank, 1996] can be attractive in memory consuming simulations, because the level 5 RAID strategy is known to be memory efficient as compared to replication.

5 LOAD REDISTRIBUTION

After a permanent breakdown of a PE, the LPs assigned to that PE have to be mapped on other available PEs. If a temporary breakdown is experienced, the trade-off between the blocking time during the breakdown and the overhead due to LP redistribution has to be considered. If the faulty PE can be restarted immediately, it may be more efficient to accept the simulation delays due to the temporary breakdown. On the other hand, if the PE restart takes more time (e.g. reboot of a workstation) the cost of waiting for that PE will exceed the cost for load redistribution.

In the case of controlled failures (e.g. the shutdown of a workstation which may be reported to the simulator in advance), load redistribution may be the only necessary reaction. In the case of an uncontrolled failure, successful fault detection and data recovery are preconditions to load redistribution.

5.1 Partitioning and Mapping of LPs

Since the highest level of detail for our discussion is the LP level, we do not consider changes w.r.t. the partitioning of the simulation model into submodels simulated by the LPs. Thus, an eventually necessary load redistribution is actually a re-mapping of the failed PE's LPs to functioning PEs. However, to allow a balanced load redistribution after a failure of a PE, a partitioning strategy which produces significantly more LPs than PEs would be preferable.

For models of several application domains, specialized partitioning strategies have been developed that are optimized for efficient distributed simulation. They produce model regions which can be mapped to PEs in a way that decreases communication overhead, reduces the number of rollbacks in time warp simulation, or maximizes lookahead for conservative simulation. Some examples of such techniques include hierarchical partitioning for time warp simulation presented by Kim et al. [Kim et al., 1998], or the mapping approach proposed by Som and Sargent [Som and Sargent, 1993]. However, a problem with load redistribution after a PE failure is that optimized structures provided by such specialized mapping algorithms may have to be broken when the LPs of the failed PE are moved to other PEs.

5.2 Using Static Load Management

Static strategies perform the partitioning and mapping operations before the simulation start and do not intend changes in the load distribution during the runtime of the simulator. Thus, one may think of static mapping approaches as being unsuitable for being used in FT mechanisms. However, static partitioning and mapping algorithms, such as e.g. that described by Boukerche and Tropper [Boukerche and Tropper, 1994], could be used to prepare several mappings for a restricted number of failure situations: consider a situation where originally, N PEs are available. If there is reason to assume that the total number of PEs that may eventually fail during the simulation run is rather small, static mappings for $N, N-1, \ldots, N-K$ PEs may be produced in advance, where K denotes the maximum number of failed PEs. In the case of a failure, these mappings may be used to redistribute the LPs. However, it has to be noted that in the change over from a mapping for N PEs to a mapping for $N-1$ elements, eventually not only the LPs of the failed PE may have to be moved. Thus, a static mapping strategy used for that purpose should be optimized in a way that mappings for varying numbers of PEs differ only slightly.

5.3 Dynamic Load Management

As opposed to static load management, load distribution during runtime is the key component of dynamic load balancing techniques. Thus, they can directly be applied to redistribute LPs originally assigned to a meanwhile failed PE. Most load balancing approaches for general purpose distributed computing are not appropriate to be used for DDES, because they do not take into account properties of the respective DDES synchronization protocols. However, general approaches that allow the consideration of *close neighbors* and *affinity* of LPs to certain PEs may be used for our purpose. Such an approach for the reassignment of tasks from a failed PE is described in [Varvarigou and Trotter, 1994]. Methods specialized for load balancing of distributed simulators (see for example [Schlagenhaft et al., 1995], [Choe and Tropper, 1999], or [Deelman and Szymanski, 1998]) can be used for the LP redistribution after a failure. Using such an approach, a failed PE may be considered as being completely overloaded, i.e. all its LPs have to be reassigned. Additionally, proactive strategies that monitor process behavior and take into account different failure probabilities at different PEs to assign critical processes to "save" PEs may be considered.

6 HLA FAULT TOLERANCE

The High Level Architecture (HLA), developed by the U.S. Defense Modeling and Simulation Office (DMSO) provides a general software architecture for distributed simulation [HLA]. The HLA is defined by its three components: the HLA rules, the object model template, and the interface specification. Its primary purpose is to serve as a tool for the common execution of a composable set of interacting simulations with the goal of increasing model reusability. However, the Run Time Infrastructure (RTI), which is the implementation of the HLA interface specification, can also be used as a building block for parallel simulation on high-performance computers [Steinman et al., 1999]. Unfortunately, in its current state, the HLA has no formal failure model [Dahmann, 1999]. This is of special significance since the most important application domain for the HLA is military simulation, where highly dependable systems may be required, especially if simulation is used as a decision tool in military operations. To increase the FT of HLA-based simulation, two extensions are desirable: (i) the RTI and its supporting processes have to be implemented in a highly dependable way, (ii) services that support FT of HLA federates should be included in the HLA specification.

6.1 FT RTI Implementation

There are at least two dependability bottlenecks in the current RTI implementations: the RTIexec, as well as the FedExec processes are located on a single node and operate without replication. The RTIexec process is the startup process of an HLA federation. Every HLA federate has to register via the RTIexec process. The FedExec process is the main administration unit of the RTI: it handles all communication among federates, it is responsible for object identification, and it manages ownership of objects and attributes. I.e., without the FedExec process, interaction between HLA federates is impossible. To improve the dependability of a distributed simulation based on the HLA, these two processes could be replicated and their data should periodically be saved to stable storage. A fault detection scheme should be provided in the RTI implementation including a rollback mechanism which allows continuation of computation starting from the last stable checkpoint before failure.

6.2 FT Support for HLA Federates

Including obligatory FT capabilities in the HLA rules may be too restrictive. However, the interface specification, which describes both, the services provided to the federates by the RTI and the services provided to the RTI by the federates could be extended by services supporting FT mechanisms. For example, such services may include fault detection services, services managing replication of federates and federate respawn after failure, or the extension of the RTI save/restore services to support periodic checkpointing to stable storage and automated recovery after failures. A detailed discussion of options for a FT-HLA is a subject of future work.

7 CONCLUSIONS

With few exceptions, fault tolerance has not been a major issue in the parallel and distributed simulation community. However, safety as well as efficiency considerations give rise to the desire for dependable distributed simulation software architectures. In this paper, we have structured the requirements for fault tolerant distributed discrete event simulation. The three major issues fault detection, data recovery, and load redistribution have been considered in more detail. Techniques to increase fault tolerance in general purpose distributed computing have been presented and their application to distributed discrete event simulation has been discussed. Where applicable, the adaptation of mechanisms which are already part of certain distributed simulation protocols to support fault tolerance has been considered. Additionally, special attention has been paid to approaches that could increase dependability of HLA-based distributed simulation.

Plans for future work point in several directions: in a recently initiated project, combined models that represent performance, dependability, and cost aspects are built to analyze and evaluate approaches proposed in this paper. A choice of approaches will be implemented as experimental prototypes based on the HLA. As a result of these research activities we plan to formulate concepts for additional services in the HLA interface specification which support fault tolerance mechanisms for HLA federates and federations. Additionally, the application of fault tolerance mechanisms to other variants of parallel and distributed simulation, such as synchronized simulation or simulation of continuous models may be considered.

References

[HLA] HLA web-page of the u.s. defense modeling and simulation office. `http://www.dmso.mil/portals/hla.html`.

[Agrawal and Agre, 1992] Agrawal, D. and Agre, J. R. (1992). Replicated objects in time warp simulations. In Swain, J., Goldsman, D., Crain, R., and Wilson, J., editors, *Proceedings of the 1992 Winter Simulation Conference*, pages 657–664.

[Alvisi and Marzullo, 1998] Alvisi, L. and Marzullo, K. (1998). Message logging: Pessimistic, optimistic, causal and optimal. *IEEE Transactions on Software Engineering*, 24(2):149–159.

[Avritzer and Weyuker, 1996] Avritzer, A. and Weyuker, E. J. (1996). Detecting failed processes using fault signatures. In *IEEE International Computer Performance and Dependability Symposium, Urbana-Champaign, Illinois, September 4-6, 1996*, pages 302–311. IEEE Computer Society Press.

[Baldoni et al., 1997] Baldoni, R., Quaglia, F., and Fornara, P. (1997). An index-based checkpointing algorithm for autonomous distributed systems. In *IEEE* 16^{th} *Symposium on Reliable Distributed Systems, Durham, NC, USA, 22-24 October 1997*, pages 27–34. IEEE Computer Society Press.

[Becker, 1991] Becker, T. (1991). Keeping processes under surveillance. In *IEEE* 10^{th} *Symposium on Reliable Distributed Systems, Pisa, Italy, Sept. 30-Oct. 1-2, 1991*, pages 198–205. IEEE Computer Society Press.

[Boukerche and Tropper, 1994] Boukerche, A. and Tropper, C. (1994). A static partitioning and mapping algorithm for conservative parallel simulations. In *Proceedings of the* 8^{th} *Workshop on Parallel and Distributed Simulation (PADS '94), Edinburgh, Scotland, UK, July 6–8, 1994*, pages 164–172. IEEE Computer Society Press. (SIGSIM Newsletter Vol. 24, No. 1).

[Chabridon and Gelenbe, 1995] Chabridon, S. and Gelenbe, E. (1995). Failure detection algorithms for a reliable execution of parallel programs. In *IEEE* 14^{th} *Symposium on Reliable Distributed Systems, Bad Neuenahr, Germany, 13-15 September 1995*, pages 229–238. IEEE Computer Society Press.

[Chandy and Misra, 1979] Chandy, K. M. and Misra, J. (1979). Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering*, SE-5(5):440–452.

[Choe and Tropper, 1999] Choe, M. and Tropper, C. (1999). On learning algorithms and balancing loads in time warp. In *Proceedings of the* 13^{th} *Workshop on Parallel and Distributed Simulation (PADS '99), Atlanta, GA, USA, May 1–4, 1999*, pages 101–108. IEEE Computer Society Press.

[Cleary et al., 1994] Cleary, J., Gomes, F., Unger, B., Zhonge, X., and Thudt, R. (1994). Cost of state saving & rollback. In *Proceedings of the* 8^{th} *Workshop on Parallel and Distributed Simulation (PADS '94), Edinburgh, Scotland, UK, July 6–8, 1994*, pages 94–101. IEEE Computer Society Press. (SIGSIM Newsletter Vol. 24, No. 1).

[Coulouris et al., 1994] Coulouris, G., Dollimore, J., and Kindberg, T. (1994). *Distributed Systems: Concepts and Design*. Addison-Wesley.

[Dahmann, 1999] Dahmann, J. S. (1999). The high level architecture and beyond: Technology challenges. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation (PADS '99), Atlanta, GA, USA, May 1–4, 1999*, pages 64–70. IEEE Computer Society Press.

[Damani and Garg, 1998] Damani, O. P. and Garg, V. K. (1998). Fault-tolerant distributed simulation. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS '98), Banff, Alberta, Canada, May 26–29, 1998*, pages 38–45. IEEE Computer Society Press. (Simulation Digest, Vol. 28, No. 1, July 1998).

[Deelman and Szymanski, 1998] Deelman, E. and Szymanski, B. K. (1998). Dynamic load balancing in parallel discrete event simulation for spatially explicit problems. In *Proceedings of the 12th Workshop on Parallel and Distributed Simulation (PADS '98), Banff, Alberta, Canada, May 26–29, 1998*, pages 46–53. IEEE Computer Society Press. (Simulation Digest, Vol. 28, No. 1, July 1998).

[D'Souza et al., 1994] D'Souza, L. M., Fan, X., and Wilsey, P. A. (1994). pGVT: An algorithm for accurate GVT estimation. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94), Edinburgh, Scotland, UK, July 6–8, 1994*, pages 102–109. IEEE Computer Society Press. (SIGSIM Newsletter Vol. 24, No. 1).

[Elnozahy et al., 1996] Elnozahy, E., Johnson, D., and Wang, Y. (1996). A survey of rollback-recorvery protocols in message-passing systems. Technical Report CMU-CS-96-181, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, USA, http://reports-archive.adm.cs.cmu.edu/anon/1996/CMU-CS-96-181.ps.

[Ferscha, 1996] Ferscha, A. (1996). Parallel and distributed simulation of discrete event systems. In Zomaya, A. Y., editor, *Handbook of Parallel and Distributed Computing*, pages 1003–1041. McGraw-Hill.

[Fleischmann and Wilsey, 1995] Fleischmann, J. and Wilsey, P. A. (1995). Comparative analysis of periodic state saving techniques in time warp simulators. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95), Lake Placid, NY, USA, June 14–16, 1995*, pages 50–58. IEEE Computer Society Press. (Simulation Digest Vol. 25, No. 1).

[Huang et al., 1995] Huang, Y., Kintala, C., Kolettis, N., and Fulton, D. N. (1995). Software rejuvenation: Analysis, module and applications. In *IEEE 25th International Symposium on Fault-Tolerant Computing, Pasadena, California, June 27–30, 1995*, pages 381–390. IEEE Computer Society Press.

[Jefferson, 1985] Jefferson, D. R. (1985). Virtual time. *ACM Transactions on Programming Languages and Computer Systems*, 7(3):404–425.

[Kim et al., 1998] Kim, K. H., Kim, T. G., and Park, K. H. (1998). Hierarchical partitioning algorithm for optimistic distributed simulation of devs models. *Journal of Systems Architecture — Special Issue: Parallel and Distributed Simulation*, 44(6 & 7):433–455.

[Patterson et al., 1989] Patterson, D. A., Chen, P., Gibson, G., and Katz, R. H. (1989). Introduction to redundant arrays of inexpensive disks (RAID). In *IEEE Spring 89 COMPCON, San Francisco, February 27 – March 3, 1989*, pages 112–117. IEEE Computer Society Press.

[Plank, 1996] Plank, J. S. (1996). Improving the performance of coordinated checkpointers on networks of workstations using RAID techniques. In *IEEE 15th Symposium on Reliable Distributed Systems, Niagara-on-the-Lake, Ontario, Canada, 23-25 October 1996*, pages 76–85. IEEE Computer Society Press.

[Rönngren and Ayani, 1994] Rönngren, R. and Ayani, R. (1994). Adaptive checkpointing in time warp. In *Proceedings of the 8th Workshop on Parallel and Distributed Simulation (PADS '94), Edinburgh, Scotland, UK, July 6–8, 1994*, pages 110–117. IEEE Computer Society Press. (SIGSIM Newsletter Vol. 24, No. 1).

[Schlagenhaft et al., 1995] Schlagenhaft, R., Ruhwandl, M., Sporrer, C., and Bauer, H. (1995). Dynamic load balancing of a multi-cluster simulator on a network of workstations. In *Proceedings of the 9th Workshop on Parallel and Distributed Simulation (PADS '95), Lake Placid, NY, USA, June 14–16, 1995*, pages 175–180. IEEE Computer Society Press. (Simulation Digest Vol. 25, No. 1).

[Shi and Belford, 1989] Shi, S. S. and Belford, G. G. (1989). Consistent replicated transactions: A highly reliable program execution environment. In *IEEE 8th Symposium on Reliable Distributed Systems, Seattle, Washington, 10-12 October 1989*, pages 30–41. IEEE Computer Society Press.

[Som and Sargent, 1993] Som, T. K. and Sargent, R. G. (1993). A new process to processor assignment criterion for reducing rollbacks in optimistic simulation. *Journal of Parallel and Distributed Computing*, 18(4):509–515.

[Steinman et al., 1999] Steinman, J. S., Berliner, G., Blank, G. E., Brutocao, J. S., Burckhardt, J., Peckham, M., Shupe, S., Stadsklev, K., Tran, T., Van Iwaarden, R., and Yu, L. (1999). The SPEEDES-based run-time infrastructure for the high-level architecture on high-performance computers. In Tentner, A., editor, *Proceedings of the High Performance Computing Symposium – HPC'99*, pages 255–266. SCS.

[Tomlinson and Garg, 1993] Tomlinson, A. I. and Garg, V. K. (1993). An algorithm for minimally latent global virtual time. In *Proceedings of the 7th Workshop on Parallel and Distributed Simulation (PADS '93), San Diego, CA, USA, May 16–19, 1993*, pages 35–42. IEEE Computer Society Press. (Simulation Digest Vol. 23, No. 1).

[Upadhyaya and Ranganathan, 1994] Upadhyaya, S. J. and Ranganathan, A. (1994). Rollback recovery in concurrent systems. In Casavant, T. L. and Singhal, M., editors, *Readings in Distributed Computing Systems*, pages 249–266. IEEE Computer Society Press.

[Varvarigou and Trotter, 1994] Varvarigou, T. A. and Trotter, J. A. (1994). Distributed reconfiguration of multiprocessor systems. In *Proc. 13th Symposium on Reliable Distributed Systems*, pages 212–221. IEEE.

[Wang, 1995] Wang, Y.-M. (1995). Maximum and minimum consistent global checkpoints and their applications. In *IEEE 14th Symposium on Reliable Distributed Systems, Bad Neuenahr, Germany, 13-15 September 1995*, pages 86–95. IEEE Computer Society Press.

Johannes Lüthi received his M.Sc. and Ph.D. degrees in applied mathematics from the University of Vienna, Austria in 1995 and 1997, respectively. From 1995 until 1997 he was with the Department of Advanced Computer Engineering, University of Vienna, as a research assistent. Since 1998 he is an assistent professor at the Department of Computer Science of the University of the Federal Armed Forces in Munich, Germany. His primary research interests include performance and dependability analysis of computer and communication systems and distributed discrete event simulation.

Clemens Berchtold holds a M.Sc. degree in mathematics from the University of Innsbruck, Austria in 1997. He is now working as research assistant at ITIS e.V. at the University of the Federal Armed Forces in Munich, Germany. His current research interests are in the areas of distributed simulation, metamodelling and design of simulation experiments.