Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

HLA-BASED OPTIMISTIC SYNCHRONIZATION WITH SLX

Steffen Strassburger

Department for Industrial Information Systems
Ilmenau University of Technology
P.O. Box 100 565
98684 Ilmenau, GERMANY

ABSTRACT

The High Level Architecture for Modeling and Simulation (HLA) comes with the promise of facilitating interoperability between a wide variety of simulation systems. HLA's time management offers a unique support for heterogeneous time advancement schemes and differentiates HLA from other general interoperability standards. While it has been shown that HLA is applicable for connecting commercial off-the-shelf simulation packages (CSPs), the usage of HLA time management in this application area is virtually always limited to conservative synchronization. In this paper, we investigate HLA's capabilities concerning optimistic synchronization. For the first time, we show its use in combination with a commercial-off-the-shelf simulation package (CSP), namely the simulation system SLX. We report on implementation details, performance results, and potential limitations in the current HLA 1516.1-2010 standard and its interpretation by runtime infrastructure (RTI) software vendors.

1 INTRODUCTION

HLA has been used for facilitating interoperability between commercial-off-the-shelf simulation packages (CSPs) for quite a number of years. First investigations into the subject were based on the "Simulation Language with Extensibility" (SLX) (Straßburger et al. 1998) and were soon followed by several other studies involving CSPs such as Matlab (Pawletta et al. 2000) and Simul8 (Mustafee and Taylor 2006). While early work on HLA integration was clearly focused on the technical feasibility of such an integration, later studies also investigated the industrial applicability of HLA-based coupling of CSPs (Boer 2005; Lendermann et al. 2007; Strassburger, Schmidgall, and Haasis 2003). HLA seems to be an adequate interoperability standard for supporting hybrid simulations (Mustafee et al. 2015).

Several implementation options regarding the HLA integration of CSPs have been discussed in literature and can be classified as using either an explicit or implicit approach of providing HLA functionality (Strassburger 2001; Wang et al. 2005). In the explicit approach, typically a CSP-specific mapping of the original HLA application programming interface (API) is integrated into the CSP somehow. This may be in the form of functions or statements extending the original simulation language (as used in SLX (Strassburger 2006)) or in the provision of HLA components ("building blocks") for component based simulators (e.g., Plant Simulation, Quest, etc. (Strassburger 2006; Strassburger, Schmidgall, and Haasis 2003)).

In the implicit approach, it is attempted to hide HLA functionality from the user, freeing him from any implementation overhead concerning HLA functionality. The feasibility of this approach was demonstrated using the simulation system Simplex 3 (Strassburger 2001).

Integrating a CSP into the HLA typically imposes significant requirements on the CSP. While the implicit approach typically requires full access to the source code of the CSP, the explicit approach at least requires some way of extending the CSPs original functionality. A common means for that are library interfaces, such as the Dynamic Link Library (DLL) interface that many CSPs based on the Windows operating system offer.

Further requirements are imposed by the need to meaningfully exchange data between the CSP and the HLA runtime infrastructure (RTI) software. CSP capabilities regarding the fulfillment of these requirements vary. Regarding the most important requirement in distributed simulation, which is the efficient synchronization; capabilities of CSPs are often quite limited.

The synchronization aspect of the HLA integration of CSPs requires the user to have some influence on the event scheduling mechanism of the CSP. The vast majority of know publications is in that regard limited to the application of conservative synchronization schemes. Often, even time stepped approaches have been applied.

While conservative synchronization schemes are rather easy to use and implement, their performance depends on the ability to specify (desirably large) Lookahead values for each simulation model. Lookahead represents a way of providing guarantees to other members of a distributed simulation on how far into the future a subsequent message/event will be scheduled. This requirement is both complicated to convey to modelers not familiar with distributed simulation and often highly dependent on the simulation problem at hand. In the worst case, Lookahead values of zero have to be assumed, leading to a quasi-sequential execution of the CSPs in the distributed simulation.

In parallel simulation, it is frequently stated that the use of optimistic synchronization provides more hope for faster model execution (Fujimoto 2000). In CSPs, optimistic synchronization has not yet been used in practice, as CSPs typically do not provide any state-saving capabilities.

The objective of the research presented here was to practically investigate, whether HLA based optimistic synchronization could actually be integrated into a CSP of our choice. With SLX, a CSP was found which fulfilled the basic requirement of a state saving capability. Our investigation focused on options of integrating optimistic synchronization into an existing HLA interface of this CSP and on ways for transparently offering optimistic synchronization to SLX.

The remainder of this article is structured as follows. Section 2 discusses related work. Section 3 introduces requirements for optimistic synchronization in conjunction with a CSP. Section 4 analyzes the capabilities of SLX and its HLA interface. Section 5 presents the concept and implementation of the extensions needed for optimistic synchronization. Section 6 presents a small case study and performance results. Section 7 discusses problems encountered in conjunction with the use of HLA's flushQueueRequest service. Section 8 summarizes and gives recommendations of further actions.

2 RELATED WORK

Efficient synchronization protocols have been a focal point in parallel discrete event simulation (PDES) research for a significant number of years. In this article it is assumed that the reader is familiar with the basic concepts behind conservative and optimistic synchronization protocols. For those who are not Fujimoto (2000) and a Perumalla (2006) provide in-depth discussions.

Conservative protocols are generally considered to be easier to implement and use. They maintain causality in a distributed simulation by forcing its members to block until it can be safely guaranteed that no messages in their logical past will be received. Their major drawback is a dependency on the ability to specify Lookahead values.

In optimistic protocols, simulations can process messages even if there is no guarantee that messages with a lower timestamp will not be received in their future. This "optimistic" execution of messages is based on the hope that causality violations, although possible, in fact will not or only sparsely occur. If an optimistically synchronized simulation receives a message that is in its logical past, it must take actions to reestablish causality. This is typically achieved by performing a rollback to a state that was previously recorded. The Time Warp protocol (Jefferson 1985) is an example of an optimistic synchronization mechanism using this approach. Other mechanisms to recover from causality violations include reverse computation (Carothers, Perumalla, and Fujimoto 1999).

The HLA is a distributed simulation standard that intends to support heterogeneous time advancement schemes, including the above mentioned conservative and optimistic protocols. The basic idea of HLA time management services is that HLA federates (i.e., individual simulations participating in an HLA based distributed simulation) have to request time advancement from the RTI. The RTI coordinates these requests and issues time advance grants according to the requests and guarantees it has.

The usage of HLA for the coupling of CSPs was traditionally based on conservative or time stepped synchronization protocols, but did not include optimistic protocols. For CSPs that allow access to event scheduling mechanisms, classical conservative synchronization was applied, based on HLA services such as "nextMessageRequest" (NMR) and "nextMessageRequestAvailable" (NMRA) (Straßburger et al. 1998; Strassburger 2006). For CSPs without access to event scheduling, often time stepped synchronization was applied (Strassburger, Schmidgall, and Haasis 2003), taking into account its potential disadvantages concerning performance and accuracy.

Not many publications focus on the usage of HLA-based optimistic synchronization in general and on bringing HLA-based optimistic synchronization to the domain of CSPs in specific. The prior is likely due to reasons of HLA not being a standard commonly frequented by the PDES community, at least if high performance and efficient execution are intended. The few exceptions include Ferenci, Perumalla, and Fujimoto (2000), who investigated the options for federating different instances of Georgia Tech Time Warp (GTW) simulations and Vardanega and Maziero (2000), who proposed the idea of a generic rollback manager for freeing optimistic HLA federates from some of the implementation overhead of optimistic synchronization.

The latter (bringing HLA-based optimistic synchronization to CSPs) is fundamentally challenged by the general unavailability of state saving capabilities in CSPs and the impossibility to implement concepts such as state saving or reverse computation in a CSP without dedicated vendor support.

The only research that investigated optimistic synchronization in the context of CSPs was conducted by Wang et al. (2005). Their work focused on ways of providing optimistic synchronization capabilities to a CSP in a manner that does not require major user involvement. Towards that goal they put forward the use of a rollback controller (comparable to that discussed in (Vardanega and Maziero 2000)) that was supposed to handle the rollback procedure on behalf of the CSP. The rollback controller would control the state saving of a CSP by calling "setState" and "getState" functions in the CSP.

While some of the ideas of the rollback controller have been adopted in our work, the major limitation in the publication of Wang et al. (2005) is that no actual CSP was used to validate the approach; instead CSP emulation was applied.

A different approach for transparently encapsulating requirements imposed from optimistic synchronization was introduced by Santoro and Quaglia (2005) under the name "MAgic State Manager" (MASM). MASM allows performing checkpointing/recovery of the state of a federate in a way completely transparent to the federate itself. MASM implementation is based on identifying and storing the memory image of the federate (i.e., the federate state) at the operating system level. Therefore the solution requires low level access to processes at the operating system level, which limits the approach to open operating systems such as Linux.

Further ideas capable of bringing optimistic synchronization to simulations only capable of conservative synchronization include the transparent time-management conversion introduced in (Santoro and Quaglia 2012). Here, conservative time-management calls to the HLA RTI are intercepted by a Time-Management-Converter (TiMaC) which interfaces with the RTI using optimistic message delivery. A prerequisite in this approach, however, is the existence of the aforementioned MASM.

Further related work concerning the HLA integration of CSPs investigated interoperability patterns that typically occur when connecting models from different CSPs (Taylor et al. 2012). As a result, interoperability reference models were defined as SISO standard SISO-STD-006-2010 (SISO 2010).

3 REQUIREMENTS FOR OPTIMISTIC SYNCHRONIZATION

Several requirements are imposed on a simulation system when optimistic synchronization shall be applied. The most prominent requirement results from the need for a recovery mechanism. A recovery mechanism must be capable of undoing the potentially damaging results that out-of-order-processing of messages can incur.

As part of a recovery mechanism, a previous state of the simulation must be restored. Typically this is achieved by applying some kind of state saving and state restoration technique (only these are considered here). For a CSP, a major requirement is therefore to provide/implement a state saving technique which is capable of storing all relevant state information during a simulation run at arbitrary simulation time stamps. Relevant state information includes the state of the user model (e.g., state variables, object instances) as well as system state (e.g., position of random number generators, event lists, etc.). Stored state information must be identifiable by timestamp. Furthermore, the CSP must provide mechanisms to restore and delete previously stored system states.

Other requirements needed to recover from causality violations are the need to cancel any sent messages that become invalid, because a straggler message has been received. To be able to cancel such messages, a federate must keep a log of all sent messages and their time stamp. To be able to reprocess any received messages after a state rollback, an optimistic federate must also keep a log of all incoming messages including their time stamps. In the HLA, all calls to services of the federate interface specification that carry time stamps should be considered as relevant messages. Such services return message retraction handles needed to identify and retract (i.e., cancel) a message. Most prominent HLA services which fall into these categories are updateAttributeValues (UAV) and sendInteraction (SI).

Other requirements imposed by optimistic synchronization protocols include the need to perform “global virtual time” (GVT) calculations (Jefferson 1985) (especially needed for fossil collection, including the deletion of unneeded system state checkpoints) as well as the need for a mechanism to correctly determine the simulation end condition.

4 CAPABILITIES OF SLX AND ITS HLA-INTERFACE

The “Simulation Language with Extensibility” (SLX) is a discrete event simulation language developed by the Wolverine Software Corporation (Henriksen 1999). The modeling world view in SLX can be characterized as process oriented. As such, it is not based on classical event scheduling. Instead, SLX offers an object oriented modelling paradigm, in which users describe active and passive objects. Active objects have their own algorithmic behavior description (typically, but not necessarily used for modeling items/parts moving through the modeled system). Passive objects have no executable behavior and are typically used for simple resources. More complex resources can have their own behavior as well.

SLX is a layered simulation language, allowing user extensions to the core SLX simulation language (“statements”) as well as extensions through a DLL interface. SLX offers a state-saving and restoration capability originally intended for purposes such as saving a model state after a warmup-period (e.g., for efficiently performing replications in non-terminating simulations).

SLX was among the first CSPs for which an HLA integration was demonstrated. The SLX-HLA-Interface makes use of the SLX DLL interface and provides a wrapper library with an SLX-specific mapping of the original HLA API (Straßburger et al. 1998; Strassburger 2006). Please note that the bi-directional calling convention that HLA implies has to be converted into a uni-directional calling convention, i.e., function calls will always be initiated by SLX, as an SLX model cannot define its own callback mechanism. This is a restriction that virtually all CSPs have to deal with.

The basic principles of the SLX-HLA-Interface are visualized in Figure 1. An interesting aspect of the SLX-HLA-Interface is that it can access SLX objects directly, e.g., when an attribute update is sent or received it can be taken/stored directly from/in SLX memory, removing the need for costly parameter passing conventions and query mechanisms.

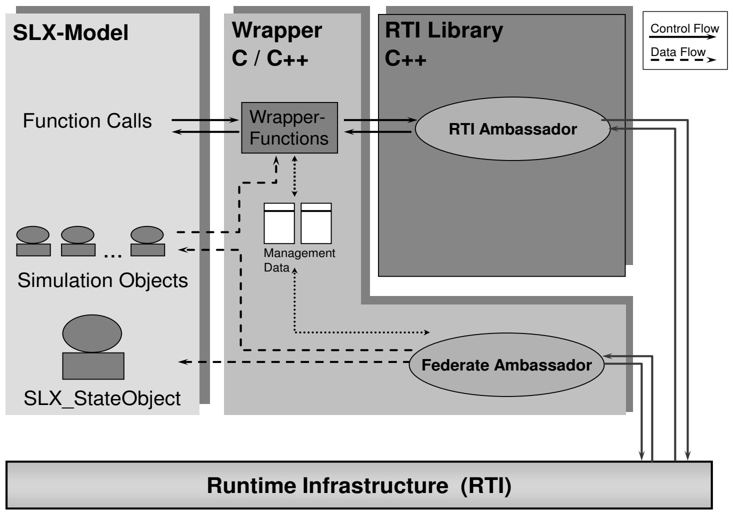

Figure 1: Architecture of the SLX-HLA-Interface.

Traditional (conservative) time advancement with the SLX-HLA-Interface works as follows: Whenever the simulation needs to advance its logical simulation time, a dedicated process ("puck" in the SLX terminology), is activated and a call to either RTI_NextMessageRequest or RTI_NextMessageRequestAvailable is executed. The code of this process is shown below.

```
procedure synchronize () {                                  //conservative approach
  double nextEventTime;
  double grantTime;

  forever {
      nextEventTime = next_imminent_time();                 //get next local event time
      grantTime = RTI_NextMessageRequest(nextEventTime);    //request time advancement
      advance (grantTime - time);                           //advance to grantTime
      …                                                     //process any received messages/updates
      yield;                                                //turn control over to other processes/pucks
  }
}
```

Code Fragment 1: Synchronization loop for conservative synchronization

In the wrapper library, the time advance request calls are translated into calls to the appropriate RTI-Ambassador methods nextMessageRequest (NMR) and nextMessageRequestAvailable (NMRA), respectively. After these methods have been called, the flow of control remains in the wrapper library, which then repeatedly calls evokeCallback until a timeAdvanceGrant (TAG) callback to the Federate Ambassador has been received. Any other callbacks received in the meantime are processed automatically. Any data received in such callbacks (attribute updates, interaction messages) are directly stored into the appropriate SLX objects. Returning control to SLX only after a TAG ensures that any received data is processed at the logically correct simulation time.

5 CONCEPT AND IMPLEMENTATION OF OPTIMISTIC EXTENSIONS

The guiding principle in designing the optimistic extensions for the SLX-HLA-Interface was to make usage of optimistic synchronization as easy as possible for the modeler. With that in mind, we intended to put as much implementation as possible into the wrapper library, following the idea of an external rollback controller put forward by other authors. For bookkeeping tasks, such as keeping message logs of received and sent messages, this was a rather straightforward task, as it simply translated into adding this logging capability to each relevant RTI and federate ambassador service. Functionality also put into the wrapper library was the detection of causality violations and the processing of incoming and outgoing message retraction requests (“cancellations”). With that, it was possible to hide most bookkeeping tasks from the user. The most important concern which conceptually had to be placed on the side of the SLX model relates to the state saving and restoration approach.

Firstly, the general applicability of the state-saving mechanism of SLX for the intended purpose had to be investigated. The “SLXCheckpoint” command can be used to store the complete state of a model in memory. The call returns a unique ID to restore or delete a checkpoint later on. Initial tests with the reference model explained in section 6 showed that a total of approximately 3500 checkpoints could be created very fast, afterwards a slowdown in checkpoint creation was observed. This observation can be most likely attributed to reaching the main memory capacity and the start of swapping memory to hard disk. In any case, a capacity of storing 3500 checkpoints was deemed completely sufficient for the ongoing tests. As the tested models contained a rather small number of concurrently active objects (typically less than 50), this issue must be revisited for industrial strength models. For such models, appropriate techniques (e.g., pruning) should be applied in order to not exceed the model specific capacity for efficient checkpoint storage in main memory.

SLX checkpoints are restored using the “SLXRestore(ID)” statement and deleted using the “SLXReleaseCheckpoint(ID)” statement. It should be noted, that storage and restoration of a models state has some peculiarities, as these activities are (naturally) performed from within the model. This characteristic renders approaches using getState/setState methods as suggested by some authors useless, as they incorrectly assume a CSP checkpoint feature accessible to be called from „outside“ a model.

As a model restoration without any precautionary measures to identify the occurrence of a restoration would lead to exactly the same model execution as previously observed (leading to infinite cycles of restoring over and over again), some part of the model state must be exempt from checkpointing and restoring.

In SLX, this requirement can be fulfilled by implementing an “exempt” class, which can contain user-defined variables and arrays. Code Fragment 2 shows the implementation proposed here. Checkpoint time stamps and their validity are maintained in two arrays contained within this class. These arrays are accessed via a checkpoint ID. This fixed-size approach is required here, as the exempt class cannot contain variable size data such as linked lists, sets, etc. As checkpoint IDs are assigned strictly in numerically increasing order by SLX, two variables (LargestValidCheckpointID and SmallestValidCheckpointID) are sufficient to indicate the range of potentially valid checkpoints in this array. Finding a checkpoint in case of a needed rollback comes down to iterating through the array of valid checkpoints and finding the one which has a timestamp appropriate for the rollback (i.e., typically a time stamp shortly before the causality violation).

```
class Exempt {
    double     CheckpointTimeStamp[10000];
    boolean    CheckpointValid[10000];
    int        LargestValidCheckpointID;
    int        SmallestValidCheckpointID = 1;
    double     LargestValidCheckpointTime;
}
```

Code Fragment 2: Exempt Class for managing checkpoints

With that, the basic prerequisites for state saving and rollback can be considered as fulfilled. The remaining question is about how to adapt the synchronization process in SLX to include checkpointing, restoration, and requesting out-of-order delivery of messages. Towards that, the following extensions of the SLX-HLA-Interface were created. Firstly, the service RTI_FlushQueueRequest was added to the interface, which allows access to the RTI ambassador method flushQueueRequest (FQR). This service is included in the HLA for requesting the RTI to deliver all messages stored in the RTI’s internal queues to the federate invoking this service, despite the fact that the RTI may not be able to guarantee that messages containing a smaller time stamp could arrive later.

Another addition which was conceptually necessary was the inclusion of an RTI_GetNextMessage function for pulling messages from the input message queue maintained in the wrapper library into SLX memory. In traditional conservative synchronization, this function was not needed, as messages were delivered in a push-mode, whenever they arrived. As this could only occur between a NMR/NMRA and TAG call, it was assured that they were processed at the correct logical simulation time. Since FQR was to be used now, this push mode was no longer appropriate. With the RTI_GetNextMessage function, messages received in FQR can now be retrieved one after each other at the correct logical time stamp.

Finally, another function is needed with which the SLX model can inform the wrapper library that it has fulfilled a request to rollback (the need for that is determined in the wrapper library). The function RTI_NotifyRestoration is intended for that purpose. Once called with a time stamp indicating the local time of the model after the restoration, the wrapper library can perform all message cancelations for messages in the output queue that have become invalid. Furthermore, all messages in the input queue that were previously delivered to the federate and must be delivered anew will be marked as undelivered.

With these functions, the conservative synchronization loop from Code Fragment 1 can be replaced by an optimistic synchronization loop as shown below. This replacement can be made completely transparent to the rest of the model, i.e., the model does not need to know if optimistic or conservative synchronization is performed.

The basic idea of the optimistic synchronization loop is very straight forward and follows the principle ideas of the time warp protocol. Initially it is validated if a checkpoint for the current simulation time exists, if not, it is created. Afterwards, it is checked if there is an external event in the federate’s input queue that has to be executed before the next local event. If so, it is retrieved and executed. Otherwise, a FQR request to the RTI is made, flushing potentially more remote messages into the input queue. If there now is an external event to be executed before our local event, it is retrieved and executed. Otherwise the next local event is executed. In case that the FQR call has resulted in a causality violation, the need for a rollback is indicated to the SLX model. The model then prepares for the rollback by identifying the appropriate checkpoint which shall be restored, informs the RTI of the time stamp it will roll back to (RTI_NotifyRestoration) and then performs the rollback.

Some code sections have been omitted in Code Fragment 3. This includes the release of any checkpoints invalidated by a rollback. Please also note that no code for fossil collection based on GVT calculations are included here, the reason for this being explained in section 7.

```
procedure synchronize () {
  double nextLocalEvent;
  double grantTime;
  int ID;

  forever {
    if (myExemption.LargestValidCheckpointTime < time) {                       //create a new checkpoint if required
          ID = SLXCheckpoint( &myExemption);
          … (code for checkpoint management omitted)
    }
    nextLocalEvent = next_imminent_time();                                      //get next local event time

    if (SLX_StateObjectPtr->earliestUnprocessedRemoteEvent <= nextLocalEvent) {   //check for earlier external event
          grantTime = RTI_GetNextMessage();                                     //retrieve next external event
    }
    else {                                                                      //no external events already received
          grantTime = RTI_FlushQueueRequest(nextLocalEvent);                    //request delivery of any message from RTI
          if (SLX_StateObjectPtr->rollbackRequired) {                           //check if FQR triggered a rollback
                … (code for determination of appropriate checkpoint ID omitted)
                RTI_NotifyRestoration(CheckpointTimeStamp[ID]);                 //inform about rollback time stamp
                SLXRestore(ID);                                                 //perform rollback
          }
          else {                                                                //no rollback required
                if (SLX_StateObjectPtr->earliestUnprocessedRemoteEvent <= nextLocalEvent) { //check for new external events
                      grantTime = RTI_GetNextMessage();
                }
                else {                                                    //execute next local event
                        grantTime = nextLocalEvent;
                }
            }
        }
        advance grantTime-time;                                           //advance local clock to appropriate time stamp
        …                                                                 //process any received messages/updates
        yield;                                                            //turn control over to other processes/pucks
    }//forever
}//synchronize
```

Code Fragment 3: Synchronization loop for optimistic synchronization (simplified)

Obviously, the shown synchronization approach does not include any gadgets known from advanced optimistic synchronization techniques such as pruning, sparse checkpointing etc. Also, due to the nature of the SLX checkpoint feature, always a full checkpoint of the model is made, precluding the application of incremental checkpointing techniques. Initial research on pruning techniques has shown no positive effect on performance, as long as checkpoints can be stored in main memory. Future research will certainly have to look at applying pruning depending on main memory availability.

6 CASE STUDY AND PERFORMANCE RESULTS

A small case study was used to test the feasibility of the outlined optimistic synchronization approach and its performance compared to conservative synchronization. The case study is based on a CSP interoperability reference model proposed in (SISO 2010). A bounded buffer entity transfer problem characterized as an IRM A.2 was implemented with two SLX federates. The federates model a simple connected queuing system (Figure 2).

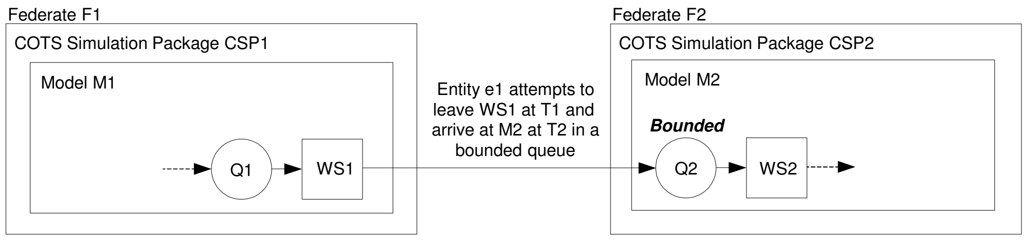
Federate F1
COTS Simulation Package CSP1
Model M1
Q1
WS1
Entity e1 attempts to leave WS1 at T1 and arrive at M2 at T2 in a bounded queue
Federate F2
COTS Simulation Package CSP2
Model M2
Bounded
Q2
WS2

Figure 2: Conceptual model of the case study (based on IRM Type A.2 (SISO 2010)).

Federate 1 produces and processes entities which are subsequently delivered for further processing to federate 2. The entities are sent into a bounded queue in federate 2. As a result, federate 1 must correctly act if the queue in federate 2 becomes full, and block its last station until the receiving queue in federate 2 becomes not full again. While being minimalistic, this simple federation contains all elements that, in the general case, will lead to a zero Lookahead requirement for conservative synchronization. As this represents a worst-case-scenario for conservative synchronization, our initial hypothesis is that the optimistic synchronization will perform better than the conservative synchronization.

Both federates were implemented in SLX (Version 2.3, Build EP 264). The commercial pRTI 5.0.0.0 (Build 1887) from Pitch was used as RTI software. The applied implementation of the SLX-HLA-Interface uses the HLA 1516.1-2010 ("HLA-Evolved") C++ API. In the experiments, both federates and the RTI were executed on a single PC with an Intel i5-3470 Multi-Core-Processor with 3.20 GHz and a main memory capacity of 16 GB. This execution environment represents a realistic working environment when interoperability between two or more CSPs is intended, as those are not typically executed in a truly distributed environment including multiple PCs or workstations.

In the experiments, the behavior of federate 1 was kept constant. It produces entities following a uniform distribution, processes the entities in a workstation and tries to transfer them to federate 2. If the queue in federate 2 is full, it will block processing at its last workstation. To induce different requirements on the synchronization algorithm, the behavior of federate 2 was varied. Two sets of experiments were conducted, each modifying the processing time in WS 2. The first set of experiments limited the length of Q2 to 10 (Figure 3, left side), the second set to 100 (Figure 3, right side). With that, different frequencies for communication between federates were induced, leading to different numbers of rollbacks required (results shown in Figure 4). A set of 10 replications was conducted for each experiment. The figures show the mean values obtained from these replications. In total, 280 simulation runs were executed.

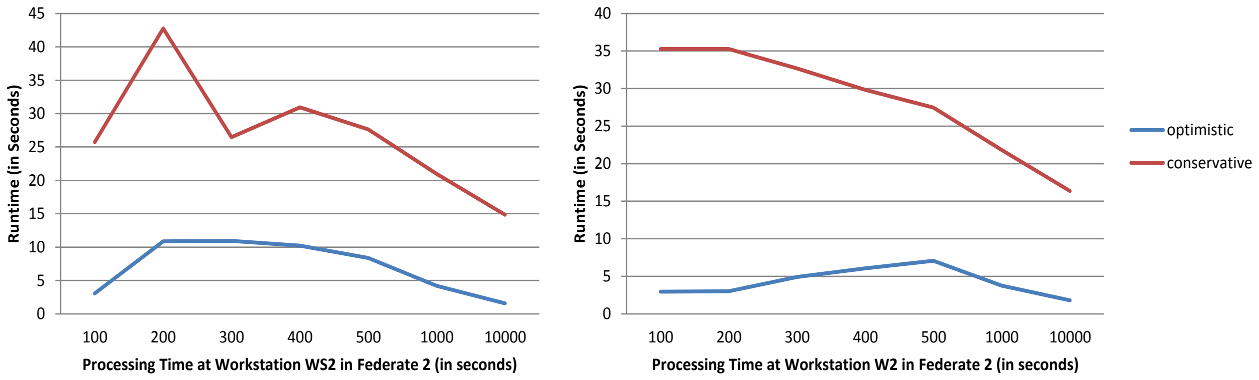

Figure 3: Performance results of two sets of experiments (left: size of Q2=10, right: size of Q2=100).

The performance results show that for this small test case, the optimistic synchronization outperforms the conservative synchronization approach in each instance. The performance gain is greater in case of larger sizes of Q2, as here the models are more loosely coupled and both federates can progress rather independently, until the queue Q2 fills up for the first time.

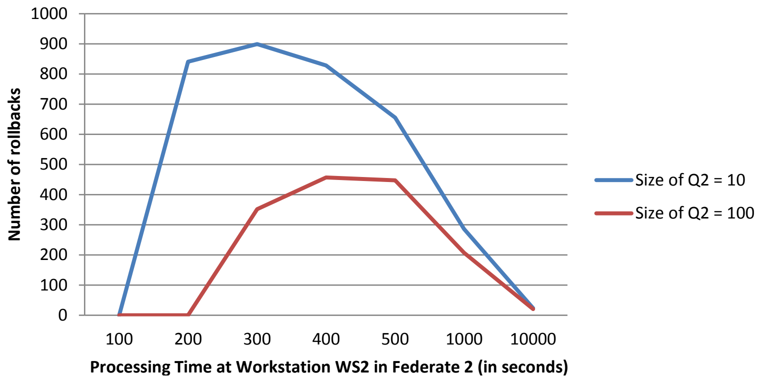

Figure 4: Combined number of rollbacks depending on Q2 capacity and processing time at WS2.

Figure 4 confirms that smaller values of Q2 lead to more rollbacks, as the probability for causality violations is larger. In the case of Q2 = 100 we see that up to a processing time of 200 in WS2 not a single rollback is incurred. This is logically consistent with the production frequency of entities in federate 1 – it is configured in a way that the queue Q2 never reaches its capacity at such a fast processing time.

Summarizing the performance results we see support for our hypothesis that optimistic synchronization outperforms conservative synchronization in the zero Lookahead case. While this result was to be expected, it was important here to validate this insight under the side conditions incurred by a CSP with its given stave saving capabilities and other intrinsic properties. While these findings are very positive, there is obviously more experimentation needed. This will involve both more complex models and fairer conditions for the conservative approach (Lookahead values larger than zero). Ultimately, it will also be necessary to more thoroughly investigate the costs associated with state-saving and rollback and to identify situations where an optimistic approach is preferable to conservative synchronization.

7 GVT AND FQR

The prior sections have knowingly omitted one important feature needed for optimistic synchronization, which is the determination of a value called Global Virtual Time (GVT). GVT in general imposes a lower bound on how far back a rollback can occur and is essential in optimistic synchronization for fossil collection. GVT can be used as a bound for deleting checkpoints and freeing memory: Checkpoints that have timestamps below GVT can generally be regarded save to delete.

Efficient GVT computation has been an important point in PDES research. In the design of the HLA Time Management, the FQR service in conjunction with the TAG service was intended to provide such functionality to optimistic federates (Carothers et al. 1997; Fujimoto 1998; U.S. DoD 1996).

In essence, the TAG service that would eventually be issued in response to the FQR request was supposed to indicate GVT to the model. This is correct, as the time stamp that the RTI passes in a time advance grant is a guarantee that all messages with time stamps smaller than the grant time have been delivered. With that, the return value of TAG can be used as the GVT value to perform fossil collection.

The history of the FQR specification and implementation in the HLA has not been untroubled, unfortunately. Wang et al. (2005) discovered in their work, that the HLA 1.3 standard had a mistake concerning the definition of the return value of FQR(t). This was corrected in the HLA 1516.1-2000 version of the standard.

In theory, HLA compliant RTI implementations should therefore now live up to the visions of “enable[ing] optimistic execution among a collection of optimistic federates” and “federations [that] may include both optimistic and conservative federates” (Fujimoto 1998).

Reality is different. The experiments described above were conducted with the latest RTIs of two leading RTI vendors, namely pRTI 5.0.0.0 from Pitch for the initial experiments and MÄK RTI 4.3 from MÄK for a verification of the observations. Both RTIs failed miserably calculating a correct grant time following the use of FQR. A detailed discussion of the observed problems is given in (Strassburger 2015).

In the setup with two optimistic federates this has rendered it impossible to perform fossil collection and releasing checkpoints. In a different setup, with one federate optimistically synchronized and the other federate conservatively synchronized, interoperation between both federates was completely halted. The symptoms here were that the conservative federate was stuck after its first NMRA-call, while the optimistic federate used FQR to advance through its simulation time. Only after the optimistic federate had resigned would the conservative federate receive a time advance grant to its NMRA call.

While both behaviors seem to be obviously different from what HLA Time Management intended, feedback from the Pitch priority support indicates a different view on the issue. They consider the behavior as a correct interpretation of the HLA standard. The main issue at hand here is, which influence the parameter t passed in FQR(t) shall have on GVT calculation. (Note that the relevant time boundaries in the HLA 1516.1-2010 are referred to as Greatest Available Logical Time (GALT) and Least Incoming Time Stamp (LITS).)

Currently, it appears, that t is not taken into account at all. This is very unfortunate, as the parameter passed to FQR constitutes a conditional guarantee that the federate will not generate a time stamped message before the next TAG service invocation with a timestamp less than the specified logical time plus the current lookahead of the joined federate. Only taking this guarantee into account can the RTI perform a correct distributed calculation of grant times.

Further discussion is needed to clarify, whether a change in the HLA standard is needed to prevent the interpretation embraced by Pitch. Currently, the FQR service description contains the sentence that “A Flush Queue Request service can always be granted without waiting for other joined federates to advance.” This is rather misleading as it implies that the time stamp passed in FQR(t) has only local relevance (as to determine up to which time stamp to flush queued messages). In fact, the phrasing above encourages ignoring t’s function as a conditional guarantee important for other federates.

8 SUMMARY AND OUTLOOK

Our work has shown that optimistic synchronization of CSPs using the HLA is possible and demonstrated a suitable form of implementation. Our case study has confirmed that optimistic synchronization can yield a significant performance increase, especially when conservative synchronizations perform poorly, e.g., due to zero lookahead requirements. Further work in this area is required (e.g., regarding fossil collection and pruning techniques), but is currently obstructed by limitations of current RTI implementations. We have further demonstrated that conservative and optimistic synchronization can be exchanged “plug-in-style” in a CSP, requiring no model modifications.

Our work has revealed that the current implementation of the flushQueueRequest service in the RTIs of leading RTI vendors (Pitch, MÄK) is in conflict with the initial intentions associated with FQR. Future work is intended to clarify these issues. It appears, however, that the current HLA 1516.1-2010 specification leaves room for different interpretations of the intentions of FQR. While on the one hand, the initial time management design document puts a clear mandate on what is intended by FQR, the current specification leaves room for a much narrower interpretation, not mentioning the exact influence that time stamps passed to FQR shall have on LBTS/GALT computation. This issue will be put forward to SISO’s HLA Product Support Group seeking corrective actions either in the next revision of the HLA standard or towards achieving a common interpretation of the current phrasing of the standard.

9 ACKNOWLEDGEMENTS

The author would like to acknowledge the contributions made by Mr. Jan Volf, who contributed to the implementation and provided experimental results.

REFERENCES

Boer, C. A. 2005. Distributed Simulation in Industry, Delft, The Netherlands: Netherlands TRAIL Research School.

Carothers, C. D., R. M. Fujimoto, R. M. Weatherly, and A. L. Wilson. 1997. "Design and Implementation of HLA Time Management in the RTI Version F.0." In Proceedings of the 1997 Winter Simulation Conference, edited by S. Andradóttir, K. J. Healy, D. H. Withers, and B. L. Nelson, 373–380, Piscataway, N.J.: IEEE.

Carothers, C. D., K. S. Perumalla, and R. M. Fujimoto. 1999. "Efficient Optimistic Parallel Simulations Using Reverse Computation." ACM Transactions on Modeling and Computer Simulation 9:224–253.

Ferenci, S. L., K. S. Perumalla, and R. M. Fujimoto. 2000. "An Approach for Federating Parallel Simulators." In 14th Workshop on Parallel and Distributed Simulation (PADS 2000), 63–70.

Fujimoto, R. M. 1998. "Time Management in the High Level Architecture." SIMULATION 71:388–400.

Fujimoto, R. M. 2000. Parallel and Distributed Simulation Systems, New York: Wiley.

Henriksen, J. O. 1999. "SLX - The X is for eXtensibility." In Proceedings of the 1999 Winter Simulation Conference, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 167–175, Piscataway, N.J.: IEEE.

Jefferson, D. 1985. "Virtual Time." ACM Transactions on Programming Languages and Systems 7:404–425.

Lendermann, P., M. U. Heinicke, L. F. McGinnis, C. McLean, S. Strassburger, and S. J. Taylor. 2007. "Panel: Distributed Simulation in Industry - a Real-World Necessity or Ivory Tower Fancy?" In Proceedings of the 2007 Winter Simulation Conference, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 1053–1062, Piscataway, N.J.: IEEE.

Mustafee, N., M. Sahnoun, A. Smart, P. Godsiff, D. Baudry, and A. Louis. 2015. "Investigating Execution Strategies for Hybrid Models Developed Using Multiple M&S Methodologies." In Proceedings of the 2015 Spring Simulation Multi-Conference (SpringSim'15), Alexandria, VA.: SCS.

Mustafee, N., and S. J. E. Taylor. 2006. "Investigating Distributed Simulation with COTS Simulation Packages: Experiences with Simul8 and the HLA." In Proceedings of the 2006 Operational Research Society Simulation Workshop (SW06), 33–42, OR Society, UK.

Pawletta, S., B. Lampe, W. Drewelow, and T. Pawletta. 2000. "HLA-based Simulation within an Interactive Engineering Environment." In Proceedings of the 4th International Workshop on Distributed Simulation and Real Time Applications (DS-RT 2000), 97–102.

Perumalla, K. S. 2006. "Parallel and Distributed Simulation: Traditional Techniques and Recent Advances." In Proceedings of the 2006 Winter Simulation Conference, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 84–95, Piscataway, N.J.: IEEE.

Santoro, A., and F. Quaglia. 2005. "Transparent State Management for Optimistic Synchronization in the High Level Architecture." In Workshop on Principles of Advanced and Distributed Simulation (PADS'05), 171–180.

Santoro, A., and F. Quaglia. 2012. "Transparent Optimistic Synchronization in the High-Level Architecture via Time-Management Conversion." ACM Transactions on Modeling and Computer Simulation 22:1–26.

Simulation Interoperability Standards Organization (SISO). Standard for Commercial-off-the-shelf Simulation Package Interoperability Reference Models (SISO-STD-006-2010), Last modified 2010. http://www.sisostds.org/DigitalLibrary.aspx?Command=Core_Download&EntryId=30829.

Strassburger, S., G. Schmidgall, and S. Haasis. 2003. "Distributed Manufacturing Simulation as an Enabling Technology for the Digital Factory." Journal of Advanced Manufacturing Systems 2:111–126.

Straßburger, S., T. Schulze, U. Klein, and J. O. Henriksen. 1998. "Internet-based Simulation using Off-the-shelf Simulation Tools and HLA." In Proceedings of the 1998 Winter Simulation Conference, edited by D. J. Medeiros, E. F. Watson, J. S. Carson, and M. S. Manivanan, 1669–1676, Piscataway, N.J.: IEEE.

Strassburger, S. 2001. Distributed Simulation Based on the High Level Architecture in Civilian Application Domains, Ghent: Society for Computer Simulation International; SCS.

Strassburger, S. 2006. "The Road to COTS-Interoperability: From Generic HLA-Interfaces Towards Plug-And-Play Capabilities." In Proceedings of the 2006 Winter Simulation Conference, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 1111–1118, Piscataway, N.J.: IEEE.

Strassburger, S. 2015. "Optimistic Synchronization in the HLA 1516.1-2010: Interoperably Challenged." In Proceedings of the 2015 Fall Simulation Interoperability Workshop.

Taylor, S. J. E., S. J. Turner, S. Strassburger, and N. Mustafee. 2012. "Bridging the Gap: A Standards-Based Approach to OR/MS Distributed Simulation." ACM Transactions on Modeling and Computer Simulation 22:1–23.

U.S. Department of Defense. 1996. "HLA Time Management Design Document: Version 1.0.".

Vardanega, F., and C. Maziero. 2000. "A Generic Rollback Manager for Optimistic HLA Simulations." In 4th IEEE International Workshop on Distributed Simulation and Real Time Applications (DS-RT 2000), 79–85.

Wang, X., S. J. Turner, Low, Malcolm Y. H., and B. P. Gan. 2005. "Optimistic Synchronization in HLA-Based Distributed Simulation." SIMULATION 81:279–291.

AUTHOR BIOGRAPHY

STEFFEN STRASSBURGER is a professor at the Ilmenau University of Technology and head of the Department for Industrial Information Systems. He has been involved with distributed simulation since 1996. His web page can be found via www.tu-ilmenau.de/wi1. His email is steffen.strassburger@tu-ilmenau.de. Further biographical details can be found elsewhere in these proceedings.