Integrating the DIS Standards Into a Fully-Immersive Simulation Application

Dioselin Courter, Carsten Neumann, Jan P. Springer, Dirk Reiners, Carolina Cruz-Neira

University of Louisiana at Lafayette

Abstract

Fully-immersive applications in a distributed environment play an increasing role for training and simulation related areas. The Distributed Interactive Simulation protocols were created to ensure interoperability across participating sites. However, until recently the deployment of applications based on this set of standards was dependent on either licenses from commercial vendors or the project-internal implementation of the specification. With the availability of open-source implementations a larger community has now access to a reusable software infrastructure making these protocols available.

We present an application framework for distributed, immersive simulations based on Distributed Interactive Simulation protocols. Our system provides components for input device processing, simulation of physical behavior, dead reckoning, and scene graph rendering of user-defined scenarios. It includes a local data-synchronization mechanism, which enables the simulation to run on a cluster of graphics workstations. The application framework is highly customizable allowing the creation and execution of exercises that are suitable for diverse research objectives.


1 Introduction

Communication protocols are a key issue for the design, implementation, and deployment of collaborative virtual environments. Applications running on a set of machines in geographically dispersed locations but interacting within the same virtual environment must agree on a common description of the world, its entities, and events. Furthermore, these machines also need to agree on how to handle global events within the shared virtual world. The Distributed Interactive Simulation (DIS) is a government and industry initiative which defines practices and structures in the context of collaborative training and simulation applications (including virtual environments) that do not rely on a centralized server for state synchronization. It provides interoperability regardless of platforms and operating systems in which these applications are developed and deployed [IEEE:1278.1:1995]. DIS has produced several standards and recommendations encompassing application protocols, communication services and profiles, exercise management and feedback, validation and verification mechanisms, as well as fidelity-description requirements.

At its core DIS prescribes a set of data abstractions as well as evaluation mechanisms targeted mainly at the simulation of military training exercises. The basic idea behind DIS is that exchanging state or events within the virtual world between all participating sites should enable each participating party to recreate external events as well as to react on them and provide appropriate feedback which is also distributed to all other simulation participants. This is achieved by defining data units that encapsulate common resources in training and simulation. Loss of data packets over the network as well as time-synchronization issues are handled by a defined set of mechanisms that allow applications to locally simulate known entities until new state arrives from the global simulation. A DIS simulation has no central authoritative server, and only requires local applications to implement the handling of entities necessary for their role in the simulation.

There have been previous experiences with integrating the DIS communication protocols for immersive virtual environments (e.g., [McCarty et al. 1994; Kuhl et al. 1995]). Our work has two additional requirements not addressed before: representation and synchronization of human interaction as well as data distribution for graphics clusters that control multi-display systems at each participating location in the virtual environment. We developed an application in the context of assessing and evaluating stress factors of dismounted soldiers using immersive projection displays in combination with an omni-directional treadmill [Darken et al. 1997; Courter et al. 2010]. The application required the development of software infrastructures for creating and testing scenarios, evaluating subjects, and supporting data analysis of past experiments. This context clearly motivated the choice of DIS as the distribution layer in our work. However, while DIS is originally grounded within the training and simulation community we also see much broader applications of our approach. Our application framework can be utilized for collaborative immersive simulations in other fields such as training in industrial settings (e.g., power plants or oil rigs), psychological experiments involving avatar interaction, or even distributed artistic performances, to name only a few of the more obvious candidates.

An additional advantage of using an industry standard for distributed simulations is the ability to immediately integrate with large scale, existing DIS exercises. The underlying framework only needs to be compatible at the protocol level, i.e. implementation details can be handled by local application instances. The recent availability of DIS implementations as open-source software makes it possible for the the general research community to incorporate it in different projects.

In the following we will first review previous and related work and discuss the structure of DIS in detail. We then describe our application’s architecture and the integration of the DIS protocol into it. This is followed by a short description of the application prototype itself and a discussion of experiences related to the integration of DIS in our application. We conclude with a summary and a discussion of possible future work.

2 Related Work

Designing and building distributed virtual environments (VE) is one of the oldest research topics within virtual reality research. Early designs explored the usage of operating-system approaches (e.g., VEOS [Bricken and Coco 1994]) or system inherent shared spaces (e.g., DIVE [Carlsson and Hagsand 1993], MR Toolkit [Shaw et al. 1993]). AVIARY [Snowdon and West 1994] introduced ideas for sophisticated mechanisms which allow to distinguish between local and global resources and their impact on updating their representations for the participants. MASSIVE [Greenhalgh and Benford 1995] further extended these ideas and enabled the use of large-scale environments as well as geographically distant sites for participa-
prototyping. Other designs rely on the consistent distribution of input-state changes to all participating nodes (e.g., VR Juggler [Just et al. 1998], Lightning [Blach et al. 1998]). Avango [Tramberend 1999] builds upon a distributed scene-graph paradigm to enable collaborative as well as render distributed VEs. While all of these systems do support applications embodying distributed simulations, they fall short on supporting a standard protocol for communicating state across the distributed system. The DIS standards, employed in our work, have been developed, in part, to specifically address this.

DIS is in widespread use within the training and simulation community. Early applications concentrated on supporting scenarios for command and control exercises [Landweer 1994]. These applications usually employ an abstract 2D visualization and user interface. Also, early interest arose to deploy the DIS standards for non-military use [Fitzsimmons and Fletcher 1995]. Using DIS within fully-immersive environments seems to be usually restricted to simulations that allow for training of complex systems such as flight simulators; McCarty et al. [1994] provide an early example.

Knerr and Lampton [2005] describe an experimental setup which deploys a range of interfaces, from desktop operated to immersive projection-based display and head tracking, for evaluating training transfer for urban warfare scenarios. The setup tested single soldiers as well as groups of soldiers in collaboration and provided commanding officers with information for tactical planning. The DIS protocol was used for providing both the information for the simulation as experienced by individual soldiers in a projection-based display setup as well as the abstract situational view for tactical planning and execution control. There has probably been more work accomplished using DIS, however projects using this standard are often confidential so we do not have access to their details and are unable to discuss this aspect further.

DIS can be seen as a predecessor to the High Level Architecture (HLA) [IEEE:1516.0:2000; IEEE:1516.1:2000; IEEE:1516.2:2000] as well as the Test & Training Enabling Architecture (TENA) [US Dept. of Defense 2002]. HLA and TENA explicitly define software abstractions and application programming interfaces (API). HLA utilizes an object-oriented approach to define services used by simulations (referred to as federates) and scenario objects along with their interactions. In contrast to DIS, which provides specific definitions of entities and their behaviors, HLA contains an Object Model Template (OMT) with rules on how to define objects. It thus emphasizes a more generic simulation architecture. This is an advantage in terms of supporting a wide range of scenarios, but at the same time makes interoperability more difficult. DIS is plug-and-play, i.e. once an application integrates the protocol it will be able to interoperate with existing simulators (even if they serve different purposes). On the other hand, all HLA simulations participating in the same exercise must agree on the same Federate Object Model (FOM) in order to be able to interact. Another feature of HLA is the support of simulations which run faster or slower than real time, which is not possible with DIS. However, this is not a problem in our context as we only support real-time simulation. Finally, because HLA as well as TENA require specific link and compiler compatibility, we believe the efforts that must be invested for initially setting up a software infrastructure that supports either one of them are larger than what is required for supporting DIS.

3 Distributed Interactive Simulation

Distributed Interactive Simulation is a set of standards that define practices and structures for collaborative training and simulation applications. It is based on experiences from the Simulator Networking (SIMNET) project, initiated by the US Defense Advanced Research Projects Agency (DARPA) in 1983, exploring the deployment of real-time applications across remote sites [Miller and Thorpe 1995]. SIMNET conceptually provided a large-scale, networked, interactive simulation infrastructure to create synthetic environments for

![Figure 1: Format of a DIS Protocol Data Unit (PDU). The packet header has a fixed length of 12 bytes while the body is of variable length. Protocol version, PDU type, and protocol family are values from enumerations defined in a separate DIS support document [SISO:REF010:2006]. Exercise id allows for PDU traffic between different simulation environments in the same network. The PDU Type field is used to correctly decode the message body. The timestamp field contains the time the PDU was created.](image)
Protocol Version is an 8 bit enumeration value.

Exercise ID is an 8 bit unsigned value specifying the exercise the PDU belongs to, allowing for multiple concurrent environments on the same network.

PDU Type is an 8 bit enumeration value specifying the PDU type contained in the message body.

Protocol Family is an 8 bit enumeration value specifying the protocol family.

Timestamp is a 32 bit unsigned value that specifies the time when the PDU was generated; it represents units of time passed since the beginning of the current hour, with the least significant bit indicating whether the timestamp is absolute, i.e., the simulation uses exact UTC time, or relative, i.e., the simulation uses its own clock.

Length of the PDU is a 16 bit unsigned value specifying the length of the whole PDU including its header.

Padding is used to add extra bits to ensure the header has always the same length; 12 bytes in the latest version of the standard.

The Live Entity protocol family adds the following field (cf. figure 2):

Subprotocol is an 8 bit enumeration value specifying the subprotocol to be used to decode the PDU. A value of zero is reserved for PDUs complying to the published standard. [IEEE:1278.1a:1998]

By using the Subprotocol field in a Live Entity PDU with a value other than zero it is possible to extend the protocol for a specific simulation without breaking interoperability among all (participating) simulations.

To ensure diverse applications handle PDUs the same way, DIS prescribes several architectural principles which simulations must follow to be standard compliant:

► There is no central authority server controlling the simulation, i.e., autonomous applications are responsible for maintaining the state of one or more entities.

► Every simulation is responsible for always transmitting the current state of its entities, also known as the ground data; other simulations must decide whether the entity is relevant (e.g., visible) or not.

► The local effect and perception of remote events and entities is locally determined by each simulation.

► Dead reckoning algorithms are used to reduce communication traffic. Every simulation maintains an accurate dynamics model of its entities plus a dead reckoned model that uses extrapolation methods to predict position, orientation, etc. The simulation issues update messages only when these two internal models differ by a pre-defined threshold.

An additional requirement in the design of DIS was to allow for different types of applications to be able to join the same exercise (within the same location or from remote sites). Applications may respond to different objectives (e.g., active training vs. silent evaluation) or different command chain roles (e.g., a trainer application may have different permissions than a trainee application). The simulation address record was defined to uniquely identify every application. It contains a 16 bit unsigned value corresponding to the site ID and another 16 bit unsigned value corresponding to the application ID.

The simulation address record is always part of the entity identifier. Every unique entity is described by its simulation address plus a 16 bit unsigned value for the local application entity ID. The entity ID is usually part of all interaction messages. One example is the collision PDU from the Entity Information/Interaction protocol family, as shown in Table 1, which is broadcast when the application detects a rigid collision between a local entity and a non-local entity. The message body contains identifiers for both entities, allowing all session participants to correctly represent and simulate the event.

Even though the specification for DIS originated from a specialized field of simulation, it provides a choice for distributed VE applications in general. Just the Live Entity and Entity Information/I
action protocol families would suffice for other non-military simulation applications. Furthermore, the extendability of the Live Entity protocol family provides a way of adding new features.

4 ARCHITECTURE

We developed an application framework for distributed, immersive simulations based on the Distributed Interactive Simulation protocols. Its multi-threaded and modularized structure was designed to respond to the real-time as well as collaborative requirements of these simulations.

Figure 3 shows the composition of our system. Local input processing provides the abstraction layer required to support executing the simulation with different input devices. Asynchronous modules for dynamics behavior and DIS communication perform the simulation of physical behavior in the environment. The DIS module also generates the corresponding PDUs that are sent to other participants (or remote sites) through the collaborative session module. The dead-reckoning component conducts extrapolation of object behavior in order to handle network latency and minimize network traffic. A master DIS simulator is running at each site participating in the collaborative session, while slave simulations on every node of the (local) graphics cluster use data synchronization mechanisms for cluster-global data to correctly render the simulation state each frame.

4.1 DIS Integration

Until recently, using the DIS standard for distributed virtual environments required the availability of a commercial implementation and license. This exhibits problems for projects that expect a long life time or must be implemented and deployed without being locked to a certain vendor. Additionally, the release cycle of commercial licenses may not be as fast as in research environments where frequent changes in hardware or software platforms are common. One solution to these issues are free and open-source software implementations in general and for DIS in particular.

Several open source implementations of the DIS application protocols are available. KDIS [KD] is a C++ implementation of DIS including PDUs, enumerations, symbolic names, and constants. Every PDU is a subclass of the common PDU header class and includes methods for encoding and decoding the PDU into and from a byte stream. As of version 1-10-0, KDIS provides approximately 60% of the application protocols; the Live Entity family was not available in that version. OpenEagles [OpenEagles] is a C++ framework for developing distributed simulations and, among other components, provides interfaces that support DIS. It is possible to use OpenEagles’ DIS library without the rest of the framework. However, only very few PDUs are currently defined. DIS/x [Web Simulations] is a library that is part of ACM, a multiuser flight-combat simulation for local networks written in C. It provides an interesting feature not available in other implementations: a simulation management server to automatically generate site and application IDs. However, DIS/x also supports a minimal subset of the protocols while its development seems to have stopped.

Most DIS implementations manually encode the DIS protocol set, i.e., they provide a set of classes that encode the protocol structure that has been designed and implemented by hand. Adding new functionality or supporting new families requires programmers to individually modify each class, which is not a small effort since the latest version of the standard defines 67 PDUs grouped in 12 families. Additionally, it results in duplicated code because the structure of PDUs from different protocol families is similar in many aspects (e.g., they include the same record types or define the same fields). In contrast to the DIS implementations mentioned before, the Open-DIS [McGregor et al. 2008; Open-DIS] open-source implementation provides a different approach. It contains a code generator that reads an XML document describing the DIS protocol and produces Java, C++, and C# source code (skeletons). Adding new families (or features) in Open-DIS is achieved by inserting the appropriate information in the XML document and executing the code generator. This flexibility was the main reason we chose Open-DIS for our development.

Similar to other implementations, Open-DIS defines an inheritance hierarchy for PDUs. The common header fields are grouped into a super class, from which every protocol family is derived. Each protocol family is then a super class for a set of specific PDU types. Open-DIS uses an XML dialect defined by its developers to create the abstract description of classes and their fields. The dialect is simple and allows the definition of classes as well as their inheritance and attributes (cf. listing 1).

5 APPLICATION PROTOTYPE

We developed a software prototype for creating, testing, and deploying virtual scenarios. This prototype consists of an editor module and an immersive simulator module.

The editor module is based on a standard 2D GUI design. It allows for importing and manipulating models to create a scene. Furthermore, dynamic changes in the scene based on event triggers and semi-autonomous behavior of virtual actors are also supported. These changes are transmitted at runtime to the immersive simulator module through a network connection.

The immersive simulator module is implemented as a VR Juggler [Just et al. 1998] application and can be run in a number of setup configurations; in our case a CAVÉ-like display system [Cruz-Neira et al. 1993] in combination with an omni-directional treadmill [Courter et al. 2010] for navigation is the final target (cf. figure 4). The display system’s outputs are created by a cluster of workstations. Synchronization of the cluster is based on VR Juggler’s cluster extension [Olson 2002] which automatically distributes input-device data and allows for application-specific data (simulation results in our case) to be synchronized across the cluster nodes.

This two-fold approach, i.e., a scenario assembly in a 2D GUI and actual exploration in an immersive environment, serves to satisfy the process of creation, testing, and final deployment of a scenario. First, the initial scenario is created in the editor module and populated according to the user’s needs. By using a simple application-specific protocol as a distribution infrastructure for commands, the editor module is able to publish content as well as incremental changes.
Figure 3: Framework architecture. An input module translates device data into meaningful parameters for the dynamics simulation. The dynamics and DIS-communication module performs physics simulation of the world as well as generates DIS messages for the collaborative session synchronization across remote-network participants. The dynamics module communicates with a dead-reckoning component which extrapolates objects’ behavior. A cluster synchronization component provides mechanisms for synchronizing global data between nodes in a graphics cluster. The scene-graph rendering module is responsible for updating the visual representation of the world.

Figure 4: The immersive simulator module of the application prototype running in a CAVE™-like display system setup around an omni-directional treadmill.

of the scenario to a running instance of the simulator module. This allows immediate review of scenario authoring changes in the immersive setup. Once the scenario is stable, its definition is saved so it can be repeatedly deployed in the immersive simulator (e.g., for evaluating test subjects).

6 Discussion

The integration of the DIS application protocol standard exhibits interesting properties to be exploited for distributed fully-immersive VEs. Namely the consistent distribution of simulation data across a collaborative setup but also the use of these updates within a single graphics cluster driving an immersive multi-display installation.

The DIS application protocols deal with communicating world changes once the session has started. They do not prescribe how static scene content, such as geometry for entities and site IDs, is distributed to the particular simulation sites. Session management protocols and recommended practices are part of the standard and deal with how different sites should agree on and coordinate the distribution of this kind of information. In the context of a DIS session with fully-immersive VE participants, a different approach could be to extent the Live Entity protocol family with a subprotocol that enables the distribution of universal resource locators connected to an entity’s simulation address.

DIS requires each participating application to constantly simulate entities owned by the application as well as a dead reckoned model of all entities (including its own). As described earlier in our work, a dynamics engine is used to simulate physical behavior of world objects at each simulation node. The dead-reckoning requirement usually is satisfied by using less detailed entity representations. However, the current state of available hardware with multiple CPU cores above four, eight, or even 16 as well as an abundance of local memory will allow for running several dynamics simulations targeting different simulation goals (e.g., one for the local simulation and one for dead-reckoning).

The integration of the Live Entity protocol family is a key issue in our application context. However, Open-DIS originally did not implement this protocol family. We were able to achieve this by adding the appropriate description of the protocol in terms of the Open-DIS XML protocol description. The Life Entity protocol family PDUs exhibit a slightly different header format, so the original PDU super class needed to be subdivided in two subclasses: one for Life Entity PDU header containing the extra subprotocol field and the classic PDU header for all other families. Once this information was added to the XML description, the code generator automagically creates all necessary classes. Figure 5 shows the modified inheritance diagram we generated.

By modifying the code generator source, it is possible to easily adjust or implement new functionality across all PDU families in Open-DIS. One such case was the addition of copy constructors and assignment operators for all PDU classes. We also modified
We presented an application framework that integrates the DIS while they are all linked through the deployment of separate distribution of local synchronization data, i.e. distributing global asset in itself that can be part of the simulation project and manually tweaked.

7 Conclusions

We presented an application framework that integrates the DIS standard into a fully-immersive distributed virtual environment. The DIS application protocol is used for the consistent distribution of simulation data in the virtual environment. While our work is not a framework in the conventional sense, our application framework does support a wide range of scenarios.

Our use of DIS may advance the development of distributed virtual environments in several ways. First, by providing a new way of achieving interoperability between participating sites that do not need to run the same software infrastructure. By only requiring the ability of sending and receiving DIS messages the actual software tools can then be chosen locally. Second, the use of the DIS protocols for distributing actual environment state to all participants greatly simplifies the amount of effort that must be spent otherwise in achieving this goal. Also, using DIS would allow for concisely separating the distribution of application state from the distribution of graphics resources.

7.1 Future Work

The use of the DIS application protocol for distributing global simulation state was already discussed. While our experiences with this approach are very positive we are aware that potential problems exist. The distribution of simulation related data within the global virtual environment exhibits a (slightly) different behavior than the distribution of local synchronization data, i.e. distributing global simulation events is usually regarded to be high-frequency but low-bandwidth while updating data in a local graphics cluster usually requires high bandwidth but also high-frequency updates. The natural design choice here is the deployment of separate distribution APIs that match the bandwidth-frequency requirements of the individual components. However, it can be speculated that the constant growth of bandwidth for wide-area networks may have diminished this distinction, but ultimately this needs further attention by the community to be better understood.

As shown in figure 3, the internal modules of our framework do not communicate through DIS. It is possible to use the protocol for local communication as well. The input processing module could generate an entity state PDU corresponding to the local participant; this would be a Time Space Position Information PDU from the Live Entity protocol family. The dynamics and real reckoning modules would also communicate through entity state PDUs. Finally, the scene-graph rendering module then needs to capture the generated PDUs from the dead reckoning and dynamics module and transform the PDU's information into the appropriate representations for updating the scene graph. This would decouple much of the framework's modules and open a path for migrating them to separate processes that (may) run on different workstations.

We also need to complete the implementation of the Live Entity family in Open-DIS. Specifically, PDUs in this protocol family usually contain optional subfields that reduce the total packet size when not present. However, the code generator needs to be modified to allow PDU constructors, attributes for PDU fields, and methods for marshalling and unmarshalling PDUs to and from byte streams.

It is our hope that this initial work may lead to a renewed effort in building large-scale virtual environments that are both rich in features as well as inter-operable across diverse systems and sites.

Acknowledgements

This work was supported in part by the 3rd Generation Omnidirectional Treadmill Project under contract with the US Army Research Lab, Human Resource and Engineering Directorate, Aberdeen Proving Ground, Aberdeen, MD. We would like to thank the Open-DIS community for developing the Open-DIS software as well as for making it available to the general public.

References


