Functional Mock-up Control with Multi-player Online Game Protocol

Mikio Nagasawa Shinichi Ishizuka
Cybernet Systems Co., Ltd.
Kanda-neribeicho 3, Chiyoda-ku, 101-0022 Tokyo

Abstract

To meet the requirement of collaboration in the system-level simulations of multi-domain models, a distributed online co-simulation environment, COSIMAS, is designed and implemented. It works as a multi-user FMI co-simulation master. Based on the online gaming protocol UPC, the environment adopts dynamical control and communication architecture for sharing co-simulation models status information. By application of Flash visualization technologies, an interactive simulation environment in the web browser is implemented. This paper introduces the main characteristics and architecture of COSIMAS.

Keywords: FMI; online; co-simulation; multi-domain; games; cloud; MapleSim; Simulink

1 Introduction

For designing the collaborative online simulation architecture, one of interesting interfaces is "Functional Mockup Interface (FMI)" [1] for multiphysics, multi-timescale co-simulations. The FMI defines a standardized interface to be used in computer simulations to develop complex physical systems. When we connect N simulation codes, the number of program interface can be reduced from $O(N^2)$ to $O(N)$ using the unified standard interface. The superposition of "Multi-X" co-simulation might be possible, which would differ from "serial connection" and "parallel coupling". Those are applicable for any "Multi-X", such as multi-method, multi-technique, multi-tool and multi-expertise, and so on. We believe that this new integration M&S methodology will be in huge demand in the near future.

As online simulation standards, we have considered the distributed application protocols: DIS, HLA, and UPC. The Distributed Interactive Simulation (DIS) is an IEEE standard [2] which describes the format of the packets and their protocol that should be exchanged between distributed simulation entities. High-Level Architecture (HLA) is a multi-domain interoperability architecture for distributed computer simulation systems to interact, communicate, and synchronize with other computer simulations [3]. Union Platform [4] is a development platform for creating connected applications such as realtime multi-player games.

A Modelica web simulation environment was presented [5], on which users can perform system simulation and analysis in the browser. Users can complete the experiment connecting the custom components, and the results of the experiment can be obtained after the simulation. However, the system was designed for engineering analysis using client server architecture with limited number of users. In order to share the simulation environment with larger number of softwares and users, this paper describes a general online simulation game environment for web-based modeling and simulation engineering.

This paper is organized as follows: Section 2 presents the system architecture of multi-player online game. Section 3 explains the synchronization mechanism needed for co-simulation consistency. In Section 4 we describe the case studies performed using our online co-simulation environment. We present our conclusions and directions for future work in Section 5.

2 Online Game Architecture

COSIMAS is a co-simulation framework that links FMU simulators of "multi-X" and physical systems. COSIMAS allows engineers to develop simulations of different subsystems in the most appropriate tool for their domain and then co-simulate the larger system by running the distributed subsystem simulations.

Online interactive co-simulations support the communication and collaboration of people through the sharing and manipulation of the Application Data Unit (ADU) via the Internet. The link with COSIMAS will

help FMU users accelerate the product development process.

COSIMAS has a distributed architecture where each user runs a part of the application. This has the distinct advantage that the application can be deployed in a lightweight fashion, without relying on a supporting server infrastructure. But at the same time, this peer-to-peer architecture raises a number of challenging problems: First, application data needs to be distributed among all FMI master/slaves. Second, consistency control mechanisms are required to keep the distributed application data synchronized.

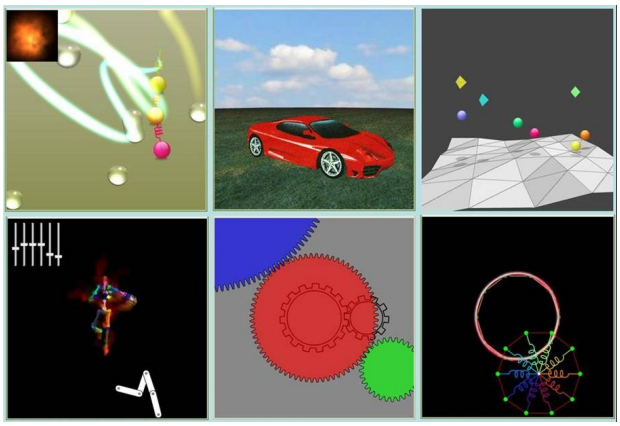

Figure 1: Online game libraries to be functional mock-up units.

We propose to exploit the knowledge of this trade-off by voluntarily increasing the responsiveness of the application despite the short-term inconsistencies. Every player has a “good-enough” view of their co-simulation environment. as shown in Figure 1.

2.1 Protocols

We use UPC protocol for data communication and implement FMI co-simulation call back. COSIMAS is a simulation framework based on a internet communication protocol architecture for multi-physics simulations at all levels of abstraction that provides a collaborative environment for engineers for designing, simulating and validating complex systems. The complete co-simulation framework provides a development platform that supports native and non-native simulation environments, a test platform that integrates test and measurements monitors, and a verification platform that supports co-simulation between different abstraction levels and non-regression of model functionality along the design flow. COSIMAS uses FMI for model exchange and co-simulation of two or more simulation tools, where each instance of each tool can be treated as a black box with no need for translation.

The processes and relationships are defined in the following:

1. Select a co-simulation room: user selects the online room that serves as a simulation stage for the joining master/slaves. This co-simulation room can be defined by the first master or slave module. If there is neither master nor slave module, the room will be cleared and reset.

2. Select and load simulation models: user selects an instance model to be communicated to start co-simulation. The loading is an iterative process.

3. Select components and ports of models: User selects models and their respective input and output ports. These selections are stored as the ADU and the user select them for visual monitoring.

4. Select visualization modes: The modeler is given the choice of viewing and event-listening mode.

5. Execute simulation : Without explicit event of co-simulation start, simulation clock is advancing. User starts simulation of the model. If any model component is selected to be viewed, the execution of the model is displayed as the animation of the model components.

As shown in Figure 2, there is a standard interface between master (controller) and slaves (solver): the master controls a solver using Flash functions, UPC protocol, and external .dll.

The UPC protocol functions are as follows:

```
SEND_MESSAGE_TO_ROOMS
SEND_MESSAGE_TO_CLIENTS
SET_CLIENT_ATTR
JOIN_ROOM
SET_ROOM_ATTR
CREATE_ACCOUNT
LOGIN
SYNC_TIME
... and 150 more
```

The FMI master/slaves and chat users can share the online root and exchange the messages.

2.2 Graphical User Interface

Visualization function should be platform independent: Only methods that are compatible with the FMI

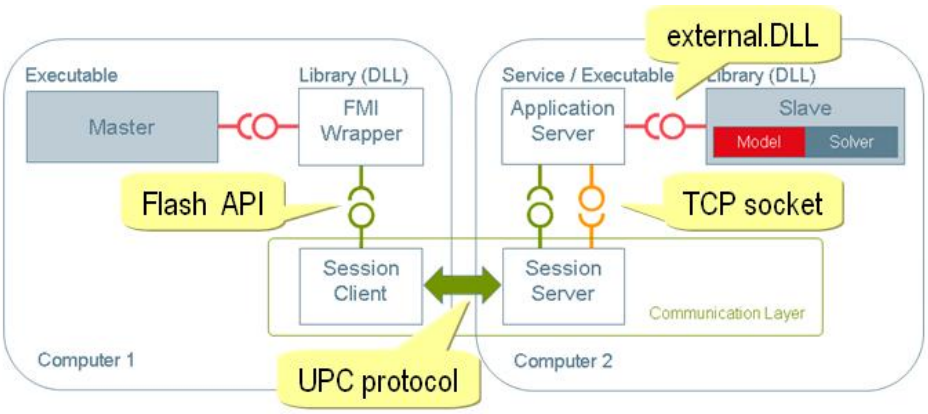

Figure 2: Implementation of FMI communication layer

specification should be used. This rules out the development in form of an extension to an existing platform.

The FMI model must contain a port of ADU sharing, which allows access from the Flash API. On the other hand, extensions which cannot be expressed in FMU need to be implemented in a language that is supported through FMU's external function interface.

The second important requirement was loose coupling between the simulated model of C/Fortran and the visualization tool of Flash. While with the choice of C as implementation language, several options for accessing rendering-tools exist.

The viewer usually requires a lot of additional features (user interface, inputs, file management). Providing those features to a FMU model in a platform independent implementation would require lots of additional work for each visualization.

So instead of directly linking the Flash API functions into a FMU model, we chose to use Flash Union communication (UPC). Typical examples of UPC applications are networked computer games and distributed virtual environments. That way, simulation as well as visualization can run as dedicated processes while sending respectively receiving messages. Any visualization needs to implement a common communication interface. Because visualization should not influence the simulation results, other communication ADU between simulation and GUI event should be established. visualization is unidirectional. Since the simulation does not expect any messages from the visualization, the communication can work synchronously. This also fits into the event-driven modeling style of FMU.

With the design decisions settled, the first task was to implement an extensible, synchronous, platform independent UPC layer in FMU. we have to wrap UPC solutions around it's C-interface. Flash allows sending of arbitrary messages as internet objects as shown in Figure 3. Message objects can be allocated, equipped with parameters and send over a connection. In case a faster solution is needed, Flash should be trivial to port to whatever UPC mechanism seems appropriate. Depending on the scene, this might yield significant runtime and memory improvements over methods like the Multi-Body visualization.

3 Synchronization Mechanism

In order to increase the time integration accuracy of co-simulation, variable communication step sizes were investigated [6]. In a serverless architecture, synchronization must be introduced to make sure that the state displayed by each FMI master/slave is consistent. Late-join slaves need to be initialized with the current application state before they are able to participate in a collaborative session[7].

For dynamical group communication, we adapt the network protocol UPC for the standardized communication of online interactive co-simulations.

COSIMAS supports dynamic initialization of co-simulation stage where master/slaves may join and leave at any time. But a participant joining an ongoing session has missed the ADU data that has previously been exchanged by the other session master/slaves. It is therefore necessary to let the late-join slave know

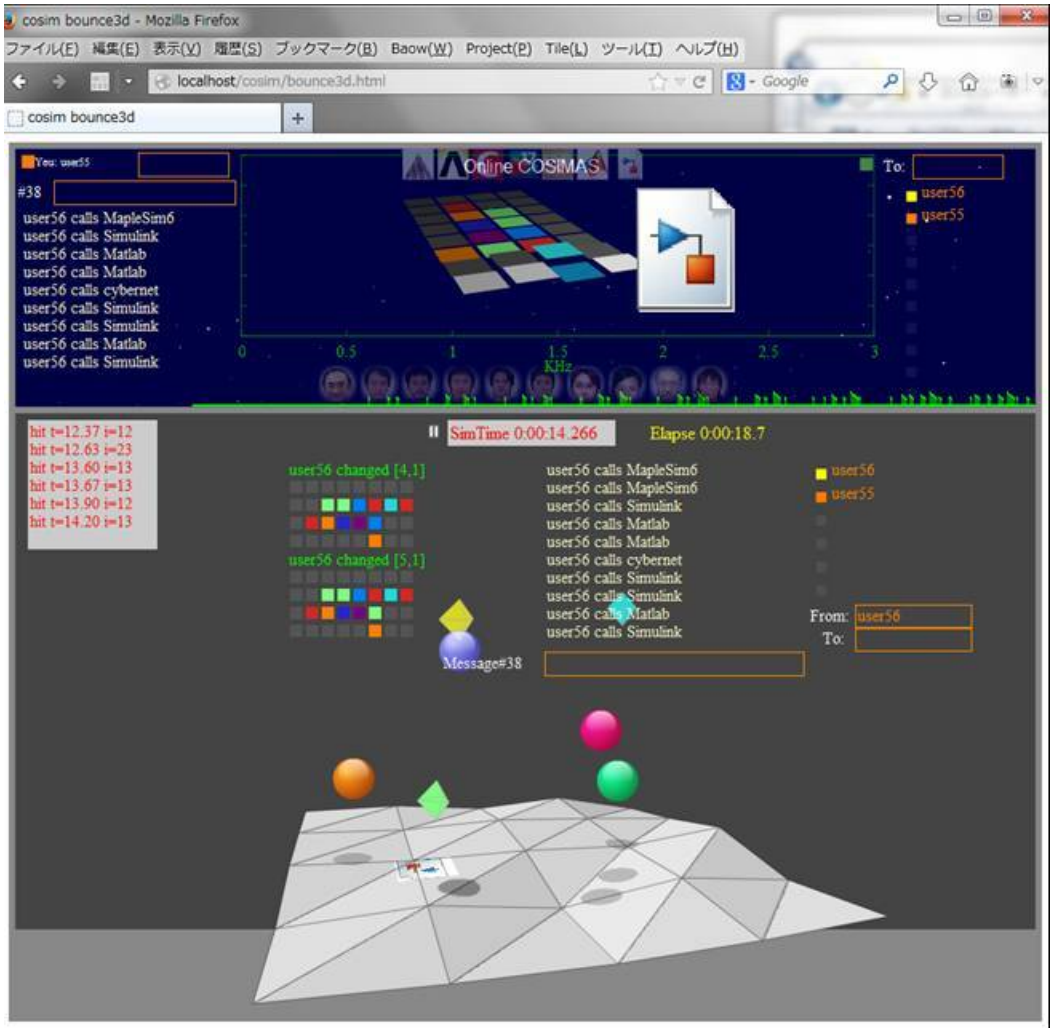

Figure 3: Online COSIMAS shown in browser

the current shared state.

In general, it is not necessary for an application instance to be initialized with the entire shared state. A prerequisite for such a partial initialization is that the application's state is partitioned into independent objects. For each object, a late-join client can then decide when the data for that object should be requested, using its application-specific request.

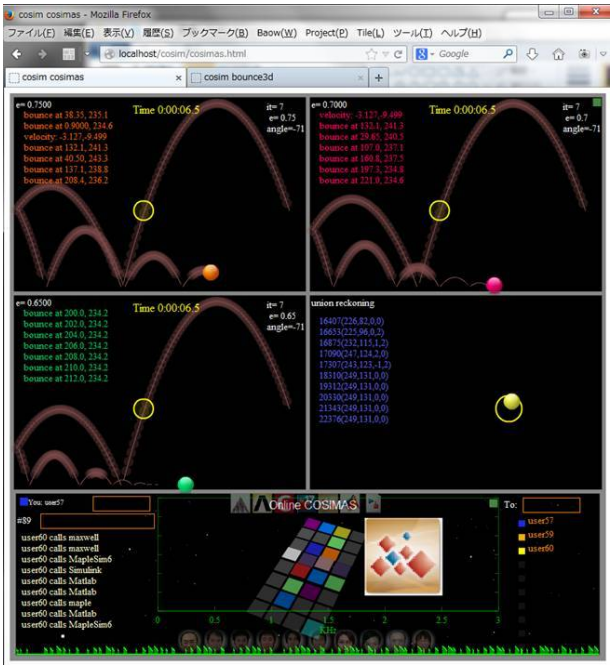

Figure 4: Synchronized bouncing ball control.

The Application Data Unit(ADU) is a chunk of data manipulated by the online game application.

- All ADUs issued at the same time by various FMI master/slaves are computed together to evaluate the state of the co-simulation.
- All the co-simulation master/slaves can display the shared co-simulation state simultaneously.

In COSIMAS, time is divided into fixed length periods and a bucket is associated with each period. All ADUs received by a player that were shared by UPC during a given period are stored by the receiver in the bucket corresponding to that interval. At the end of every bucket interval, all ADUs in that bucket are used by the FMI solver to compute its local and global states. Buckets are computed in 60 fps after the end of the sampling period during which ADUs have been issued (500ms is the ping/alive delay). Figure 4 shows the interactive co-simulation trajectories. The bucket mechanism is used to reduce network jitter effects in packet audio information. In other words, to compute a new global state, a FMI solver computes all the ADUs available in the current bucket.

The bucket frequency defines the rate at which a new game state is computed and displayed. Since human vision perceives smooth motion when the frame rate exceeds 30 frames per second, we have chosen to compute 30 buckets per second. The bucket frequency is a receiver application parameter that should not be influenced by network parameters. With COSIMAS's current settings, the ADU transmission frequency being equal to the bucket frequency, there should be one new ADU message per slave at the time a bucket is processed.

To deliver a complete view of the simulation game, the bucket algorithm requires at least one ADU per participant to be available in each bucket. However an ADU can be missing for various reasons. It may have been lost by the network or it may be late. Dead reckoning is used to replace missing ADUs. For each missing ADU, the state computation algorithm goes back to the previous buckets, looking for the most recent ADU received for the missing slave. Once found, this ADU is dead reckoned to estimate the position where the slave should be at the current time. The accuracy of the evaluation depends on the dead reckoning algorithm parameters such as the age of the ADU used. We have implemented the simplest possible dead reckoning algorithm in COSIMAS. When a communication position is missing at the time, we compute a bucket, we simply replay the last known position of this slave. We decided to have a simple dead reckoning algorithm in order to analyze online synchronization.

3.1 Dead Reckoning

Dead Reckoning is an extrapolation technique used in the aviation systems to compute an estimate of the current position of a plane based on the knowledge of its position in the past and on its trajectory [8]. Dead reckoning is commonly employed for consistency control in continuous applications such as distributed virtual environments and simulation games. It uses a combination of state prediction and state transmission. Each object of the shared state has a single controlling application instance. State prediction means that all sites are able to calculate state changes caused by the passage of time locally, e.g., the path of a bouncing ball. Only the controlling instance is allowed to issue operations, e.g., when the user changes the direction of his/her ball. These operations modify the state of the affected object such that it differs significantly from

the predicted state. Thus, the controlling instance has to notify all participants by propagating the updated state. Following the soft state approach, operations are transmitted unreliably as states, and communication is repaired by periodic ping/alive protocols. Each slave is only responsible for the shared ADUs. Instead, the controlling slave transmits an update when it detects the collision.

The main advantage of the dead reckoning approach is its low complexity. Thus, dead reckoning is employed for large simulations and distributed virtual environments. However, dead reckoning cannot guarantee correctness because it transmits states instead of events. Thus, it is not possible for a slave that receives a state update to determine how that state came to be. This may lead to an incorrect state. For example, consider a situation where two balls A and B approach each other. At some point in time, the instance controlling ball A receives a state update for B that puts B past A. If some preceding updates were lost, there is no way for the controlling slave of A to determine how B got to this position and whether the two balls collided or not.

3.2 Collision Problem

Dead-reckoning method suffers from error during the period of an occurrence of an event and the communication of this event to the relevant slaves. It also handles the false positive and false negative collision detection problems. The new method is required for handling state changes due to unpredictable user interventions or object collisions.

We suggest a family of consistency protocols for multiplayer games that explicitly addresses the two issues of concern when enforcing consistency: (a) when an event occurs, how do we figure out the set of clients that need to be informed of this event, and (b) how do we schedule the dissemination of updates to meet both update receipt deadlines, as well as receive parameters.

4 Case Study

4.1 Multi-method Multi-body Coupling

With the component model synthesis technique, the FlexibleBody model was constructed on the standard Modelica MultiBody library [9]. A coupled rigid-flexible mechanism and a control subsystem were modeled and simulated.

The variable macro step size are investigated within the framework for the automotive drive cycle simulation [10].

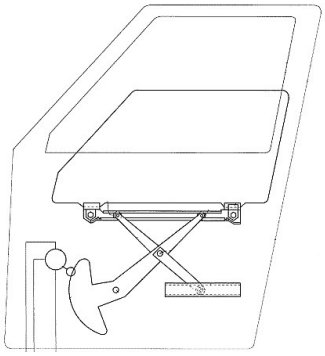

Figure 5: A power window simulation model

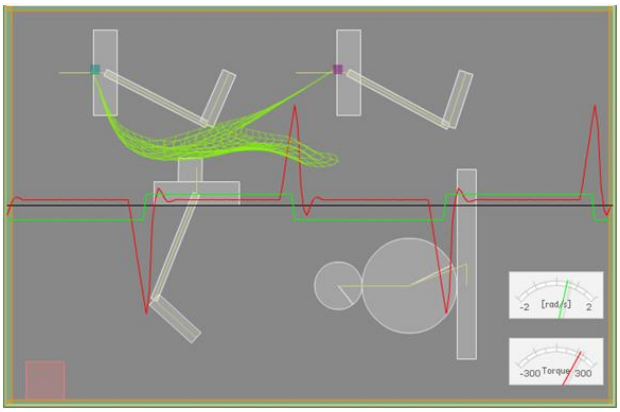

Figure 6: Multi-method co-simulation for rigid-flexible coupling

For the analysis of the power window as in Figure 5, we implement the online co-simulation workbench as shown in Figure 6. This is a co-simulation stage where rigid body slaves and elastic body saves are playing.

In order to advance time integration in a stable manner, the 1st order symplectic Euler algorithm is applied for rigid body slaves, while the 4th order Runge-Kutta scheme is necessary for elastic body slaves. The elastic body is modeled with spring-and-mass components. Those two slaves exchange the position and velocity information of the two pinned points at every time step. This is a kind of multi-method co-simulation with common constant time steps.

4.2 Collaborative Control

As a development methodology, “Multi-X-In-the-Loop” is proposed. The X stands for model, software, hardware, and players. While, specialized consistency control and synchronization methods for multiplayer games are needed.

The implementation is through a minimal set of C-functions. As an example, an engineer may have control software written in C, control system elements modeled in Simulink and/or MapleSim, and power electronics, mechanical, magnetic and other system plant models developed in ANSYS. The user would like to combine the simulator control, communication and synchronization of signals, and retain the integrity of the respective models and code, and simulate the entire system.

The Functional Mockup Interface (FMI) has become one kind of unified model exchange and co-simulation standard for multi-domain physical systems. The verification of system-level modeling of multi-domain systems using Functional Mockup Unit(FMU) is difficult to be accomplished by an individual or individual enterprise because of its complexity and multidiscipline. The COSIMAS aims the testbed to form the unified online platform for collaborative co-simulation. Through transplanting the standalone tools of modeling and simulation for FMU into the Web, the usage scenarios are widely expanded.

5 Conclusions and Future Work

In this paper, we have proposed an online functional mock-up platform to support collaborative co-simulation. It considers how interaction with simulation messages is perceived by co-simulation models and users under the existence of network latency. It also handles the collision detection problems and the unpredictable user interaction events. Our current study is regarded as in a Proof-of-Concept phase, we would like to proceed the detailed performance evaluation applying COSIMAS platform to be compliant to FMI 2.0RC1 as our future work. In addition, some legacy batch models written in Fortran code should participate in this online paradigm. It may widen the application of FMI co-simulation to more global human and software network cloud, such as massively parallel supercomputers concerted in harmony with Android and iOS smartphones.

References

[1] FMI 2.0 RC1: Functional Mock-up Interface for Model Exchange and Co-Simulation. https://www.fmi-standard.org/.

[2] DIS: IEEE Standard for Distributed Interactive Simulation – Application Protocols. (IEEE Std 1278.1-2012) IEEE Computer Society, 2012.

[3] HLA: Standard for Modeling and Simulation High Level Architecture. (IEEE Std 1516.1-2010) IEEE Standards Association, 2010.

[4] UPC: Union Procedure Call Protocol Specification. http://unionplatform.com/specs/upc/.

[5] Qi, L., Tifan, X., Qinghua, L., Liping, C.: WebMWorks: A General Web-Based Modeling and Simulation Environment for Modelica. In: Proc. 9th International Modelica Conference, Munich, Germany, pp.549-555, 2012.

[6] Schierz, T., Arnold, M., Clauß, C.: Co-simulation with communication step size control in an FMI compatible master algorithm. In: Proc. 9th International Modelica Conference, Munich, Germany, pp.205-214, 2012.

[7] Li, F.W.B., Li, L.W.F., Lau, R.W.H.: Supporting Continuous Consistency in Multiplayer Online Games, Proc. 12th ACM Multimedia, New York, pp.388-391, 2004.

[8] Lin, K. C., Schab, D. E.: The Performance Assessment of the Dead Reckoning Algorithms in DIS. In: Simulation, Vol.63, No.5, pp.318-325, 1994.

[9] Gang, X., Yan, Z., Fanli, Z., Liping, C.: Modelling and Simulation of the Coupled Rigid-flexible Multibody Systems in MWorks. In: Proc. 9th International Modelica Conference, Munich, Germany, pp.405-416, 2012.

[10] Günther, F., Mallebrein, G., Ulbrich, H.: AModular Technique for Automotive System Simulation. In: Proc. 9th International Modelica Conference, Munich, Germany, pp.589-598, 2012.