HLA Based Collaborative Simulation with MATLAB Seamlessly Embedded

Wei Xiong, Pengshan Fan and Hengyuan Zhang

***Abstract*—In order to build multi-domain collaborative simulation systems, the method of making MATLAB programs join into HLA/RTI based distributed interaction simulation system is need to be studied. Aiming at giving full play to the advantages of MATLAB in engineering computing and visualization, improving the efficiency of collaborative simulation systems, this paper proposed a method making MATLAB programs and SIMULINK modules seamlessly join into HLA/RTI based simulation system. The principles and designs of the main modules are elaborated, and the codes automatic generation orienting the method is also introduced. The demonstrations and testing results show that the method could fully exert the simulation functions of MATLAB, and has advantage in simulation efficiency over existing methods.**

***Index Terms*—Software integration, HLA, Collaborative simulation, MATLAB.**

I. Introduction

As one of the most popularized commercial simulation software in market, MATLAB has been widely used both abroad and at home. Besides its integrated feature of engineering computing, numerical analysis, modeling and visualization, MATLAB has a number of toolboxes serving for different fields as well as SIMULINK system simulation tools, which have all contributed to its significance in conducting simulation. However, in most cases, MATLAB is applied in single host model environment instead of distributed environment. As criterion of distributed interactive simulation system, HLA (High Level Architecture [1]) aims to solve the interoperation and reusing among different simulation applications. By integrating the functions of different simulation systems based on RIT (Run-Time Infrastructure), HLA can develop larger simulation systems with more sophisticated structure and broader functions.

Therefore, it would be of great significance for MATLAB joining into HLA/RTI based distributed interactive simulation system, taking its advantage in modeling and simulation field and making full use of its existing simulation codes [2-3].

A. Related Works

In view of the absence of a plug on MATLAB for HLA/RTI, many approaches have been looked into to make MATLAB join into HLA based simulation, forming the multi-domain collaborative simulation system, which are cited as below.

According to [4], [5], MATLAB is linked to RTI by a middleware (see Fig.1), which joins HLA as a federation member, calling MATLAB program, reading and writing MATLAB workspace, supported by MATLAB engine running on the backstage. Besides, the middleware is in charge of the data exchange between MATLAB and HLA federation, and time synchronization.

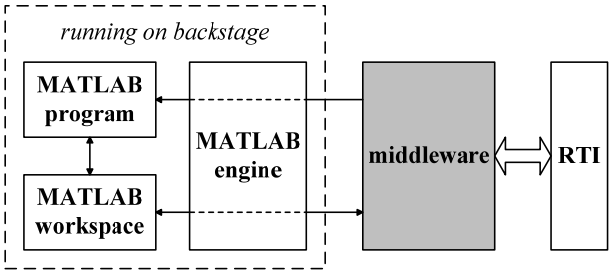


Fig. 1. MATLAB-RTI integration based on MATLAB engine

As illustrated in [6], by employing the TCP/IP plug on MATLAB instrument control toolbox, a SOCKET is built between HLA and MATLAB modules, which realizes the communication between MATLAB program and RTI, and hence makes the MATLAB program join into HLA simulation (see Fig.2).

[7], [8] propose an idea to link MATLAB with RTI through MEX, but no detailed approaches are presented. In addition, MAK has introduced HLA/DIS toolbox and provided API in M language to carry out HLA functions (hereafter cited as HLA-MAPI). With these functions, MATLAB program itself can develop into a full HLA federate without any C++ code.

The existing approaches for connecting MATLAB program and RTI, and relevant commercial software may have some deficiencies and restrictions as following:

1) The approach cited by [4], [5] needs to read and write MATLAB working space, while in paper [6], extra SOCKET communication is required, which will deduce the efficiency of simulation.

2) In [4]–[6], the approach is supported by data mapping; nevertheless, the user would have difficulty to figure out the direct relationship between SOM (Simulation Object Model) and the input/output of MATLAB simulation model.

3) [4], [5] present a method calling MATLAB program on the backstage, which has no obvious linkage to the MATLAB environment, thus fails to make full use of the visualization function of MATLAB, and causes

Manuscript received March 12, 2012; revised March 28, 2012.

Wei Xiong and Pengshan Fanare with the Academy of Equipment, Huairou, Beijing, China (e-mail: 13331094335@189.com; fpspengshan@163.com).

Hengyuan Zhang is with School of Computer Science and Engineering, Beihang University, Haidian, Beijing, China (e-mail: zhanghy@vrlab.buaa.edu.cn).

inconvenience in the debugging of M program as well.

4) The HLA/DIS toolbox introduced by MAK only supports its built-in FOM (Federation Object Model), and no interfaces for further development according to the specific needs are presented.

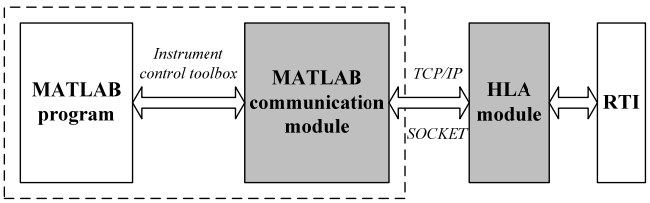


Fig. 2. MATLAB-RTI integration based on SOCKET

B. The Summary of Our Method

Oriented at enhancing the operating efficiency of collaborative simulation, building direct linkage between MATLAB module input/output and SOM data, and giving full play to advantages of MATLAB, the paper proposes an approach for MATLAB program to join into HLA/RTI simulation system without any middleware or communication module. As illustrated in Fig.3, just as the other federates, MATLAB federate communicates with RTI through LRC (Local RTI Component). Its structure from downside to upside are presented as following: 1) HLA foundation codes packs numerous complicated HLA services and callbacks into C++ classes; 2) According to the interface specification of MEX, create MEX functions that calls HLA foundation codes and performs MATLAB/C++ data translation, and compile it into MEX-DLL (hereafter cited as HLA-MEX) that can be invoked directly by MATLAB; 3) HLA-MAPI wraps up the whole process of HLA-MEX invocation by M language; 4) As a simulation-flow framework program written in M language, M federate framework calls HLA-MAPI and MATLAB simulation model. Based on the above structure, a MATLAB federate with M programs as its top layer, seamlessly linked to RTI through access layer, is created in its real sense.

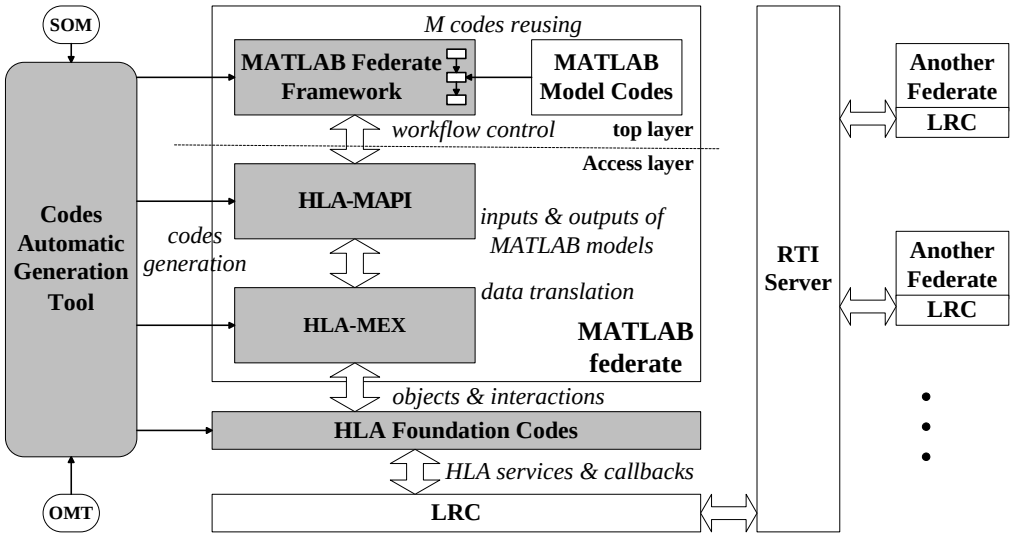


Fig. 3. System design of our method

The approach is highlighted by its top efficiency for the data exchange is conducted between MATLAB federate and its counterparts without any middleware, the operating efficiency increases greatly. Besides, with M program located as the top layer of MATLAB federate the inputs and outputs of MATLAB model associate with SOM directly. In addition, when MATLAB federate operates in MATLAB environment, it will make full use of MATLAB functions, and serve for the debug of M codes.

To facilitate FOM extension, the author regulates the structure and procedure of MATLAB federate framework, and reaches the automatic generation of codes. HLA foundation codes, HLA-MEX interface codes, HLA-MAPI codes and MATLAB federate framework codes are generated based on OMT (Object Model Template) and SOM. Work out the matchup between the inputs/outputs of MATLAB models and SOM, wrap up MATLAB models into M functions or scripts and call them by MATLAB federate framework, so that the FOM extension for MATLAB federate development is substantially simplified. As cited in [9], considerable amendment needs to be made on the original program when wrapping up RTI services into MEX function for MATLAB invocation, which makes RTI service opaque to the developers. While when using the approach stated above, only a small amount of work is required for code amendments.

II. Fundamental and Design

In this section, the fundamental and design of HLA-MEX, HLA-MAPI and federate M-framework in Fig.3 will be elaborated. But HLA foundation codes generation will be omit, because it is the basic work in the development of most HLA federations.

A. HLA-MEX

MEX is the interface by which MATLAB can invoke programs written in other languages. MEX-DLL is the C/C++ based DLL (Dynamic Link Library) which can be loaded and executed by the MATLAB interpreter in the same way of calling its built-in functions [10], [11]. HLA-MEX is

the MEX-DLL that performs HLA simulation related tasks, including: 1) establishing the direct relationship between M-program and RTI in the layer of HLA services; 2) translating the data from MATLAB program to C/C++ program, and vice versa.

When invoked by MATLAB, the input and output of MEX-DLL must be the MATLAB data type, and the C/C++ variables and objects created in stack by MEX-DLL procedure will be destroyed after the invocation. Therefore, in the process of multi-times MEX-DLL invocation, the HLA federate needed *RTIambassador* object and *FederateAmbassador* object cannot consistently exist in the MATLAB process memory. MATLAB program cannot call HLA services through the same *RTIambassador* object and obtain HLA callbacks through the same *FederateAmbassador* object, result in the discontinuous life period of MATLAB federate.

Take the process illustrated in Fig.4 as an example. First, the MATLAB program invokes MEX-DLL *A*, whose interface function creates *RTIambassador* object *rtiAmb* and *FederateAmbassador* object *fedAmb*, and makes the MATLAB program join the federation through calling HLA service *joinFederationExecution*; then the MATLAB program invokes MEX-DLL *B*, trying to turn on the asynchronous delivery mode of the MATLAB federate through calling HLA service *enableAsynchronousDelivery*. Obviously, *B* cannot fulfill its task, because *rtiAmb* has been destroyed after the invocation of *A*.

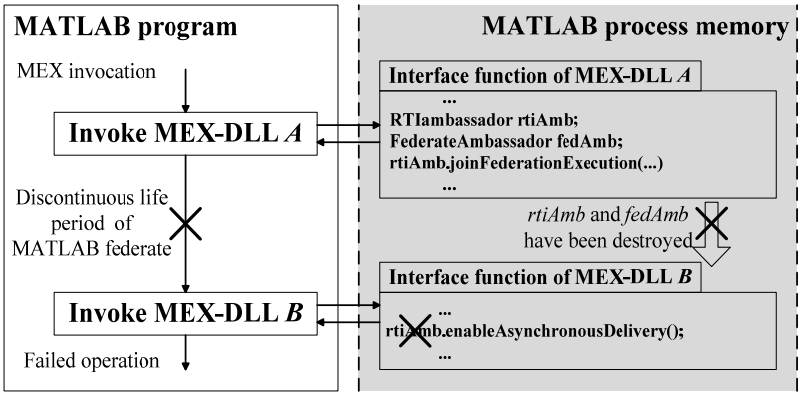


Fig. 4. The case of discontinuous life period of MATLAB federate

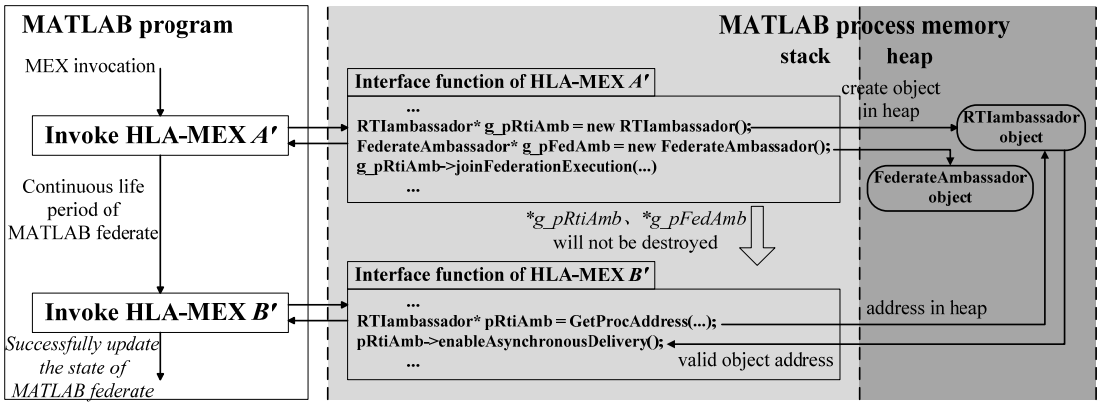


Fig. 5. The method of the life period of MATLAB federate can last

According to the memory management mechanism of Windows operating system, objects created in stack within a function procedure will be destroyed after the function returns, but objects created in heap can continuously exist in memory until the objects be deleted explicitly. We observed through experiments that, MATLAB will not release the memory image of MEX-DLL after invocation, unless removing the MEX-DLL module using 'clear' command. Therefore, by creating *RTIambassador* object and *FederateAmbassador* object in the heap of MATLAB process through MEX-DLL, combined with exporting and addressing of DLL variables, the life period of MATLAB federate can last for multi-times MEX-DLL invocation. HLA-MEXs possess these features.

Take the process illustrated in Fig.5 as an example. First, the MATLAB program invokes HLA-MEX *A'*, whose interface function creates *RTIambassador* object and *FederateAmbassador* object in the heap of MATLAB process using 'new' operator, and exports their addresses as DLL variables. Since the *RTIambassador* object and *FederateAmbassador* object are created in heap, they will not be destroyed after the invocation of *A'*. Then when the MATLAB program invokes HLA-MEX *B'*, because the memory image of *A'* remains in MATLAB process, the handle of *A'* can be obtained in the interface function of *B'*, and the *RTIambassador* object and *FederateAmbassador* object can be addressed using the handle. Finally, through the valid pointer of the *RTIambassador* object, the asynchronous delivery mode of the MATLAB federate can be turned on through calling HLA service *enableAsynchronousDelivery*.

In this way, the life period of MATLAB federate can last.

Besides making the MATLAB program possess the ability of directly calling HLA services, HLA-MEXs are also in charge of the data mapping and translating between MATLAB program and C/C++ program, aiming to associate the inputs and outputs of the MATLAB simulation model with the SOM. Data mapping and translating mainly deal with class attributes and interaction parameters in FOM, whose data are carried by the C++ class members in HLA foundation codes, and the structure fields in MATLAB codes using our method. Therefore, the essential of data mapping and translating is using HLA-MEX to assign the C++ class members the value of MATLAB structure fields, and vice versa, which is shown in Fig.6.

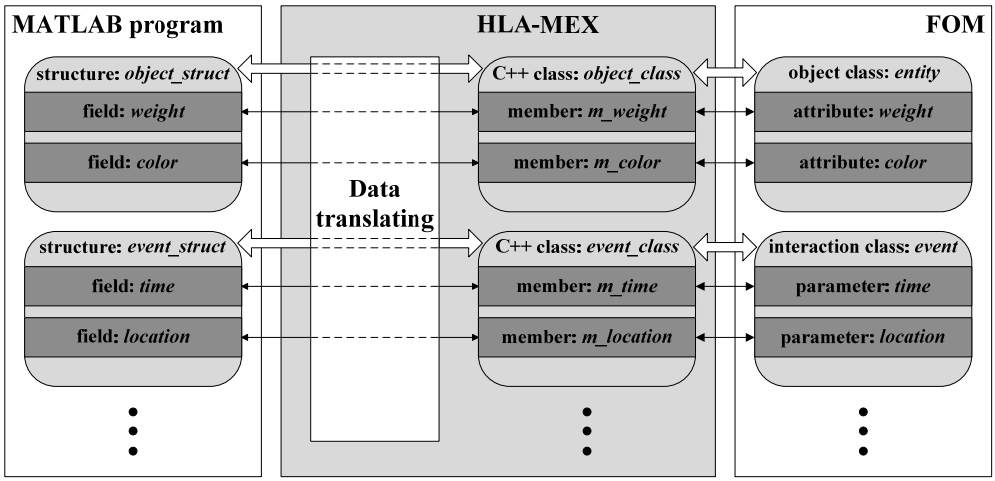


Fig. 6. Data mapping and translating using HLA-MEX

B. HLA-MAPI and MATLAB Federate Framework

Because of the great number and complex procedure of HLA services, in order to facilitate the development of MATLAB federate using M language, the processes of HLA-MEX invocations by M language is wrapped to canonical HLA-MAPI functions with clear interfaces. The effects of HLA-MAPI are in four aspects: 1) inputs/outputs conversion according to the design of simulation system; 2) parameter checking, exception handling and fault-tolerant processing of HLA-MEX invocations; 3) concealing the tedious underlying processes and details to provide transparent interfaces to MATLAB federate framework; 4) releasing the HLA-MEX modules no longer needed. Fig.7 illuminates the functional configuration and logical flow of HLA-MAPI.

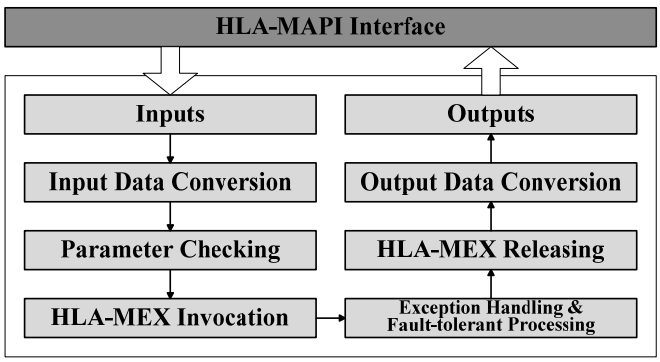


Fig. 7 The functional configuration and logical flow of HLA-MAPI

MATLAB federate framework is the M program in charge of simulation process control, federate life period management, maintenance and update of simulation model states, and interaction with other federates, by calling HLA-MAPI and encapsulated model functions or scripts. For HLA based distributed interaction simulation, the general behaviors and basic processes of federates in their life periods are roughly the same, no matter how simulation models, objects and interactions varies.

Fig.8 illustrates the general behaviors of MATLAB federate in its life period, involving four of six categories of HLA services, including federation management, declaration management, object management and time management. Our method formulates and designs MATLAB federate framework based on Fig.8, in which the development interfaces denoted by the darkest block are reserved for users to add or edit corresponding codes for federate initiation, simulation model utilization, interaction procession and etc., to meet the demand of diversified simulation applications. The workload of MATLAB federate development is minimized by the changelessness of the architecture and most codes of MATLAB federate framework. It is necessary to explain that due to the incapability to carry out multithreaded callback of MATLAB, MATLAB federate cannot respond interactions immediately, even though its asynchronous delivery mode is turned on by calling HLA service enableAsynchronousDelivery. For this reason, it is appropriate to deal with all the interactions after timeAdvanceGrant when designing MATLAB federate framework.

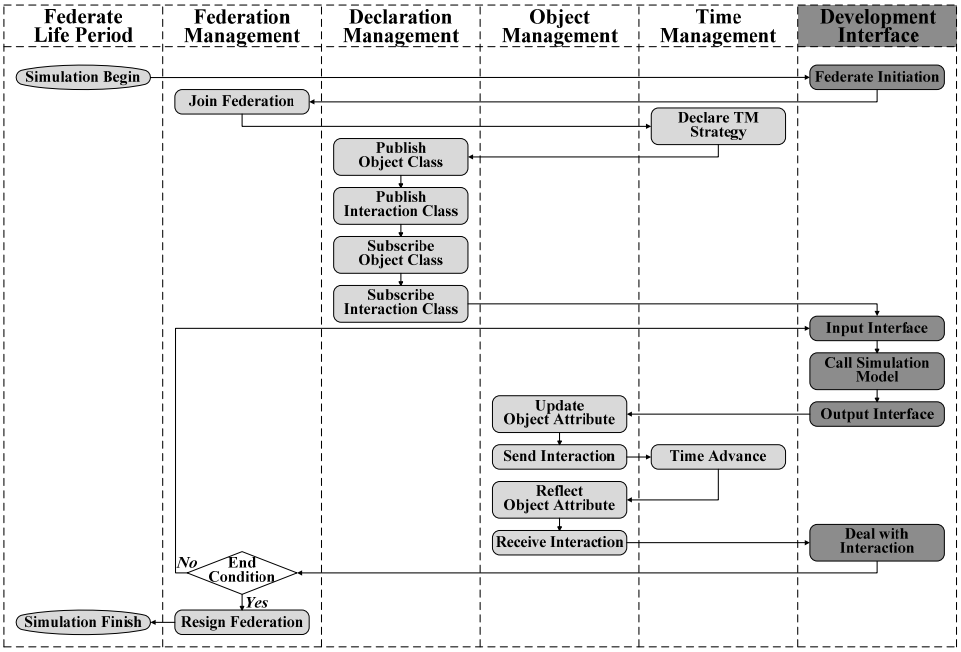


Fig. 8. The general behaviors of MATLAB federate

III. Codes Automatic Generation

In order to reduce the workload of FOM extension and conquer the limitation pointed out by [9], the technique of codes automatic generation catered to our MATLAB-RTI integration method is necessary. With the automatic generation of most codes our method needed, the federate developer could focus on the simulation content. Consulting the hierarchy in Fig.3, the technique of codes automatic generation is designed as follow.

1) Take the federation OMT as input, read in the names of object classes and interaction classes, the names and data types of object class attributes and interaction class parameters, output the files of HLA foundation codes, containing user extended classes, whose functions are illustrated in Table I. These user extended classes is the reification of FOM and OMT, according to the program language supported by specific HLA software.

TABLE I: User Extended Classes (HLA Foundation Codes)

Class Name	Function	Class Name	Function
XXStateRepository	Store and manage object attributes	XXObjectDecoder	Decode object attributes from network transmission
XXPublisher	Publish object class, Register and delete object instance, update object attributes	XXInteraction	Store and manage interaction parameters
XXReflected	Reflect object attributes	XXInteractionEncoder	Encode interaction parameters for network transmission
XXReflectedList	Discover and remove object instances	XXInteractionDecoder	Decode interaction parameters from network transmission
XXObjectEncoder	Encode object attributes for network transmission		

TABLE II: HLA-MAPIs

HLA-MAPI name *(HLA-MEX name)*	HLA-MAPI function *(HLA-MEX function)*	HLA-MAPI name *(HLA-MEX name)*	HLA-MAPI function *(HLA-MEX function)*
JoinFederation *(mtCreateExConn.dll)*	Join federation *(Create and export global variants of exercise connection manager, etc. and relevant functions)*	PublishObject *(mtPublishObject.dll)*	Publish object class, and register object instance *(Create XXPublisher instance of the object class)*
ResignFederation *(mtDeleteExConn.dll)*	Resign federation (Destroy global variants, remove or delete object instances)	SubscribeObject *(mtSubscribeObject.dll)*	Subscribe object class *(Create XXReflectedList instance of the object class)*
SetSendFedTime *(mtSetSendFedTime.dll)*	Set the transfer order of messages to be RO/TSO	UpdateObjectAttributes *(mtUpdateObjectAttributes.dll)*	Update object attributes *(Translate MATLAB data to C++ data, then update object attributes)*

EnableAsynchronousDelivery *(mtEnableAsynchronousDelivery.dll)*	Turn on the asynchronous delivery mode of federate	SendInteraction *(mtSendInteraction.dll)*	Publish interaction class, then send interaction instance *(Publish interaction class, and Translate MATLAB data to C++ data, then send interaction instance)*
EnableTimeConstrained *(mtEnableTimeConstrained.dll)*	Set federate to be time constrained, and wait for granted	SubscribeInteraction *(mtSubscribeInteraction.dll)*	Subscribe interaction class *(Subscribe interaction class, and register callback function)*
EnableTimeRegulation *(mtEnableTimeRegulation.dll)*	Set federate to be time regulation, and wait for granted	DrainInput *(mtDrainInput.dll)*	Discover object instances, reflect object attributes and receive interactions *(Discover object instances, reflect object attributes, update the queue of received interactions, and translate C++ data to MATLAB data)*
TimeAdvanceRequest *(mtTimeAdvanceRequest.dll)*	Request time advance, and wait for granted		

2) Take the federation OMT and SOM of MATLAB federate as inputs, output the HLA foundation codes based source code files of MEX interface functions, which can be compiled to HLA-MEXs. Some of these interface functions calling necessary HLA services, others translate the data of object attributes and interaction parameters between MATLAB program and C++ program.

3) Take the federation OMT and SOM of MATLAB federate as inputs, output the HLA-MEX based HLA-MAPI code files. HLA-MAPI invokes HLA-MEX, provide MATLAB federate framework with transparent interface by concealing the tedious underlying processes and details. Table II is the list of HLA-MAPIs and the description of the invoked HLA-MEXs.

4) Take the federation OMT and SOM of MATLAB federate as inputs, output HLA-MAPI based MATLAB federate framework, which is the MATLAB program conducting the processes in Fig.8, assisting and guiding the development of MATLAB federate. Due to space limitation, details of MATLAB federate framework are omitted in this paper.

IV. Test Results and Analysis

In order to demonstrate the advantages of our method in visualized simulation, we developed demonstration federation I consisting of MATLAB federate and MFC federate, whose running condition is shown in Fig.9. It is thus evident that the MATLAB federate based on GUI can be developed using our method, by which the visualization function of MATLAB can be fully utilized.

In order to demonstrate the ability of our method to make SIMULINK models join into HLA/RTI simulation system, we developed demonstration federation II consisting of SIMULINK federate and MFC federate, whose running condition is shown in Fig.10. It is thus evident that, by using our method, the collaborative simulation based on HLA can fully utilize the functions of SIMULINK.

The average actual time consuming of simulation time advance is the uppermost indicator to evaluate the running efficiency of HLA federations. In order to compare the performance of existing methods and ours, we developed three testing federations A, B and C, based on MATLAB engine, SOCKET and our method respectively. We made A, B and C conducts the same simulation calculation under the same software and hardware conditions, and compared their average actual time consuming of simulation time advance.

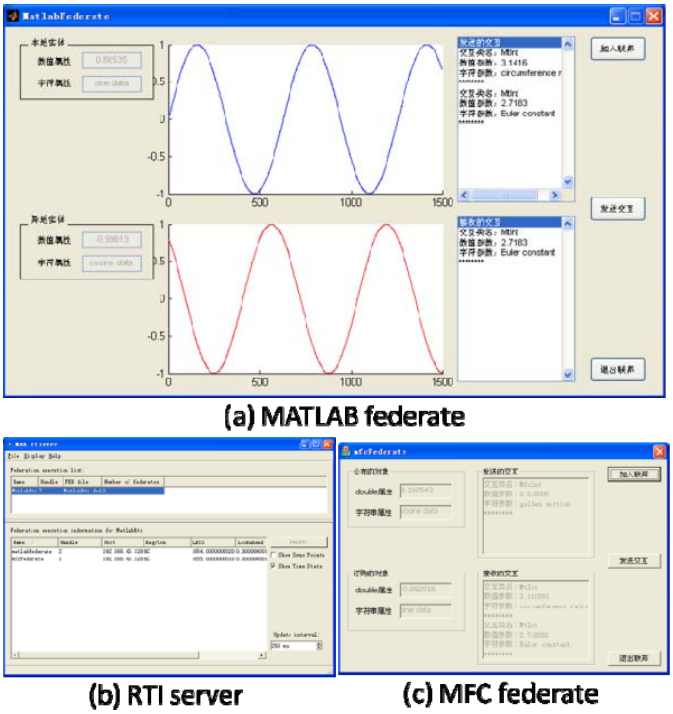


Fig. 9. Running condition of demonstration federation I

Federation A, B and C consist of federate A1 and A2, B1 and B2, C1 and C2 respectively. A1 is a C++ program which calls the MATLAB model and reads/writes MATLAB workspace through MATLAB engine; B1 is composed of C++ based HLA module and M based MATLAB module in charge of calling the MATLAB model, and the modules communicate through SOCKET; C1 is a MATLAB program directly calls the MATLAB model and carries out HLA operations through HLA-MAPI. A2, B2 and C2 are C++ programs only record the time of receiving *timeAdvanceGrant* callback, and the recorded data can be processed to the actual time consuming of every simulation time advance. We set Ai, Bi and Ci to be time regulation and constrained (RC), and their lookahead to be zero, to make Ai, Bi and Ci strictly synchronized respectively, so that the actual time consuming of simulation time advance of A2, B2 and C2 are exactly that of the federations they belong to.

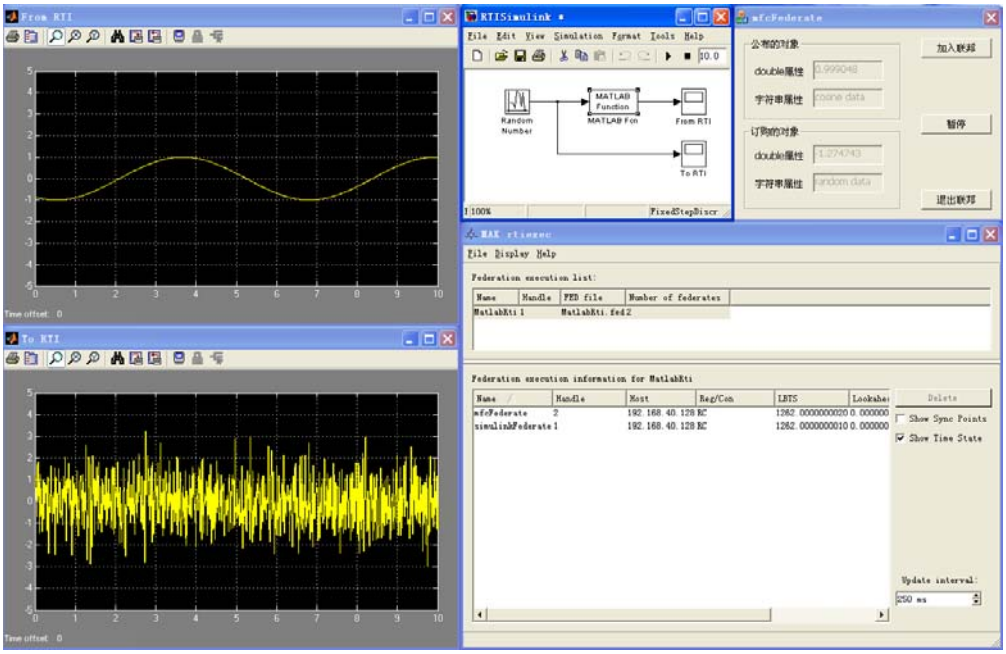


Fig. 10. Running condition of demonstration federation II

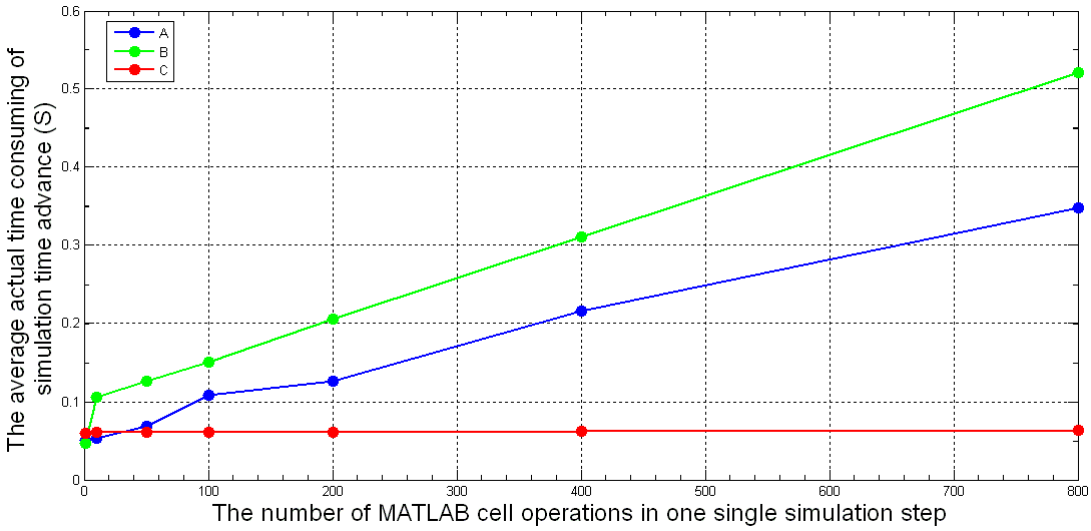


Fig. 11. Running condition of demonstration federation II

In the test, the simulation model called by A1, B1 and C1 is the same simple function in MATLAB. According to the technical characteristics of the methods, the authors defined MATLAB cell operation for each type of MATLAB federates: 1) for A1, calling the simulation model once, reading and writing (4 bytes) MATLAB workspace once; 2) for B1, calling the simulation model once, MATLAB module and HLA module communicating (4 bytes) once; 3) for C1, calling the simulation model once. The number of MATLAB cell operations conducted in one single simulation step reflects the simulation content complexity of the federate. In order to obtain the average actual time consuming of simulation time advance of the federations under different simulation content complexities, the authors made tests in the case of conducting 1, 10, 50, 100, 200, 400, 800 MATLAB cell operations in one single simulation step respectively.

The testing results are shown in Fig.11. It is evident that when the complexity of simulation content is low, the average actual time consuming of simulation time advance of the testing federations are similar; when the complexity is high, the average actual time consuming of simulation time advance of federation A and B are far greater than that of C. Furthermore, along with the increasing of the number of MATLAB cell operations, the average actual time consuming of simulation time advance of federation A and B rise markedly and linearly, but that of C rises slightly. Hence we can see that the costs of the additional operations, including calling MATLAB engine, reading and writing MATLAB workspace, communicating through SOCKET, which are necessary to the MATLAB-RTI integration methods based on MATLLAB engine and SOCKET, usually exceed the cost of MATLAB model calculation. Whereas, the additional cost of the MATLAB-RTI integration method of this paper is far less than that of MATLAB model calculation.

So it is clear that the method of MATLAB-RTI integration proposed in this paper is markedly prior to existing ones in simulation efficiency. Furthermore, our method is helpful to

real-time simulation, for the little additional cost making the time consuming in simulation steps easy to forecast [12].

V. Conclusion

In order to build multi-domain collaborative simulation systems, the method of making MATLAB programs join into HLA/RTI based distributed interaction simulation system is need to be studied. Based on the analysis of the deficiencies and limitations of existing MATLAB-RTI integration methods, this paper puts forward a method making MATLAB programs seamlessly join into HLA/RTI simulation system, elaborates the principles and designs of HLA-MEX, HLA-MAPI and MATLAB federate framework, and introduces the codes automatic generation orienting the method. Finally, the demonstrations and testing results show that our method could fully exert the visualized simulation functions of MATLAB, and has advantage in simulation efficiency over existing methods. In addition, our method not only applies to embedding MATLAB programs into simulation systems based on various HLA platforms, but provides important suggestions on embedding other simulation software (e.g. LabView, Scilab) into HLA/RTI simulation system.

References

[1] *Modeling and Simulation (M&S) High Level Architecture (HLA) - Framework and Rules*, IEEE Standard 1516-2000.

[2] Straßburger, S., "Distributed Simulation Based on the High Level Architecture in Civilian Application Domains," PhD Dissertation, University of Magdeburg, Germany, April, 2001..

[3] X. Tong, J. Huang, D. Zhang, "HLA Based Multidisciplinary Joint Simulation Technology for Servo Mechanism Analysis," *Proceedings of the 2009 IEEE International Conference on Mechatronics and Automation*, 2009, pp. 4517-4522.

[4] B. Guo, G. Xiong, X. Chen, "Researching and Development of the General Adaptor of MATLAB and HLA/RTI," *Journal of System Simulation*, vol. 16(6), pp. 1275-1279, 2004.

[5] H. Wang, J. Tang, W. Mao, "Research on Implementation and Application of MATLAB_RTI Middleware," *Computer Engineering and Design*, vol. 27(20), pp. 3827-3830, 2006.

[6] C. Sung, J. Hong, T. Kim, "Interoperation of DEVS Models and Differential Equation Models using HLA/RTI: Hybrid Simulation of Engineering and Engagement Level Models," *Proceedings of the 2009 Spring Simulation Multiconference*, 2009, pp. 387-392.

[7] S. Pawletta, W. Drewelow, T. Pawletta. "HLA-based Simulation within an Interactive Engineering Environment," *Fourth IEEE International Workshop on Distributed Simulation and Real-Time Applications*, 2000, pp. 97-102.

[8] C. Stenzel, S. Pawletta, R. Ems, P. Bünning, "HLA Applied to Military Ship Design Process," *Simulation News Europe*, vol. 16(2), pp. 51-56, 2006.

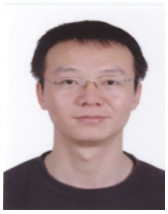
[9] H. Qiao, X. Tian, K. Huang, "Using Simulink Model in HLA Simulation," *Journal of System Simulation*, vol. 18(2), pp. 335-340, 2006.

[10] D. Pan, B. Wang, Z. Zhou, "Research on Mixed Programming Technology of MATLAB and C/C++," *Computer Engineering and Design*, vol. 30(2), pp. 465-468, 2009.

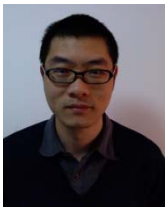
[11] L. Hong, J. Cai, "The Application guide of mixed programming between MATLAB and other programming languages", *Computer and Automation Engineering*, 2010, pp. 185-189.

[12] Z. Li, H. Zhang, "Research of HLA Real-time Simulation Method Based on Alterable Time Advance Step," *Journal of the Academy of Equipment Command & Technology*, vol. 20(2), pp. 106-110, 2009.

Wei Xiong received the B.S. degrees in Electronic Engineering from the National University of Defense Technology, Changsha, China, and the M.S. degrees and the Ph.D. in Military Communication and Information System from the Academy of Equipment Command & Technology, Beijing, China, in 1992, 1998 and 2005 respectively. He is presently the Research Fellow in the Academy of Equipment, Beijing, China. His current research interests are system analysis and integration.



Pengshan Fan received the B.S. and M.S. degrees in Flight Vehicle Design from the Academy of Equipment Command & Technology, Beijing, China, in 2005 and 2008 respectively. His current research interests are System analysis and integration.



Hengyuan Zhang has received both B.Eng and M.Eng degrees from The Academy of Equipment. He is presently a Ph.D candidate in the School of Computer Science and Engineering under Beihang University. His research areas cover information visualization, computer simulation and information security.