J. King Saud Univ., Vol. 13, *Comp. & Info. Sci.*, pp. 63-76 (A.H. 1421/2001)

Frameworks for Component-based Simulation

Hussam M. Soliman
Department of Information Systems, College of Computer & Information Sciences
King Saud University, P.O.Box 51178, Riyadh 11543, Saudi Arabia

(Received 09 February 2000; accepted for publication 05 September 2000)

Abstract. The need to reduce development costs of simulation models has led to recent efforts for setting simulation standards that foster model reuse and interoperability. Specifically, the High Level Architecture (HLA) is a new simulation standard supported by the US Defense Modeling and Simulation Office (DMSO). It has been adopted as the standard technical architecture for all US Department of Defense simulations. In the meantime, the commercial sector has had successful efforts in developing enabling technologies for distributed computing; namely, the Common Object Request Broker Architecture (CORBA) by the Object Management Group (OMG). CORBA is a large and complex set of specifications and protocols that utilizes the object-oriented paradigm to achieve distributed object-oriented computing environments that allow object interoperability and reuse. When used as an infrastructure for simulation model reuse and interoperability, both HLA and CORBA exhibit merits and limitations. Since HLA and CORBA were developed independently, need exists for a comparative evaluation of the two architectures as a basis for component-based simulation. In this paper, both HLA and CORBA are presented in the context of component-based simulation model development and interoperability. The two architectures are compared against four comparison criteria that are related to their conceptual foundation and design.

1. Introduction

The considerable resources invested in developing large-scale simulation models and the increased budget pressures has led to a greater need than ever to reduce simulation cost by seeking a common framework that supports simulation model reuse and interoperability. Standardizing the simulation architecture is a key element for achieving such a framework. Two recent standard architectures will be compared in this paper: HLA and CORBA.

The High Level Architecture (HLA) is an architecture for the reuse and interoperability of simulations. CORBA, on the other hand, is a standard for interoperability in heterogeneous computing environments standardized by the Object

Management Group (OMG). It enables applications to cross the boundaries of different computing machines, operating systems, and programming languages. Unlike the HLA, however, it was not developed for simulation applications in particular. This implies major differences in the way HLA and CORBA can be used to realize component-based simulation.

This paper will attempt to shed light on the fundamental differences between HLA and CORBA in this context. In section 2, an overview of HLA is given. In section 3, CORBA fundamentals are presented with emphasis on how component-based simulation can be done using CORBA as an infrastructure. Section 4 compares the two architectures in the context of component-based simulation. Section 5 gives conclusions.

2. The High Level Architecture (HLA)

The High Level Architecture is a standard framework for synthesizing complex simulations from several constituent simulation components. Usually, complex simulations require the simulation of several different aspects and subsystems that make up the overall system of interest. In addition, simulations of some of these component systems may already exist but are incompatible with each other. This incompatibility would prevent the developer from reusing these component simulations to build the total simulation model, and would usually require him to write a new simulation model from scratch.

Such considerations have created the need for developing a standard framework, such as HLA, for the reuse and interoperability of simulation models. *Reusability*, here, means that component simulations can be reused in different simulation applications. *Interoperability*, on the other hand, means that the reusable simulation models can be combined to work with each other without change.

The HLA was originally developed by the Defense Modeling and Simulation Office (DMSO) to meet the needs of the United States Department of Defense's (DoD) military simulations [1]. However, it is now increasingly accepted in other application areas. DMSO policy is to disseminate information about the HLA as widely as possible worldwide, and to provide free supporting software to new users. With respect to DoD simulations, DMSO designated the end of fiscal year 1999 as the beginning of new era where the DoD will not pay for any non-HLA-compliant simulations. By the end of fiscal year 2001, the DoD will force all existing DoD non-HLA-compliant simulations to retirement.

The HLA design rationale is that no single monolithic simulation can satisfy the needs of all users. Moreover, there is no way to anticipate all uses of simulations and the

possible ways of combining them. Therefore, HLA designers based their design on the following principle: federations of simulations are to be constructed from modular components with well-defined functionality and interfaces. In addition, specific simulation functionality is separated from general-purpose, supporting runtime infrastructures.

HLA considers a complex simulation as a hierarchy of component simulations each called a *federate*. There can be multiple instances of a particular type of federate. Federates participate in a *federation* that represents the aggregate simulation. Federates interact and interoperate using the protocols specified by the HLA. They may join and resign from the federation as the simulation executes. In practice, a federate may also be an interface to human operators or an interface to general software performing functions such as data collection and display. Fig. 1 illustrates the general architecture for HLA-based simulations.

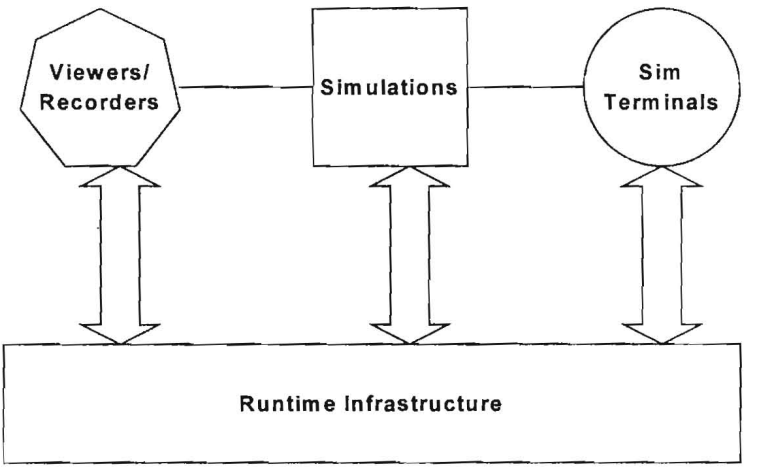

Fig. 1. The General HLA-based simulation architecture.

2.1 The HLA specifications

The HLA specification consists of three components: *HLA Rules*, *Interface Specification*, and *Object Model Template*. The HLA Rules are principles that govern the interaction of federates during a federation execution. There are ten rules which must be applied by federates and federations to achieve HLA compliance. The first five rules are the *Federation Rules*. These establish the basic rules for creating a federation. Rule1

covers documentation requirements stating that federations must have a Federation Object Model (FOM) documented according to the Object Model Template (OMT). Rule 2 states that object instance representations must be in federates, not in the Run-time Infrastructure (RTI) software. Rule 3 requires that all data exchanges of FOM data among federates must be done via the RTI. Rule 4 states that all interactions between federates and the RTI must observe the HLA Interface Specification. Rule 5 requires that an instance attribute must not be owned by more than one federate at a time.

The other five rules are the *Federate Rules* which deal with individual federates. Rule 6 requires that federates must have a Simulation Object Model (SOM) documented according to the OMT. Rule 7 defines HLA compliance with respect to attributes and interactions (simulated occurrences). It states that federates will initiate the appropriate behavior with the RTI and will respond to RTI-initiated services with respect to each attribute and interaction in its SOM. Rule 8 requires that federates must also implement their part of the ownership transfer protocols defined in the Interface Specification. Rule 9 allows federates to vary the conditions under which they provide updates of attributes, as specified in their SOMs. Finally, Rule 10 requires a federate to use some set of the time management functions of the RTI to manage its logical time and to help others manage theirs.

The HLA *Interface Specification* defines a standard for the Run-Time Infrastructure (RTI). The RTI is the software that allows a federation to execute together. It is general-purpose distributed operating system software that provides the common interface services during the runtime of an HLA federation. It also implements generic functions to coordinate among simulations. It may be implemented as many processes or as one process, but it remains conceptually as one entity. It is the interface between federates and the RTI that is standardized by the Interface Specification.

The Interface Specification is divided into six management areas: Federation Management, Declaration Management, Object Management, Ownership Management, Data Distribution Management and Time Management. *Federation Management* services deal with the existence of a federation in terms of the definition of existence and membership to a federation execution. They also deal with federation-wide operations such as federation-wide synchronization and checkpointing. *Declaration Management* services allow federates to declare their intent to publish or subscribe to data. Subscriptions are used by the RTI to decide which federates should be informed of the creation or update of entities, and to prune inapplicable data. *Object Management* services are those that are used by federates to register new instances of an object class and to update their attributes, or to discover new instances registered by federates and receive updates of their attributes. They are also used to send and receive interactions. *Ownership Management* services help determine the federate(s) responsible for simulating an entity, and allow the sharing and transfer of this responsibility. *Data*

Distribution Management services form a general scheme for characterizing the production and consumption of data, using the notion of routing spaces, to allow the RTI to route data only to interested federates. *Time Management* services deal with the problem of properly ordering events between federates making up a federation. They allow federates to advance their logical simulation time. They also control the delivery of time-stamped events among federates in a way that preserves causality.

The Object Model Template (OMT) Specification is the standard format for documenting HLA Object Model information. HLA prescribes that OMT objects and interactions can be defined and exchanged with no modification to the HLA-compliant simulation. The OMT defines mainly the Federation Object Model (FOM) and the Simulation Object Model (SOM). OMT is the *meta-model* for FOMs and SOMs. Each federation has an FOM that defines what objects and interactions will be shared among federates participating in this federation. It represents the vocabulary of data exchanged through the RTI during the federation execution. It includes an enumeration of all object and interaction classes related to the federation, along with a list of their attributes or parameters. There is only one FOM per federation. On the other hand, each federate has a Simulation Object Model (SOM) that describes the data the federate produces or consumes. It describes objects and interactions that can be used externally to allow the federate to participate in a particular federation. The OMT Specification requires that both federations and federates use all of its seven component tables that specify information about classes of objects, their attributes and their interactions. The seven component tables of the OMT are briefly described below. A more complete description can be found in the DMSO web-site [2].

The OMT specifies the following seven tables: Object Model Identification Table, Object Class Structure Table, Interaction Class Structure Table, Attribute Table, Parameter Table and Routing Space Table. The *Object Model Identification Table* provides key identification information about the federation or federate. The *Object Class Structure Table* contains information about the class-subclass hierarchy of the object classes, including whether each class is *Publishable*, *Subscribable*, or *Neither*. The *Interaction Class Structure Table* defines interaction classes in a federate or federation, and classifies the federate/federation capabilities with respect to its interaction classes. The *Attribute Table* documents the object class attribute types that make up the object's state. The *Parameter Table* contains the full set of parameters associated with every interaction class identified in the interaction class structure table. Finally, the *Routing Space Table* specifies *routing spaces* which are the way data distribution management services control data distribution to limit the flow of object attributes and interaction data delivered to federates.

3. The Common Object Request Broker Architecture (CORBA)

CORBA is a standard platform-independent architecture for interoperable distributed software components. In CORBA's underlying Object Management Architecture (OMA), distributed application objects communicate over the Object Request Broker (ORB) software bus in a heterogeneous distributed computing environment. Both client and object implementation are isolated from the ORB by a standard interface language, the OMG IDL. Every object's interface must be defined in OMG IDL. Since clients see only the object's interface, not its implementation, software component can interoperate without regard to any component's implementation details such as platform, operating system, programming language, or network hardware and software.

A client request does not pass directly from the client to the object implementation. Rather, the ORB manages every invocation of a CORBA object. The invocation may pass from one ORB to another if the object implementation is remote. All distribution details are handled by the ORB not by the application objects. The ORB is usually implemented as a library of routines that are linked into an executable module along with clients and object implementation. ORB-to-ORB communication is supported by the OMG's standard General Inter-ORB Protocol (GIOP) which specifies all aspects of interoperability. The GIOP, layered over TCP/IP transport, forms IIOP (the Internet Inter-ORB Protocol) which is a mandatory standard for CORBA-compliant distribution.

The OMG IDL interface definition specifies operations the object can perform, the input and output parameters for each operation, and any exceptions that may be generated. This interface represents a promise that, for any proper invocation the client sends to an object through its interface, the expected response will come back. It also represents an obligation on the object developer to implement, in some programming language, all of the operations specified in the interface. Distributed objects can be implemented in any programming language that has an IDL mapping. CORBA currently has IDL mappings to C, C++, Smalltalk, Cobol, Ada, and Java.

CORBA supports both a static invocation interface, using client stubs and server skeletons, as well as dynamic invocation and dynamic skeleton interfaces. Static invocations are sent to the client's ORB through a stub that is compiled in the target programming language from the IDL interface definition. Dynamic invocations use an interface repository to allow information about distributed objects to be discovered at runtime. This information is used to build dynamic invocations.

The OMA defines an environment for component-based software development by requiring that applications provide their functionality only through a standard interface.

It builds on the CORBA architecture and OMG IDL to realize a plug-and-play component-based software environment. As shown in Fig. 2, OMA consists of CORBA services and CORBA facilities, application objects, and the ORB. OMG defines the specifications for CORBA services and CORBA facilities while the vendors provide the implementations for them. The CORBA services define a set of low-level services such as naming, events, trading and security. The CORBA facilities suite is a collection of services that many applications may share. They define a set of high-level services that applications frequently require when manipulating distributed objects. They are divided into two categories: the horizontal CORBA facilities and the vertical CORBA facilities. OMG's original plan for the horizontal CORBA facilities covered four major categories: User Interface, Information Management, Systems Management, and Task Management. Since the OMG's Common Facilities Task Force no longer exists, new facilities are being added slowly, usually in response to specific industry demands [3, 4]. The vertical/domain CORBA facilities are defined by OMG's eight Domain Task Forces: Business Objects, Finance/Insurance, Electronic Commerce, Manufacturing, Healthcare, Telecommunications, Transportation, and Life Science Research. These domain facilities specify frameworks for specialized but industry-standardized components in each of the above areas. Application objects are not standardized by OMG because this is where vendors will compete to provide the best features and products for their customers.

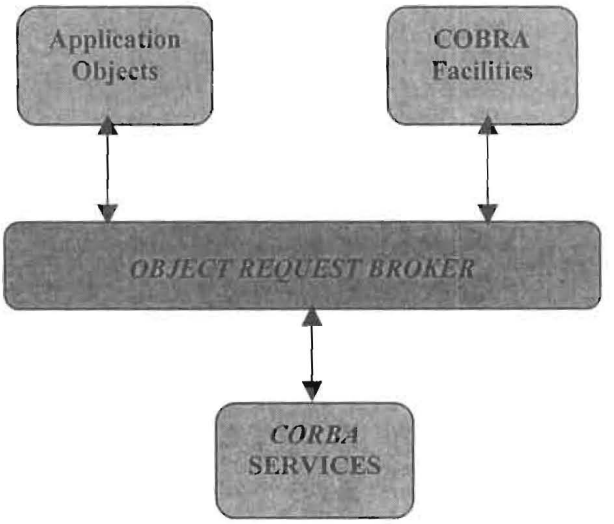

Fig. 2. The object management architecture.

3.1 CORBA and component-based simulation

Two approaches can be identified for constructing simulation software from components based on the CORBA infrastructure. The first approach is to partition the software into components on the basis of simulation tasks. Common tasks can be defined as services with published interfaces that can be implemented by different vendors using different languages and techniques. They can be acquired and integrated with other necessary components to construct the full simulation application. The second approach is to partition the simulation software on the basis of the simulation model structure. That is, the simulation model components represent submodels for the simulated system's component parts. These may be either complete working simulations that can be combined with other similar submodel simulations to build the complete simulation model of the system. According to this definition, a simulation component is analogous to the concept of a federate in HLA. They may also be individual simulation objects representing simulation model entities. This, however, requires that all simulation entities be developed under one common conceptual modeling framework [5].

Several attempts have been made to utilize CORBA in developing component-based simulation software. As an example of the first approach above, a CORBA-based discrete-event simulation facility is proposed in [6] to help develop portable and interoperable simulation models on the Internet by using CORBA and Java. The proposed facility is defined by a CORBA IDL interface which defines operations for object definition, inter-object communication and event scheduling. Based on the given IDL interface definition, different vendors could supply different products by using different simulation algorithms, different programming languages or different operating systems and hardware platforms. With respect to the simulation models, they see a consistent interface across all products. The simulation facility IDL interface is demonstrated with a prototype implementation in Java using a commercial ORB product.

An example of the second approach above is given in [7]. A web-based environment is constructed to allow the user to download interesting simulation models from the web, customize them and save them on his local file system. These retrieved simulation models can then be run by transparently accessing the appropriate simulation tool on the web. The environment architecture is based on the use of Java and CORBA. It consists of a web server from which a client Java applet is downloaded, a set of simulation tool servers, and a set of simulation model servers. Each simulation tool server is assumed to be running a specific simulation tool, and each simulation model server is assumed to be holding one or more simulation models that require a specific simulation tool to be run. Finally, a CORBA-based infrastructure is used to interface the client applet with the simulation tool and the simulation model over the Internet.

4. Comparison

In the following, four criteria will be used to compare HLA and CORBA as frameworks for component-based simulation. These are *component interface description*, *communication infrastructure*, *component directory service*, and *time management scheme*. To work together, distributed software components should have a *well-defined interface* that fully describes their behavior and indicates how to access their services. They must also have a common understanding of how to communicate with one another. That is, they must share some *communication infrastructure* that provides a unified way of communication. This infrastructure should allow for passing component references to requesting clients, instantiating component objects, and marshalling object requests between different locations. In addition, distributed components should also have some sort of *directory service* where clients can discover the objects available on a server for access, retrieve their references along with any necessary method signatures and arguments. This mechanism, if available, can help establish flexible distributed computing environments where inter-object communication can be delayed until runtime. The *time management scheme* must define a global measure of simulation time for the entire multi-component simulation as well as the method to compute it. To ensure simulation result correctness and to satisfy causality constraints, simulation model components need to follow one simulation time management scheme. This is the mechanism that controls the way each component advances its local simulation clock, in order to ensure that events being processed by each component are processed in a chronological order.

In the following subsections, HLA and CORBA's conceptual foundations will be compared in terms of the above four criteria of component-based simulation.

4.1 Component interface description

HLA uses the OMT as the component (federate) interface description language because it provides a common representational framework for documenting federates and federations object models. It does so through a set of seven mandatory tables that must be filled in by the federation/federate designer. On the other hand, CORBA uses the OMG IDL which defines object interfaces in a syntax that resembles C++. The IDL file is then complied into code in one of the supported languages. This code represents a starting point for implementing the interfaces defined in the IDL in their final form. These interface can be implemented as distributed objects in any of the supported languages and still be able to work together without even knowing each other's language.

4.2 Communication infrastructure

The HLA communication infrastructure is the RTI. The RTI may be viewed as a special purpose distributed operating system that provides services to interconnect

simulations. HLA Rules require that all data exchanges between federates and all federate interactions be made through the RTI. The Interface Specification sets the standard for federate interactions with the RTI. It defines how RTI services, which are divided into six service areas, can be accessed. The specification is provided as an application programming interface in several languages including OMG IDL. The communication infrastructure in CORBA is the ORB. It acts as a software bus that handles all distribution details, including address resolution and data marshalling, transparently. All client/server interactions must go through the ORB. Therefore, every object participating in a CORBA application must have some ORB software installed on its machine. ORBs can communicate with each other whether or not they are located on the same LAN or on different LANs.

4.3 Directory service

The HLA directory service is provided per each *federation execution* through the RTI process called RTIExec. This process manages the creation and destruction of multiple federation executions running on the same system (which must be independent of each other and may not exchange any information). Only one RTIExec process may exist at a time. Each federation contains its own FedExec process which manages the federation and allows federates to join and to resign from the federation. When a federate, acting as a manager, creates a federation execution by invoking the RTI method *createFederationExecution,* the RTI then reserves a name with RTIExec, and spawns a FedExec process. This FedExec process registers its communication addresses with RTIExec in preparation of the federation execution. Once a federation exists, other federates can join it by asking the RTI to consult RTIExec to get the address of FedExec, and invoking *joinFederationExecution* on FedExec.

HLA also supports a dynamic publish-and-subscribe mechanism by which each federate defines to the federation what data are to be published for each attribute update or event, and which updates and events it is interested in receiving. This mechanism depends on RTI service calls. *Publish* RTI calls are used to describe the data to be sent in messages to the RTI by a class of objects. New instances of these objects are registered with the system by the RTI service call *RegisterObjectInstance*. Federates that have subscribed to objects of this class will receive a *DiscoverObjectInstance* callback from the RTI to inform them of the existence of the new objects.

CORBA, on the other hand, provides the Dynamic Invocation Interface (DII) to avoid the need for precompiled client stubs. DII allows the user to discover the desired remote object, obtain its interface, obtain a specific interface's method details, and then invoke that method on the remote object. Interfaces that can be accessed dynamically are stored in an Interface Repository at the server.

4.4 Time management scheme

In HLA, federates use the Interface Specification's Time Management services to coordinate the advance of their local simulation time. The time management services include mechanisms to ensure time-stamp ordered delivery of messages, as well as mechanisms for federates to advance simulation time so that the federate does not receive messages with time stamps in their simulated past. The time management scheme is transparent in the sense that it allows federates to use different local time management mechanisms and still interoperate. Unlike HLA, which is oriented towards distributed simulation, CORBA is oriented towards general applications. Consequently, it lacks the specific simulation support of HLA, including time management services. CORBA does not even include a notion of simulation time. This implies that, to use CORBA as a framework for component-based simulation, it is necessary for the modeler to use some distributed simulation synchronization protocol between the component simulations to ensure temporal causality. These protocols may be conservative or optimistic [8]. The same protocol must be implemented by all participating simulation components. Other related distributed algorithms may also be needed to support the synchronization protocols. For example, global virtual time algorithms in the case of the Time Warp optimistic synchronization protocol.

Table 1 below summarizes the features of HLA and CORBA with regard to the four designated comparison criteria. Other differences exist between HLA and CORBA [9]. For example, HLA provides publishing and subscription services but does not support direct object communication as in CORBA. In addition, the HLA's transfer of object ownership capability between federates has no counterpart in CORBA.

Table 1. Differences between HLA & CORBA

	HLA	**CORBA**
Component interface description	OMT	OMG IDL
Communication infrastructure	RTI	ORB
Component directory service	RTIExec/ FedExec for federates. Publish-and-subscribe mechanism for objects	DII and the interface repository
Time management scheme	Time management services of the RTI	None (user-defined synchronization scheme)

5. Conclusion

With the proliferation of the web, a new computing model is emerging that calls for the development of software from distributed software components that can be combined together to build new applications. HLA and CORBA are two architectures

that can be used to realize component-based simulation. HLA is more oriented towards simulation and provides more simulation-related services for the developer. CORBA supports most major language bindings, and provides better interoperability features when components are implemented in different languages.

An attempt has been made to highlight the differences between HLA and CORBA as potential infrastructures for component-based simulation on the basis of their conceptual foundations and software architecture. Four comparison criteria were defined and used as a basis for the comparison. It can be concluded from the discussion that both HLA and CORBA provide equally powerful *component interface description* methods, although HLA's OMT addresses the needs of the simulation community more directly. However, CORBA's OMG IDL supports more language bindings and provides for better interoperability of simulation components.

In terms of the *communication infrastructure*, the RTI software and Interface Specification seems to be less mature than CORBA's ORBs, especially in communication protocol compatibility between ORB's from different vendors. This is attributed mainly to the wide industrial support that CORBA enjoys, the relatively older age of the technology, and the much wider application scope. This situation is also behind the more superior and more sophisticated *directory services* offered by CORBA. However, HLA supports important mechanisms that are particularly useful in simulation, such as the publish-and-subscribe mechanism and the transfer of object ownership capability between federates.

Finally, CORBA-supported component-based simulation suffers from the serious lack of simulation *time management mechanisms,* which are elaborately defined and provided in HLA. This may be the most important difference between the two technologies which may make HLA a better choice for the developer at present, given the expertise and effort required to incorporate distributed simulation synchronization schemes into CORBA-supported component-based simulations.

In brief, although CORBA seems like a more mature and robust technology that appeals to component-based simulation developers, HLA addresses and elaborately solves difficult issues and problems related to the interoperability of simulation components. This point qualifies HLA, at least for now, to be a better choice as an infrastructure for component-based simulation. More efforts are needed to utilize CORBA's capabilities in the component-based simulation domain and to adapt it to the domain's needs and special requirements.

References

[1] Dahmann, J. S., Kuhl, F. and Weatherly, R. "Standards for Simulation: As Simple as Possible but Not Simpler: The High Level Architecture for Simulation." *SCS SIMULATION*, 6, No. 1 (1998), 378-387.

[2] The DMSO web-site: *http:\\hla.dmso.mil*

[3] Siegel, J. "OMG Overview: CORBA and the OMA in Enterprise Computing." *Communications of the ACM*, 41, No. 10 (October 1998), 37-43.

[4] Siegel, J. *CORBA Fundamentals and Programming*. New York: John Wiley & Sons, Inc. (1996).

[5] Beugnard, A., Jezequel, J. M., Plouzeau, N. and Watkins, D. "Making Components Contract Aware." *IEEE COMPUTER*, 32, No. 2 (July 1999), 38-45.

[6] Shen, C. "Discrete-Event Simulation on the Internet and the Web." *Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation* (1998).

[7] Iazeolla, G. and D'Ambrogio, A. "A Web-based Environment for the Reuse of Simulation Models." *Proceedings of the 1998 International Conference on Web-Based Modeling and Simulation*, (1998).

[8] Soliman, H. M. "Parallel and Distributed Simulation: Methodologies and Techniques." *Journal of King Saud University (Computer and Information Sciences)*, 10, No. 3 (A.H. 1418/1998), 27-51.

[9] Buss, A. and Jackson, L. "Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI." *Proceedings of the 1998 Winter Simulation Conference* (1998), 819-825.

أطر للمحاكاة المعتمدة على المكونات

حسام محمد سليمان

قسم نظم المعلومات، كلية علوم الحاسب والمعلومات،

جامعة الملك سعود، ص.ب: ٥١١٧٨ ، الرياض ١١٥٤٣ ، المملكة العربية السعودية

(قدّم للنشر في ٢٠٠٠/٠٢/٠٩م؛ وقبل للنشر في ٢٠٠٠/٠٩/٠٥م)

ملخص البحث. دعت الحاجة إلى خفض تكلفة تطوير نماذج المحاكاة إلى محاولات حديثة لوضع قياسات تدعو إلى إعادة استخدام النماذج والتشغيل البيني لها. بالتحديد،تم تبني قياس HLA الجديد والمدعوم من وزارة الدفاع الأمريكية؛ ليكون المعمار القياسي لجميع نماذج المحاكاة العسكرية. في نفس الوقت، قام القطاع التجاري بجهود ناجحة لتطوير تقنيات محكمة للحوسبة الموزعة، وبالتحديد معمار CORBA من مجموعة OMG. و CORBA هي مجموعة ضخمة من المواصفات والاتفاقات الإتصالية تعتمد على المفاهيم الشيئية لتحقيق بيئة شيئية موزعة للحوسبة تسمح بإعادة الاستخدام والتشغيل البيني.

وعند استخدام HLA و CORBA كبنية تحتية لإعادة استخدام نماذج المحاكاة والتشغيل البيني لها، يظهر لكل منهما حسنات ومساوئ. وبما أن كل منهما تم تطويره باستقلال عن الآخر، فإن هناك حاجة إلى مقارنة تقيم كليهما كمعمار للمحاكاة المعتمدة على المكونات. وفي هذا البحث، نقدم كل من HLA و CORBA في سياق تطوير المحاكاة المبنية على المكونات والتشغيل البيني لهذه المكونات. كما نعقد أيضاً مقارنة بينهما على أربعة محاور تتعلق بأسس التصميم لكل منهما.