Net Centric Modelling and Simulation Using ActorDEVS

F. Cicirelli, A. Furfaro, A. Giordano, L. Nigro
Laboratorio di Ingegneria del Software (www.lis.deis.unical.it)
Dipartimento di Elettronica Informatica e Sistemistica
Università della Calabria
87036 Rende (CS) – Italy

Extended Abstract

Summary Under the perspectives of the DEVS-World project, whose goal is the development of a net-centric modelling and simulation (NCMS) infrastructure having the net as *the* computer, thus favouring different levels of interoperability among research groups operating world wide, this paper proposes an architecture based on web services for NCMS using ActorDEVS. ActorDEVS is a lean and efficient agent-based framework in Java supporting modelling of Parallel DEVS systems under both centralized and distributed simulation. A key point of ActorDEVS is its support of application-dependent control engines. The paper discusses some architectural scenarios for wrapping ActorDEVS in the DEVS-World infrastructure, opening to interoperability with other DEVS or (possibly) non-DEVS systems. The proposal clearly separates model and simulation concerns. An entire model is partitioned among a number of simulation nodes with web services, in a case, which act as the transport layer for inter-node message exchanges. A global coordinator with a minimal interface of operations governs the "in-the-large" simulation aspects.

Key words: M&S using the Internet, agent-based DEVS, web services, interoperability, separation of concerns

Introduction

The DEVS-World project [9] aims at developing a world-wide standard platform for modelling and simulation (M&S), promoting collaborative research and experimentation in the engineering, i.e. design, evaluation, implementation, deployment and execution of complex, scalable, dynamic structure systems [10] belonging to diverse and significant problem domains like biology and bioinformatics, environment systems, traffic simulation etc. The project has its strength in the use of DEVS [16] as the unifying M&S formalism and an exploitation of nowadays software technologies and middleware such as agents [1,17,3] and services [13,6], which are a key for software interoperability. The main goal is enabling the exchange of both models and experiments among researchers and developers operating in academic or industry labs, thus favouring cooperation.

In this paper the ActorDEVS [4,5,7] framework is put under the perspective of DEVS-World in order to identify possible extensions and cooperation scenarios. ActorDEVS (see Fig. 1) is a lean and efficient agent-based framework in Java supporting modelling of Parallel DEVS systems under both centralized and distributed simulation. The approach clearly separates modelling from simulators, middleware and hardware.

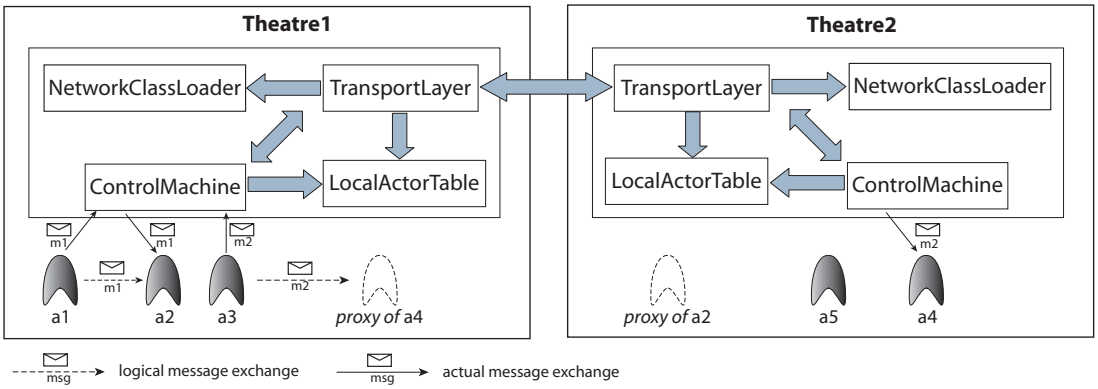

Fig. 1. Actor/Theatre architecture for ActorDEVS

Both simulation and real-time execution modes are supported for model continuity which rests on the possibility of changing the control engine and ultimately the time notion regulating the evolution of the application. The approach is *control-centric*, in the sense that it allows customizing the control machine (Fig. 1) which offers basic scheduling and dispatching message services to actor components.
Key factors underlying ActorDEVS are the adoption of actors [1,8] as programming in-the-small building blocks, and of theatres [3] as programming in-the-large execution loci (see Fig. 1). Adopted actors are thread-less reactive objects which encapsulate an internal data state (which include *acquaintances*, i.e. known actors which can be contacted by messages), have a behaviour patterned as a finite state machine, and communicate to one another by asynchronous message passing. Actors can migrate dynamically from a theatre to another for reconfiguration purposes.
ActorDEVS is supported by a minimal API in Java. Typed input/output ports are mapped on to actor messages. Configuration operations correspond to updating receiver information in output ports, also during the runtime. More in general, changing actor's acquaintance network, a concept which is often referred to as link mobility, is a natural way to achieve model structure dynamism [3,8,7]. Good execution performance is ensured by having a DEVS model is flattened from the point of view of the simulation engine.
The paper is structured as follows. Subsequent section introduces the DEVS-World project and its objectives and key features. After that, a description is provided of how Theatre/ActorDEVS architecture can be extended for supporting NCMS. Last section discusses some issues relevant to a pragmatic use of the resultant architecture. Finally, future and on-going work is summarized.

An Overview of DEVS-World

Novel in the DEVS-World project is the definition of a development methodology for supporting world-scale distributed *open* systems of systems M&S [9]. Openness is a fundamental property which expands along different directions with different levels of integration and interoperability.
A first level of integration is relevant to model interoperability. Many different implementations of DEVS simulators currently exist, and usually each of them uses a built-in modelling language often tied to a specific programming language like Java or C++. To cope with this problem, specific conversion tools capable of translating a DEVS model from a language to another can be realized. A more general solution would be that of adopting emerging DEVS standard language such as DEVSML [9].
Another direction of integration concerns interoperability at architectural level. In [9] but also in [12] the proposed world-wide architecture is aimed at harmonizing heterogeneous models based on special-case DEVS tools, programming languages and engines, through the use of Web Services and SOAP dependent messages and other DEVS concepts (ports, simulators, coordinator etc.). Web Services are viewed as a world-wide *glue* enabling interoperation through DEVS/SOA mechanisms, with WSDL used for web services interface specification.
Besides standardization of models and simulation infrastructure, the definition of a standard simulation protocol [14] is mandatory. The protocol (see Fig. 2) describes how a DEVS model should be simulated and how service/simulation engines should coordinate each other. Such a protocol opens also to a scenario in which both DEVS and non-DEVS simulators may (possibly) participate in a simulation.

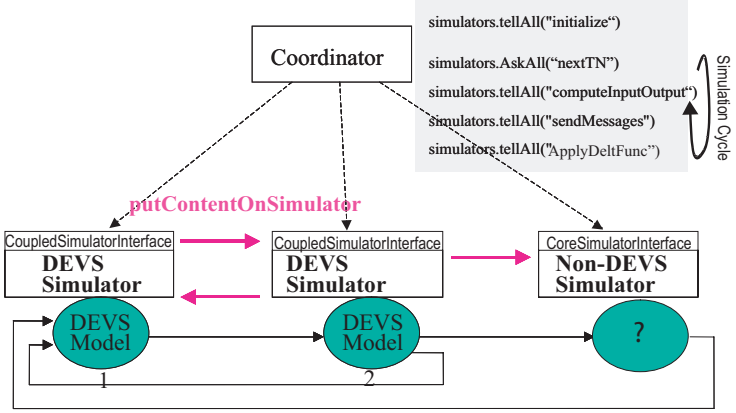

Figure 2. Simulation protocol in a federation of DEVS and non-DEVS simulators

CoreSimulatorInterface is the common interface to simulators. The term “core” means “essential” in that as long as a simulator implements this interface, it can participate in a simulation driven by a DEVS coordinator. In the case of DEVS-simulators, the *CoupledSimulatorInterface* is considered. This interface extends the core interface by providing other functionalities e.g. for adding/removing couplings among DEVS models.
The coordinator is in charge of synchronizing the activities of the various simulators guiding them through the simulation control cycle. Basic steps of the simulation cycle are shown in Table 1.

Step	**Description**
nextTN	the coordinator requests that each simulator sends its time of next event and takes the minimum of the returned values to obtain the global time of next event
computeInputOutput	each of the simulators applies its *computeInputOutput* method to produce/gather an output that consists of a collection of *Contents* (i.e. port/value pairs)
sendMessages	each simulator partitions its output into messages intended for recipient simulators and sends these messages to these recipient simulators. Send a message imply to call the recipient’s *putContentOnSimulator* for any target simulator
applyDeltFunc	each of the simulators executes its *ApplyDeltFunc* method which computes the combined effect of the received messages and internal scheduling on its state, a side effect is in producing the time horizon gives back at the nextTN

Table 1. Simulation cycle steps

CoordinatorInterface must be implemented by the coordinator. In handling simulation of hierarchical coupled models, a coordinator orchestrates a set of controlled simulators within it and, at the same time, can participate with peers in a coupled model above it. To allow such downward/upward facing interfaces, the *CoupledCoordinatorInterface* is introduced which extends both the *CoordinatorInterface* and the *CoupledSimulatorInterface*.

Wrapping ActorDEVS in DEVS-World

This section highlights a service-based approach extending the Theatre/ActorDEVS architecture in order to meet requirements of DEVS-World project. Provided extensions support architectural interoperability among heterogeneous DEVS simulators. The approach adopts previously described DEVS simulation protocol. At the moment, interoperability at modelling language level is not addressed. Each DEVS model is assumed to be implemented as a Java class complying with the ActorDEVS API [7].
The main idea is to wrap a whole ActorDEVS system, which can span from a single atomic model to a complex coupled model, as a Web Service (i.e. a logical *node*) and associated simulation engine. A Coordinator service is introduced in order to coordinate the evolution of the overall simulation. System components are made available as Web Services by means of specific objects called Wrappers. Client-side interactions are instead mediated by means of specific Proxy objects. It is worthy of note that in a service oriented architecture the roles of client and provider are not univocally defined, in particular the same node may act as client/provider on the basis of the required/offered functionalities.
Wrappers and Proxies are transparently used. As a consequence, would e.g. Java RMI be used in place of Web-Services based protocols, only Wrappers and Proxies would be accordingly changed. Figure 3 shows the architecture of a resultant Theatre/ActorDEVS system.

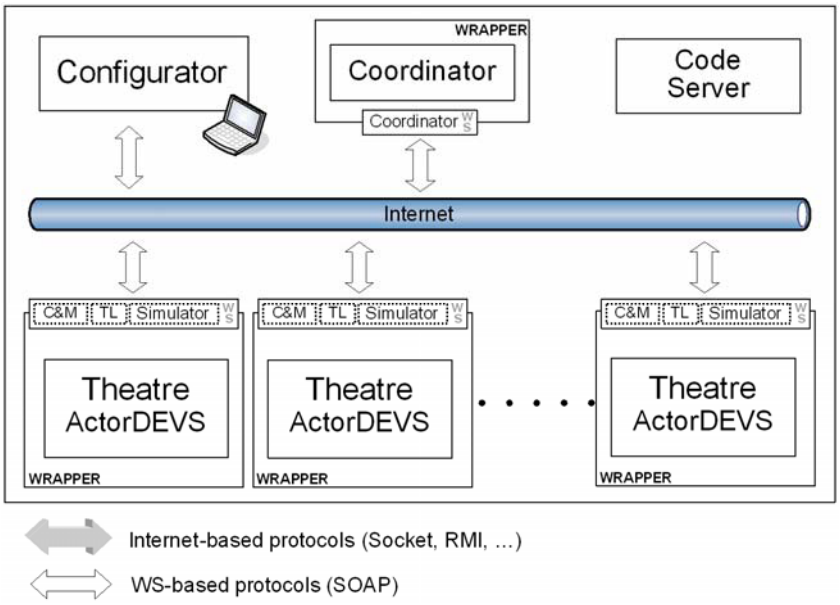

Fig. 3. Architecture of a Theatre/ActorDEVS system.

The Theatre nodes, the Configurator and Code Server nodes are not exclusive to a DEVS simulation, they are common to all Actor-based applications. The Coordinator node, instead, is tightly related to DEVS-World prospective and is in charge of implementing the DEVS simulation cycle. The Configurator makes it possible to configure the whole simulation system and start simulation. Configuration consists of four steps.

The first step is devoted to setting-up the Theatre nodes by specifying the control machine, the code server and the transport layer to use. This is accomplished by exploiting the Configuration and Management Web Services (see the C&M-WS in Fig. 3). A specific *DEVSControlMachine* has been developed in order to work in pair with the Coordinator and be compliant with the DEVS simulation protocol. This control machine implements a *CoupledSimulationInterface*–like and behaves as a DEVS simulator. When the control machine is instantiated its functionality is made available as a Web Service which is automatically published (see the Simulator-WS in Fig. 3). The *DEVSControlMachine* oversees message exchange with other simulators. As a consequence, the transport layer (see the TL-WS in Fig. 3) in this scenario is used only to manage inter-theatre control messages.

The second step consists in assigning to each Theatre the specific DEVS model to simulate. A model may correspond to a single atomic or coupled DEVS component. A model corresponds to a Java class whose name requires to be specified. This step is carried out by exploiting the C&M-WS and completes when each model gets assigned to the target Theatre, i.e. downloaded from the code server and instantiated.

The third step consists in establishing the necessary bindings among coordinator and simulator services (i.e. acquaintance relationships). In particular, a *CoordinatorInfo* object is provided to each simulator and a list of all *SimulatorInfo* objects, relevant to simulators involved in the federation, is furnished to each simulator and to the coordinator. An info object contains the name of the service and the relevant service endpoint address which is necessary to contact and use it.

The fourth step consists in defining couplings among deployed models in order to build the entire simulation model. This is achieved by invoking the method *addCoupling* onto a simulator. Coupling information mainly contains a couple of names, identifying the two ports to be connected. The first name is relevant to an output port of a DEVS component local to the simulator. The second name is relevant to an input port of a DEVS component residing on a remote simulator. Along with coupling information, the name of the remote simulator is also provided. At runtime, couplings get actualized by means of the so called *RelayPort* objects. Making a coupling implies linking an output port of a DEVS component to a relay port which, in turn, is logically connected to a remote input port. All of this makes the DEVS Component unaware of network partitioning.

At configuration end, the Configurator may launch the simulation by calling the method *simulate* on the Coordinator which in turn triggers into execution the simulation control loop.

An Example

[*Full version of the paper will include details about a modelling and simulation example using the achieved implementation of WS-based Theatre/ActorDEVS*.]

Conclusions

A prototype version of the Theatre/ActorDEVS architecture based on Web Services was realized. The implementation relies on Java technology. In particular, the SOAP engine Axis [2] is used for managing WS related aspects. The following are some points which deserve some discussion within the community of DEVS-World.

- The DEVS simulator protocol appears "too much synchronous" for a networked context. Many interactions among the simulation-protocol participants are required for each simulation step independently from the complexity of the simulated model. A systematic exploitation of a kind of "lookahead" could alleviate the problem. By exploiting lookahead the coordinator could give a granted time to each simulator thus allowing a more independent evolution of local simulation.
- Another (obvious) issue concerns simulation performance achievable by the use of WSs. This is not only tied to the use of verbose XML for SOAP messaging but mainly to the management of network connections. Simulation experiments confirmed that network resources (connections) of operating system may be wasted considerably during simulation and need in general careful control.

On-going work is directed at:

- improving the Configurator component by providing a friendly GUI for visual system configuration, model composition, deployment and simulation control
- replacing Axis by other Web Service infrastructure e.g. related to latest J2EE
- introducing a model repository service, enabling model reuse and sharing
- adopting standard formalisms like DEVSML for supporting DEVS modelling
- favouring model and experiments interchange by developing translation tools allowing model transformation from a high-level implementation-independent formulation into the terms of a specific DEVS setting (e.g. ActorDEVS and Java) and vice versa
- experimenting with Theatre/ActorDEVS architecture in an heterogeneous environment where diverse DEVS simulators have to cooperate
- developing tools for visual modelling.

References

[1] Agha G., *Actors: A model for concurrent computation in distributed systems*, Cambridge, MIT Press, 1986.

[2] Axis website. http://ws.apache.org/axis/index.html. Accessed on May 2008.

[3] Cicirelli F., A. Furfaro, A. Giordano, L. Nigro. An agent infrastructure for distributed simulations over HLA and a case study using unmanned aerial vehicles. In *Proc. of 40th Annual Simulation Symposium*, IEEE Computer Society Press, pp. 231-238, March, Norfolk (VA), 2007.

[4] Cicirelli F., A. Furfaro and L. Nigro. A DEVS M&S framework based on Java and actors. In *Proc. of 2nd European Modelling and Simulation Symposium (EMSS 2006)*, pp. 337-342.

[5] Cicirelli F., A. Furfaro, L. Nigro. Conflict management in PDEVS: an experience in modelling and simulation of time Petri nets. In *Proc. of Summer Computer Simulation Conference (SCSC'07)*, pp. 349-356.

[6] Cicirelli F., A. Furfaro, L. Nigro. Integration and interoperability between Jini services and web services. In *Proc. of IEEE Int. Conf. on Services Computing (SCC'07)*, pp. 278-285, July 2007.

[7] F. Cicirelli, A. Furfaro, L. Nigro. Actor-based Simulation of PDEVS Systems over HLA. In *Proc. of 41st Annual Simulation Symposium (ANSS'08)*, Ottawa, Canada, April 14-16, 2008, pp. 229-236.

[8] Cicirelli F., A. Furfaro, L. Nigro, F. Pupo. A component-based architecture for modelling and simulation of adaptive complex systems. In *Proc. of 21st European Conference on Modelling and Simulation (ECMS'07),* 4-6 June, Prague.

[9] DEVS_WORLD: A platform for developing advanced discrete-event simulation at worldwide scale, *Internal document*, December 2007.

[10] Hu X., B.P. Zeigler, S. Mittal. Variable structure in DEVS component-based modelling and simulation. *Simulation*, vol. **81**(2):91-102, 2005.
[11] Hu X., B.P. Zeigler. Model continuity to support software development for distributed robotic systems: A team formation example, *J. of Intelligent and Robotic Systems*, vol. **39**(1):71-87, 2004.
[12] Mittal S., B.P. Zeigler, J.L.R. Martin, F.Sahin, M. Jamshidi. Modeling and simulation for systems of systems engineering, Book chapter, System of Systems – Innovations for the 21st Century, Wiley, 2008 *(in press)*.
[13] Papazoglou M.P., D. Georgakopulos. Service Oriented Computing. *Communication of the ACM*, vol. **46**(10):25-28, 2003.
[14] Xiaolin H., B.P. Zeigler. A Proposed DEVS Standard: Model and Simulator Interfaces, Simulator Protocol, *Internal document*, January 2008.
[15] Yu Y. H., G. Wainer. eCD++: an engine for executing DEVS models in embedded platforms. In *Proc. of SCS Summer Simulation Multiconference*, pp. 323-330, 2007.
[16] Zeigler B.P., H. Praehofer, T.G. Kim. *Theory of modeling and simulation*. 2nd edition, New York, NY, Academic Press, 2000.
[17] Wooldridge M. *An introduction to multi-agent systems*. John Wiley & Sons, Ltd., 2002.