[Devs]Server: Ambient Intelligence And DEVS Modeling Based Simulation Server

Mostefa Mokaddem[a,b]
[a] Computer Science Department,
Fac. of Exacts & Applied Sciences
[b]LIO lab
Univ. of Oran 1 Ahmed Ben Bella
mokaddem.mustapha@univ-oran.dz

Baghdad Atmani[a,b]
[a] Computer Science Department,
Fac. of Exacts & Applied Sciences
[b]LIO lab
Univ. of Oran 1 Ahmed Ben Bella
atmani.baghdad@univ-oran.dz

Abdelmalek Boularas
Computer Information System Department
Ahmed Bin Hamed Military College
Doha, Qatar
boularas@abmmc.edu.qa

Abstract
To improve disease surveillance systems with faster and more accurate outbreak detection and epidemics propagation capabilities, the availability of fine-tuned models is required and server based solution, simulating the effects of public health authorities' measures must be designed, integrating Ambient Intelligence (AmI) capabilities to semantize epidemic models. Hosting DEVS models, these AmI servers and their communication protocols are different, miscellaneous and require interoperability. The Triple Space Computing (TSC) paradigm addresses interoperability by sharing information represented in a semantic format through a common virtual space. We have to discuss the main characteristics of existing synchronization methods for SOA oriented simulation, within the TSC paradigm. As long as DEVS modelers add standard ontologies to design models, these models using the same spaces will interact and communicate automatically. The focus of this paper is to present [Devs]Server, a fully distributed TSC simulation server solution (middleware) designed to meet the needs of parallel and distributed discrete event simulation. [Devs]Server defines an SOA interface [2] for the TSC operations. This interface convinces with DEVS formalism and focuses on simplicity, conviviality and modularity, so that a single or many simulations that support different models can still interact.

Keywords
Ambient intelligence, Semantic web, Web of things, Tuple space, Space-based computing, Service Oriented Simulation, DEVS Formalism, Epidemic modeling, Digital library, Restful Web service.

INTRODUCTION

In recent years, contamination and its interaction with huge flow of quantitative social, demographic and behavioural data is used to improve disease surveillance systems (DSS) with faster and more accurate outbreak detection and epidemics propagation capabilities which depend on the availability of fine-tuned models. Epidemic Modeling and computational infrastructures, such as SOA oriented architecture [2], enable creating very detailed representations. With accurate models we can predict the outbreak detection, the spread of diseases and simulate the effects of public health authorities' measures.

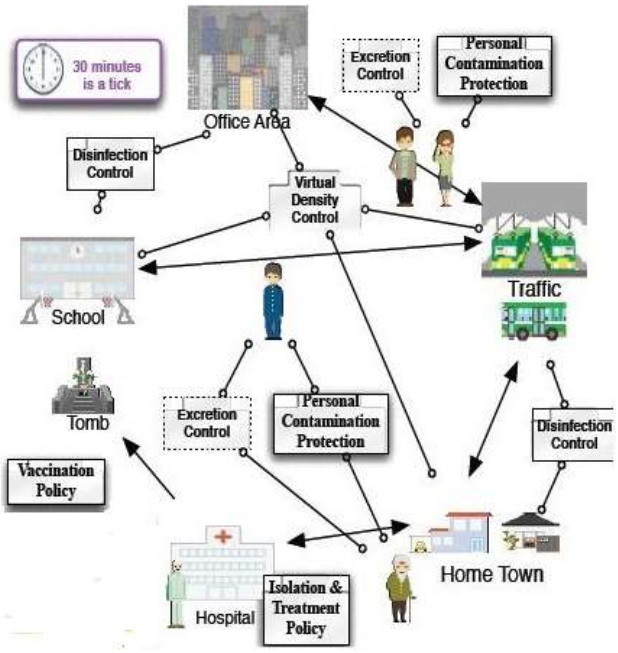

Figure 1: disease simulation in a virtual city

Figure 1 shows a human interaction model in a modern society. Ambient Intelligence (AmI) is widely used in such system capturing required information. Spatial and social network structures influence infectious disease transmission within populations. Due to significant changes in health contexts, the SEMEP, figures 2 and 3 [6], needs designing new DSS with epidemic modeling complements integrating AmI capabilities. AIR researchers involved in such design issues are developing a specific modeling platform to help model, simulate and evaluate DSS. Allowing the design of atomic and coupled models, they integrate Epidemic Modeling Digital Libraries (EMDL) [26] within simulation servers. EMDL holds models as RDF graph to semantize modeling. The knowledge associated to models describes the howto of models. Simulation servers need to provide modeling/simulation with semantics about the requested models. To pursue this goal, simulation servers use EMDL to manage models while describing models with ontologies. Knowledge (ontologies) may be shared between modelers.

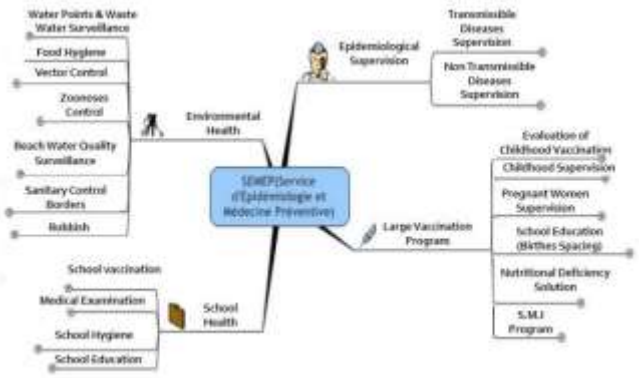

Figure 2: SEMEP cartography

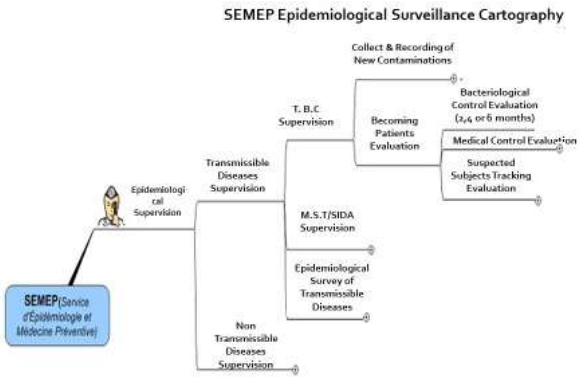

Figure 3: SEMEP epidemiological surveillance process

Simulation servers perform a distributed simulation execution requiring interoperability and natural and transparent interactions that are important in AmI to defend the fact that servers should subtly work on behalf of the human tasks and to minimize the psychological impact of servers use. Furthermore, AmI based servers can be used in the same way in simulation to avoid modelers from doing low-level but yet time-consuming modeling tasks such as interoperability. Modelers can now focus on modeling with a high aggregate-value, where the importance of the human capital is vital [10,25]. To achieve these aims, AmI based servers need to integrate and coordinate heterogeneous data sources or service providers. Current trends, such as the Web of Things (WoT) initiative [11], propose a straightforward integration of servers with the web using Restful services. Independently of the used model, the messages usually exchanged between servers are diverse and simulation session dependent. This implies that messages will not be meaningful in other simulations unless a specialized system converts and reinterprets them. A way to solve this problem is annotating the message semantically as proposed by the WWW [5,10].

Let us consider two coupled epidemic models AB and AC with atomic models A, B and C according to DEVS formalism, figure 4. Each of A, B and C is stored in a separate simulation server and invoked via Restful web services. A is participating in two parallel simulation sessions at the same time; AB and AC simulations are sessions performed on separate servers by two different clients. Servers store models in respective EMDL. Each epidemic model consumes information from its own EMDL [26]. Thus, A, B and C are a data mining based model where a mining of some data is used to generate output as described in[4,19-20]. Data are in the same EMDL as the model using them. A and B are described with ontologies to allow a modeler to link servers hosting these models while trying to use each of them as sub-model of its own coupled model. Many modelers may realize a coupled model conjointly. AB, A and B and AC, A and C, respectively, represent exchanged information as a Friend of a Friend (FOAF) ontology. This ontology is designed as a shared space between these models.

In figure 4, two shared spaces are used by the two sessions. In each shared space, models output RDF triples to be added to the shared space. Models may read triples into the shared space and perform their transition functions. A is participating in the two sessions; it uses the shared space according to the session. Finally, it stores such ontology in the modeler's server Triple Spaces. Since Triple Spaces is used as shared space, this knowledge is available for other servers using that shared space.

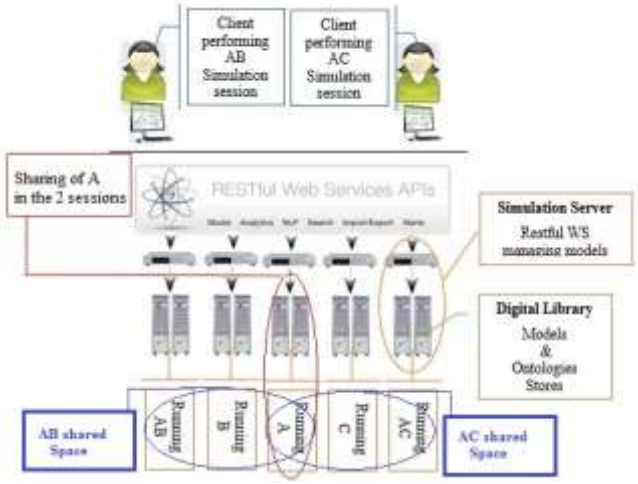

Figure 4: simulation sessions with model sharing

Another model (coordinator) may notify the other models or users that there is a friend in this simulation session or in this shared space. This coordinator may populate the shared space with information produced by different other models [28]. Prior to the information retrieving, the coordinator semantizes the information according to the FOAF ontology. Finally, it periodically looks into the space to check whom is friend of this model. Data mining based model uses preview data as follow: data are organized in a view from databases inside digital libraries. Each row of this view is an entity. Many columns are properties of this entity. We apply KDD on these data to extract rules that will be used by the model [4,19-20]. Therefore, a modeler begins by applying KDD to discover

knowledge that is used to define transition functions of the model. The interoperability is achieved when AC model, which does not support the AB digital libraries, automatically discovers a friend who is using the first model. This is possible because both clients may share information in a common space and use the same ontology.

In order to achieve this interoperability through Triple Spaces, we propose a simulation Server middleware solution called [Devs]Server. This solution provides two core features: (a) it is designed to be simple, modular and extensible and (b) it runs in different computational platforms, allowing Java SE, Java Mobile and Android interaction. The underlying interface is based on SOA[18] and covers isolated features such as discovery, maintenance or data access. Different simulations can provide only certain features and still interact with each other. This way, it is possible to embed it in other real-time simulations using digital devices or Sensors. Triple Space Computing (TSC) has been involved as a coordination paradigm supporting indirect communication based on semantic data. As simple as possible, a model writes semantically annotated information in a shared space that is queried out and used by other models.

Parallel Discrete-Event Simulation (PDES) has received increasing interest as simulations become more time consuming and geographically distributed. A rich literature has already been developed in the last three decades, taking advantage of the increasing availability of parallel and distributed computing platforms, especially on emerging platforms such as many-core processors, internet scale simulation environments, and cloud-based virtualized infrastructures [1,14].

In parallel and distributed simulations, the entire simulation task is divided into a set of smaller subtasks with each executed on a different processor or node. Hence, the simulation system is viewed as a collection of concurrent processes, each modeling a different part of the physical system and executing on a dedicated processor in a sequential fashion. These processes communicate with each other by exchanging time-stamped event messages. An event refers to an update to simulation system state at a specific simulation time instant. Throughout the simulation, events arrive at destination processes. Depending on the delivery ordering system of the simulation, they are processed differently. The two commonly used orderings are (1) receive-order and (2) timestamp-order. With the first type, events are delivered to the destination processes when they arrive at the destination. On the other hand, with the timestamp-order, events are delivered in non-decreasing order of their timestamp, requiring runtime checks and buffering to ensure such ordering. To ensure correct synchronization, a PDES system must satisfy the following necessary and sufficient condition, termed as the Local Causality Constraint by Fujimoto [1,7,14]. To satisfy the local causality constraint, different synchronization techniques have been proposed for PDES systems which generally fall into two major classes of synchronization: conservative, which strictly avoid causality violations; and optimistic, which allow violations and recover from them. In the past three decades, different researchers in this field have proposed numerous approaches. [1,14] summarize these approaches focusing on emerging platforms such as Web services, clouds, and heterogeneous multicore processors. As well as some popular PDES environments followed by their hybrid synchronization techniques. In SOA oriented simulations, the entire simulation task is divided into a set of services (each model is handled by a service) with each executed on a different server.

The rest of the paper is organized as follows. Section 2 outlines related work. Section 3 describes the conceptual model for an SOA based TSC solution, let's say [DEVS]Server principles. Section 4 details the implementation made to adapt it to the necessities of AmI needs. Finally, Section 5 concludes and discusses future work.

RELATED WORK

In the following subsections, we analyze both semantic solutions of interoperability involving mobile and embedded devices and concluding with TSC paradigm, and the SOA based modeling and simulation as synthetized by Al-zoubi and Wainer [1]. Among the analysis, we compare our solution with the rest emphasizing their strengths and weaknesses.

Ambient Intelligence Interoperability

Regarding REST, its use in resource constrained devices is a current trend defended by the WoT initiative [11]. WoT proposes to embed web servers in everyday things. These objects expose their capabilities following the REST principles. In this way, they fully integrate with the web. This has several benefits: Availability of digital libraries and frameworks in most of the existing computing platforms. Reuse of mechanism that have made the web truly scalable. E.g. searching, caching, load-balancing or indexing. The users can interact with the objects through a familiar tool: the browser. They can browse or bookmark them, share on social networks, etc. Direct integration with other web applications.

Tuple Space (TS), also called space-based computing, is a coordination paradigm based on the shared memory approach [8]. TS works with semi structured data, which is accessed in an associative manner. Several TS solutions have used semantics to enhance the shared data [19]. sTuples was conceived for scenarios [15]. In sTuples, the clients access a centralized space through a communication gateway. The centralization completely simplifies the solution, but makes the whole system dependent on a single machine. Besides, Otsopack [10] avoids the need of gateways by requiring a prominent protocol (i.e. HTTP) for the communication

between the nodes. TripCom4 distributes the space among different super-peers using distributed hash tables. Specifically, it uses a hash function over the subject, predicate, object and space URL to decide where to store each triple. TripCom draws a clear distinction between the clients, which consume data, and the devices where the space resides. Otsopack also promotes the direct communication between devices. Doing so, they can access to the most updated data and manage their own data. Finally, Smart-M3 [13] constitutes a remarkable effort to bring the semantic space-based computing to many different devices and protocols. To that end, it distinguishes between two types of nodes: Knowledge Processors (KPs) and Semantic Information Brokers (SIBs). The SIBs manage the space. The KPs are nodes accessing the space information. The Smart Access Protocol (SSAP) is used for the communication between both types of nodes. Although theoretically possible, to the best of our knowledge no results have been presented on the federation of two or more SIBs. This makes the solution de facto centralized. Apart from the distributed nature of our work, it also avoids the definition of any new communication protocol. Instead, it assumes that all the nodes will be able to work at HTTP-level or have a gateway to do so on their behalf. Thanks to that and to the prominence of libraries and tools for this protocol the implementation on new platforms is greatly simplified.

As was previously stated, our API is based on the TSC paradigm. TSC is a TS variation where the information is stored in RDF. Three key concepts are important at this point: models share information in a common space. A space is identified by an URI. Therefore, all the operations in TSC are performed against a particular space. By default, all simulation sessions connect to a common standard space, but they can optionally choose to connect to a particular private space. Within a space, the information is stored in sets of triples called graphs. Each graph can also be identified by an URI. The RDF triples are the underlying concept of all the Semantic Web (SW) languages. Each triple is composed by a subject (which is a URI), a predicate (also a URI) and a value (which can be a URI or a literal). As detailed later, the operations supported attempt to add or remove graphs, as well as to query for graphs or for sets of triples retrieved from different graphs. In order to perform the queries, which enable the selection of a subset of the semantic content hold in a given space, a template is required. We follow Otsopack to address these operations. [9] presents further discussion about knowledge distribution strategies.

Modeling and Simulation Interoperability

Wainer [1] presents a best simulation interoperability concept background, and describes his RISE middleware that fits within this concept. Wainer objectives were to enhance interoperability, by decoupling/hiding implementations. He highlighted some guidelines to be followed to a general Web-based middleware container. Interoperability, as Wainer stated, enables two or more different software systems to interface and use each service correctly [27]. The complexity of interoperability arises when systems are heterogeneous, as in the case of distributed simulation. This is usually because systems have been developed independently with different semantics (i.e. the meaning of the exchanged information) and/or syntactic (i.e. the rules of structuring and exchanging the information). Since such capabilities are realized in software design and implementation, interoperability needs to be studied from the software perspective, in particular, at the Application Programming Interface (API) level (since this is how systems access and use other systems services).

Wainer presented RISE middleware as a layered architecture where each layer defines its interoperability methods, and provides services to the layer above it. Following this concept, RISE is organized in the following layers: middleware, the simulation, and modeling. The middleware layer provides a number of services to the simulation layer, such as all means of communication and managing all simulation experiments lifecycle and executions. The simulation layer deploys different simulation environment types, each of which supports its own time management. The modeling layer operates above the simulation layer. This represents the system under study, which is simulated by a specific simulation environment. This RISE model layers match other existing interoperability conceptual layers, particularly the Level of Conceptual Interoperability Model (LCIM).

In RISE, all functionalities are hidden in resources, named with uniform resource identifiers (URIs). Those resources (URIs) are connected to each other via uniform virtual channels in which the simulation synchronization is done using XML messages. Thus, the RESTful interoperability approach allows system designers, to accomplish the following: (1) decompose the systems in components (i.e. called resources/URIs), (2) hide the implementation within those components, hence separating component interfaces from software implementation. These fundamentals were adapted by the RESTful WS style, which was adapted by the World Wide Web (WWW), the largest open computing environment. In contrast, existing simulation interoperability approaches do the opposite to these principles by following procedural programming style, hence mixing systems implementation and interface. By going against the Web interoperability principles will always cause serious difficult interoperability issues when interoperating on the Web with other existing systems. These issues became obvious during the current efforts on standardization of Discrete Event System Specifications (DEVS) [29-31,34]. This standardization effort is aiming on interoperating various DEVS-based implementations systems via the Web [30]. RISE is not a real server based solution and no semantic is

present may be in message nor in models. Uploading model to specified servers to perform simulation is not a full or real SOA based simulation. SOA means that Application (simulation) Servers hold services that are orchestrated over the web. Services are hosted by the servers not uploaded at runtime. So the best solution is to implement simulation servers that host their own models. The Admin of the server develop and update models that are stored in a library. Epidemiologists can develop epidemic models for a full variety of diseases in different formalisms and store these models in digital libraries. These models are ready to use by communities, they have just to integrate them through restful web services in their own work.

Therefore, we have to develop models in a way that they can be invoked by a web service using java reflection. This web service can hold any model if this model has a standard specification such as DEVS. All previously developed models in the DEVS formalism (DEVSJAVA) will be conserved in the server store as they are without any change. Restful web service is an abstraction designed to call DEVS class methods using java reflection (figure 5).

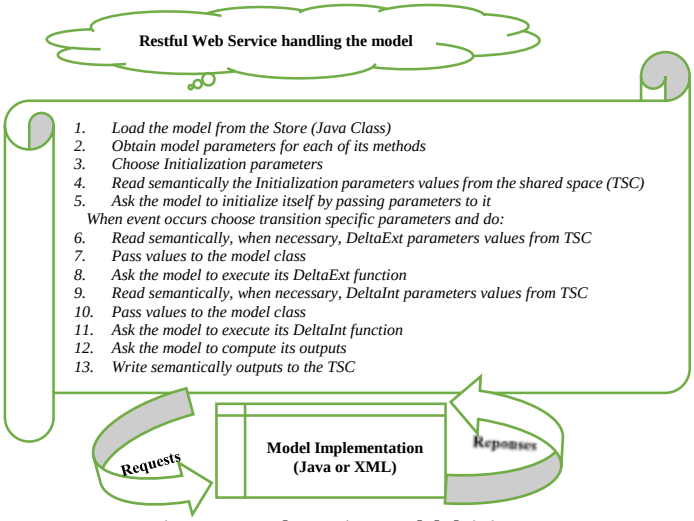

Figure 5: web service model driving

The service first load the class model from the store. It obtains all the model parameters. Each method, let it be *DeltaExt()*, *DeltaInt()*, *Out()* or the *Conf()*, has its proper parameters that differ from one model to another. So, the service ask the class for each method parameters. It must obtain parameters values from the TSC. Thus, these parameters must be annotated semantically. Obtaining parameters, it passes them to the loaded class (DEVS model) and asks it to perform its specific transition function according to the events scheduling. In the TSC, information must be semantically annotated. Especially, messages need to be semantized, designed as graphs. Each Restful WS can easily find all graphs (messages) destined to it. So, a WS takes access to the TSC, looks for its input messages, performs its transitions and generates semantically outputs (graphs) that it writes to the TSC.

DEVSSERVER PRINCIPLES

A node, in DevsServer architecture (figure 6), is composed with a Repository, a Collector and a Mediator. The Repository stores epidemic models, their datasets and ontologies to characterize their semantic information. The Mediator is a collection of web services that will provide access to internal data and external sources, using state-of-the-art semantic-web/grid technologies. The Collector **r**etrieves diseases models and their information from publicly available DevsServer servers. Interfaces enable the Admin of the Server and the Client to perform Modeling and Simulation. Each node hosts its own models developed by its Administrators. Models are kept in a NoSQL database to ensure a quick search of models and to deal with semantics inside models description for a machine adequate use. A node makes use of the TSC paradigm to establish communication between models during simulation. A simulation is performed by many Mediators and Restful web services each one handling specific models. Many nodes interact to perform a distributed simulation execution.

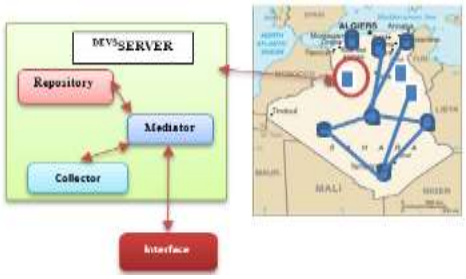

Figure 6: Simulation server

In this section, we detail how models are hosted by DevsServer and how TSC is used to perform an SOA simulation. Models read inputs and write outputs to the TSC parameterized by the Client (DEVS Modeler). Two subsections are detailed. The 1st one concerns the Admin of the server and the 2nd one the client part. The Admin of the server is responsible to add, update or delete models from the server. Any client can perform a simulation while using the models stored in the server. Let us see these 2 tasks in detail involving the Mediator, the Repository and the Collector.

The Admin Tasks

The figure 7 depicts the diagram of the admin operations on the server. The configuration of the server consists of adding and updating users and models. Models need to be described with semantics to let machines (other DevsServer) use them adequately. Semantic is conceived with RDF triples. The semantic of each function must consider parameters of the model and the way they are utilized. A model is described as an RDF graph to be easily added to the NoSQL database. Such graph can be also converted to an ontology.

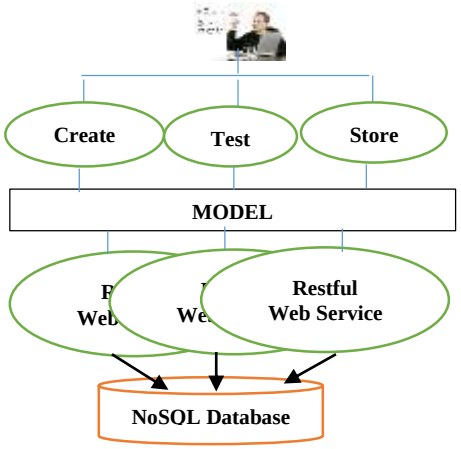

Figure 7: admin server configuration

The following are some RDF triples depicting the graph of a DEVS Dengue of the figure 8.

<u:Dengue> <u:hasOwner> <u:Mokaddem>.
<u:Dengue> <u:hasType> <u:AtomicModel>.
<u:Dengue> <u:hasCreationDate> <u:12/02/2014>.
……..
<u:Dengue> <u:hasInitialize> <u:DengueDengueInitializeIndex>.
<u:Dengue> <u:hasDeltaInt> <u:DengueDeltaIntIndex>.
<u:Dengue> <u:hasDeltaExt> <u:DengueDeltaExtIndex>.
<u:Dengue> <u:hasJavaClass> <u:DengueJavaClassIndex>.
<u:Dengue> <u:hasXML> <u:DengueXMLIndex>.
.....
<u:Dengue> <u:hasInPorts> <number>.
<u:Dengue> <u:hasOutPorts> <number>.
<u:Dengue> <u:hasInPort> <u:Inport1>.
<u:Dengue> <u:hasInPort> <u:Inport2>.
<u:Dengue> <u:hasOutPort> <u:OutPort1>.
<u:DengueInitializeIndex> <u:hasIntputParameter> <u:param1>.
<u:DengueInitializeIndex> <u:hasOutputParameter> <u:param2>.

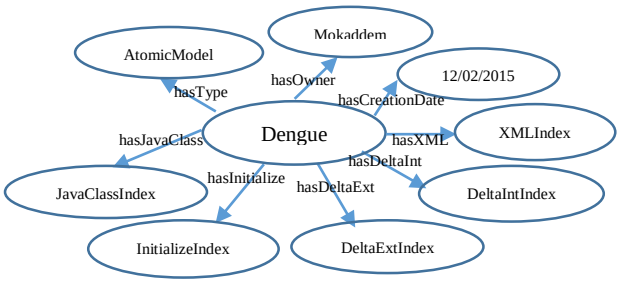

Figure 8: graph Model of a DEVS Dengue Example

Each model owns its functions such as *Initialize()*, *DeltaExt()*, *DeltaInt()*, *Out()* and the *Conf()*. The *Initialize()* function lets the model perform its initialization. Such initialization concerns the time and the states of the model. *DeltaExt()* transition function concerns the action to be performed when some inputs X_1, X_2; etc. are received. Let us notice that the inputs number differs from a model to another. *DeltaInt()* transition function deals with the inside states change. The *Out()* function generates the outputs of the model. The predicate <hasJavaClass> refers to the java code that will be loaded by the Restful web service by java reflection. That means when calling a Restful web service handling any model to initialize itself then the web service just does a java reflection to call at runtime the *initialize()* method of this java class included in this code. This class is inserted as a LOB (Large Object Binary) in the NoSQL database. So, the related web service must first load this class from the store then use it by reflection (figure 9).

```
@XmlRootElement
public class ModelSchema {
String Name; int Inports; int Outports; String Owner; String DateCreation;
String Description; String Inisliaze; String DeltaExt; String DeltaInt;  byte[] Modelclass ;
```

```
@XmlRootElement
public class ModelAccess {
private static Key key = null;     private static KVStore mystore;
public static final String MOD_PREFIX = "DEVS";
```

```
@GET
@Produces(value = { "application/xml", "text/plain" })
@Path("readModelClass/{username}/{modelName}")
public Object readModelClass(@PathParam("username") String username,
@PathParam("modelName") String modelName) {

    ModelSchema model = new ModelSchema();
     model = ModelAccess.getModel(username, modelName);
     Object o = null;
     ByteArrayClassLoader baCL = new ByteArrayClassLoader();
     Class modelClass = baCL.findClass(model.getClasse());
    try {
       o = modelClass.newInstance();
      } catch (IllegalAccessException e) { System.out.println("access impossible ");    }
        catch (InstantiationException e) { System.out.println("instance impossible....");    }
      return o;
    }
```

```
@GET
@Produces(value = { "application/xml", "text/plain" })
@Path("readModelClass/{username}/{modelName}")
public void WSDeltaExt (@PathParam("username") String username,
@PathParam("modelName") String modelName) {
    ……..

try {
     Object o = mv.readModelClass(username, modelName);
      System.out.println("nom de la classe  depuis le main  "+ o.getClass().getName());
      System.out.println("fields.....");
      for (Field field : o.getClass().getDeclaredFields()) System.out.println(field.getName());
      System.out.println("methods.....");
       for (Method method : o .getClass().getDeclaredMethods())
System.out.println(method.getName()+"   parameter type   "+method.getParameterTypes());

        o.getClass().getDeclaredField("state").equals("S");
        System.out.println(o.getClass().getDeclaredField("state").get(String.class));

        System.out.println(o.getClass().getDeclaredMethod("DeltaExt",String.class));
        Method method = o.getClass().getDeclaredMethod("DeltaExt",String.class);
        System.out.println("before invoke..");
        Object r = method.invoke(o,  "infection" );
        System.out.println(r);
     } catch (Exception io) {
        io.printStackTrace();
        System.out.println("model "+  modelName+ "  not found");
     } finally { }
      ModelSchema mm = mv.readModel(username, modelName);
      System.out.println("nom de la classe    "+mm.getDeltext());
……..
```

Figure 9: Restful Web Service model invocation

A description of the model in natural language may also help understanding the model behavior. The web service always reads/writes from/to TSC. When an event occurs, the web service detects the event type. Il the event is an internal event, the web service ensure that all parameters are ready to invoke the model *DeltaInt()* transition. It does so for the external and initialization events. The restful web service looks like the driver of the model. *insertModel()*, *deleteModel()*,

getModel() are some Admin methods to insert, delete and load a model from the Store. The *readModelClass()* is used by *getModel()* to load the java class (devsjava) of the model. The *WSDeltaExt()* performs a *DeltaExt()* reflection.

Client Tasks

Figure 10 shows a client loop, depicting the steps followed by a client in a simulation session. A client may simulate an atomic or a coupled models. A coupled model execution is more complex. In the following, we show the 2 executions respectively, commenting figure 10 for each case.

Atomic Model Execution

A client starts by selecting a model (Atomic), no coupling is needed, only the interaction with the client interface (CI). This selection is performed as a natural language search. The CI invokes the Mediator which performs a lookup by starting its Repository which returns the URLs of the web services handling the searched model which may be remote. The Mediator passes this URLs to its Collector which invokes the local/remote web services that extract the requested model RDF graphs (information on model). The Mediator places these graphs (ontology) in the TSC and allows the client to query it (SPARQL). The client can ask for the input/output ports, the owner, a short description, etc. finally, he chooses one from the listed models. The Mediator updates the TSC by deleting the no needed models and adding the full information, figure 8, of the retained one as inserted by the Admin. Since there is no coupling, the client launches its simulation, after initializing the simulation parameters. The Mediator adds these parameters to the TSC and invokes the Restful WS to perform its simulation.

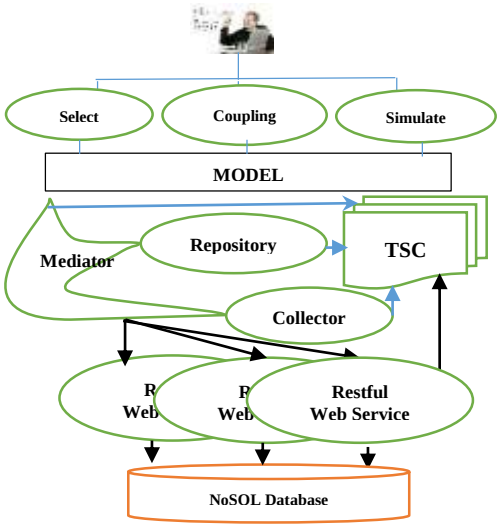

Figure 10: Client Loop

The WS will follow a standard DEVS simulation of atomic model, according to figures 5 and 9, but intelligently. Before requesting any step or any transition function, it looks for its parameters and adds appropriate triples to allow a significant and correct run. For each output, *out()*result, it semantizes its results before inserting in the TSC. The Mediator sends these information to the CI for visualization. This simulation driving is given in figure 5.

Coupled Model Execution

When a coupled model is invoked, this means that each atomic model involved must execute and synchronize itself with respect to the coupling rules.

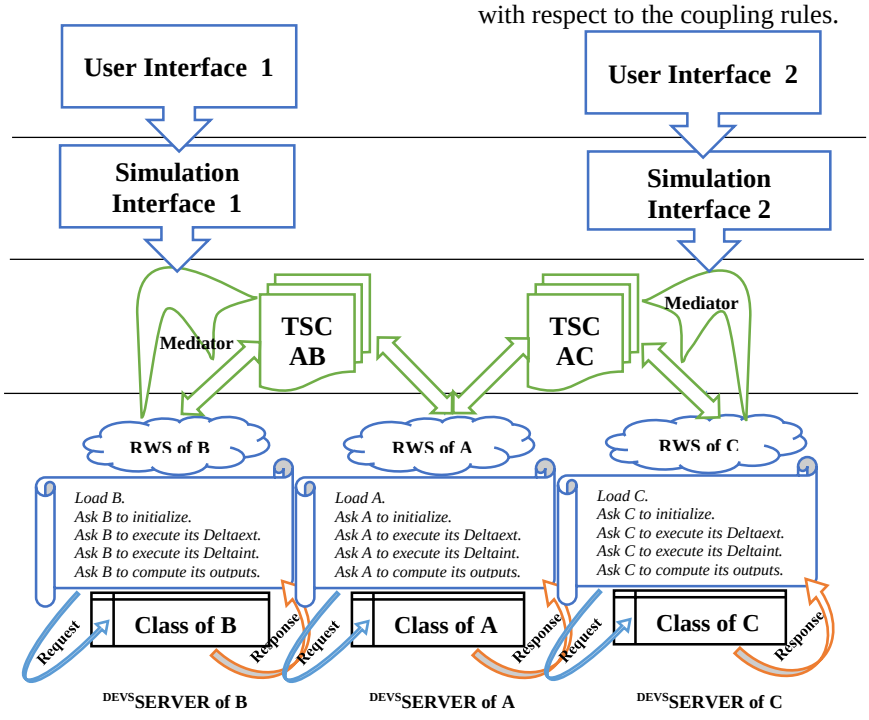

Figure 11: a coupled model execution within two distant Servers

Only Restful Web services supervising the models have to communicate and synchronize and each web service routes the request to its own model.

Figure 11 depicts 2 client sessions. The 1st one runs the AB coupled model and the 2nd the AC one. Each user uses a CI to connect. He calibrates his simulation using the simulation interface (SI) implemented as a Session Bean. Once a simulation started, each Mediator creates a TSC and inserts the simulation parameters. These parameters involve the atomic models, the coupling information and the initialization conditions. Now, WS read/write intelligently from/to TSC until the simulation ends. Periodically, the SI picks and sends to CI intermediate results for visualization.

Coupling rules are RDF triples added by the Mediator once the client finished his simulation configuration. These rules can be obtained intelligently according to the model ontology. The coupling is specified by the *<u:isCouledTo>* predicate as follows:

<u: A> <u: hasInPort> <u: InPort1>.
<u: A> <u: hasOutPort> <u: OutPort1>.
<u: B> <u: hasInPort> <u: InPort2>.
<u: B> <u: hasOutPort> <u: OutPort2>.
<u: AB> <u:hasInPort> <u:InPort3>.
<u: AB> <u: hasOutPort> <u:OutPort3>.
<u:InPort3> <u: isCoupledTo> <u:InPort1>.
<u:OutPort2> <u: isCoupledTo> <u:InPort1>.

Since the Restful WS driving the model has the same functionalities as the driven model, the WS may read semantics of each transition function before invoking it to synchronize itself within the simulation. The synchronization is also designed as triples. Each model has and can compute its T_L and T_N times of its last and next events with respect to T_{NOW} the current simulation time. When any transition function is invoked by the WS, it checks if necessary parameters are already computed. Though, a model B can now know by reading the TSC that a model A must product its output at a specified time and write it in the TSC. If B does not find A output, it must wait until its arriving before running its *DeltaExt()*. This is described by the semantic of the *Out()* of A. According to a disease spread, Epidemic Modeling can predict the next infected contact time which is reported in the TSC.

The triple *<u: A> <u:hasOutPort1NextTime> <u: value>* specifies that A will product its next output on OutPort 1 at time equals *value*. All the history of the events times are recorded in the TSC until the end of the simulation session. OutPorts events next times are also reported according to T_{NOW}. Intelligently, the WS evaluates the transition function parameters before execution.

AMBIENT INTELLIGENCE ADAPTATION

Nowadays Constrained Devices such as Smartphones are furnished with GPS system. Free Android/iOS applications can record contact locations in data stores (NoSQL) using Restful WS. A same way, other health data, temperature, blood pressure, etc. can be captured and recorded. These data are integrated to Real-time Simulation to allow simulation be conducted with online data.

The mathematical modeling of infectious disease spreading has been extensively studied for a long time [12]. A lot of epidemic models, such as the compartmental models that are composed of differential equations, have been developed and analyzed [12,32]. The population is divided into different compartments and each compartment corresponds to an epidemiological state which depends on the characteristics of the particular disease being modeled and its transmission over complex heterogeneous networks where a node is an individual and an edge stands for interaction between nodes allowing disease transmission [19-21,23-24,32-33,35–37]. It is shown that the SIS [3] and the SIR [21] models indicated that the connectivity fluctuations of the network play a major role by strongly enhancing the incidence of infection. To deal with these connectivity fluctuations, AmI is strongly used.

CONCLUSION

A new concept and ideas for distributed simulation are implemented as DevsServer showing that the design DevsServer principles (TSC paradigm, Mediator, Collector, Repository) can achieve interoperability. The TSC interoperability style at the Web level (Restful WS) allows DevsServer to take advantage of new Web-based features or technologies. On the other hand, DevsServer provides better interoperability applying RESTful WS principles among the TSC paradigm.

DevsServer is specially designed to deal with DSS but it can be applied to similar systems. DSS with Epidemic Modeling deal with a large number of contacts moving dynamically and temporally. Each contact is using AmI devices to join/disjoin the distributed structure dynamically at run time. Therefore, DevsServer is designed to adapt to AmI environments and aims to distribute the simulation among different types of nodes in a dynamic way.

Before detailing future directions such as cloud implementation of DevsServer, we aim to consider a full and extended version of this work integrating a full execution of many simulation scenarios under AmI assistance to show the DevsServer ability to a wide range of interaction and pursue its intelligent interoperability aspect.

ACKNOWLEDGMENTS

This work has been supported by a CNEPRU research project entitled BIOSIF II (*Code: B*01820120086*) an extension of BIOSIF I (*Code: B*01820080016*) funded by the AIRLIO Team under the supervision of the LIO labs and the SEMEP services of the Algerian Health Ministry.

REFERENCES

1. Al-zoubi K., Wainer G. (2013). RISE: A general simulation interoperability middleware container. Journal

of Parallel and Distributed Computing, Elsevier. Vol. 73. Issue 5.
2. Amamra L., Mokaddem M., Atmani B., Mesure de la qualité de la vaccination guidée par les données. Sixième Atelier sur les Systèmes D**é**cisionnels ASD'2012, 1 -3 Avril 2012, Université Saad Dahlab, Blida, Algérie. ISBN 978-9947-0-3416-3.
3. Barabási A-L, Albert R. Emergence of scaling in random networks. Science 1999;286:509–12.
4. Barigou F., Mokaddem M., Atmani B., Beldjilali B. 'Towards an Automated System for Extracting Named Entity from Medical Reports' International Congress on Models Optimization and Security of Systems. Du 29-31 May 2010, Tiaret, Algeria.
5. Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. Scientific American, 284, 3443.
6. Brahami M., Atmani B., Mokaddem M., CARTOCEL: un outil de cartographie des connaissances guidée par la machine cellulaire CASI. EGC 2010: 625-626.
7. Fujimoto R. (2000). Parallel and Distribution Simulation Systems, John Wiley & Sons, New York.
8. Gelernter, D. (1985). Generative communication in Linda. ACM Transactions on Programming Languages and Systems (TOPLAS), 7, 80112.
9. Gómez-Goiri A., López-de-Ipiña D. (2012). Assessing data dissemination strategies within triple spaces on the web of things. In: Sixth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), (pp. 763–769).
10. Gómez Goiri A., Orduña P., Diego J., López-de-Ipiña D.. (2014). Otsopack: Lightweight semantic framework for interoperable ambient intelligence applications" In Computers in Human Behavior. vol. 30. p. 460-467. DOI: 10.1016/j.chb.2013.06.022.
11. Guinard, D. (2011). A web of things application architecture integrating the real-world into the web. Ph.D. ETH Zurich.
12. Hethcote HW. The mathematics of infectious diseases. SIAM Rev 2000;42:599–653.
13. Honkola, J., Laine, H., Brown, R., & Tyrkko, O. Smart-M3 information sharing platform. In 2010 IEEE Symposium on Computers and Communications (ISCCs) (pp. 1041–1046). IEEE. 2010.
14. Jafer, S; Liu, Q; Wainer, G, (2012) Synchronization methods in parallel and distributed, DEVS Integrative M&S Symposium, SpringSim Multi-Conference.
15. Khushraj, D., Lassila, O., & Finin, T. (2004). sTuples: Semantic tuple spaces. In The first annual international conference on mobile and ubiquitous systems:Networking and services, 2004. MOBIQUITOUS 2004 (pp. 268–277).
16. Liu J, Zhang T. Epidemic spreading of an SEIRS model in scale-free networks. Commun Nonlinear Sci Numer Simul 2011;16:3375–84.
17. Liu Z, Hu B. Epidemic spreading in community networks. Europhys Lett 2005;72:315–21.
18. Mokaddem M., Bahnes A., Atmani B., 'création dynamique orientée services de contenu pédagogique en e-Learning', JDLIO'2011, Journées Doctorales du Laboratoire d'informatique d'Oran, 2011 , Oran, Algérie
19. Mokeddem S., Atmani B., Mokaddem M. Supervised Feature Selection for Diagnosis of Coronary Artery Disease Based on Genetic Algorithm. First International Conference on Computational Science and Engineering (CSE 2013). Dubai, UAE. May 2013. pp 53-64. ISSN 2231-5403, ISBN 978-1-921987-23-6.
20. Mokeddem S., Atmani B., Mokaddem M. 'An Effective Feature Selection Approach Driven Genetic Algorithm Wrapped Bayes Naïve'.Int. Nat. Journal of Data Analysis Technics and Strategies, (2015).ISSN online:1755-8069 ISSN print: 1755-8050.
21. Moreno Y, Pastor-Satorras R, Vespignani A. Epidemic outbreaks in complex heterogeneous networks. Eur Phys J B 2002;26:521–9.
22. Nixon, L. J., Simperl, E., Krummenacher, R., & Martin-Recuerda, F. (2008). Tuple space-based computing for the semantic web: A survey of the state-ofthe-art. Knowledge Engineering Review, 23, 181212.
23. Olinky R, Stone L. Unexpected epidemic thresholds in heterogeneous networks: the role of disease transmission. Phys Rev E 2004;70:030902(R).
24. Pastor-Satorras R, Vespignani A. Epidemic dynamics in scale-free networks. Phys Rev Lett 2001;86:3200.
25. Pirkkalainen, H., & Pawlowski, J. M. (2012). The knowledge intervention integration process: A process-oriented view to enable global social knowledge management. International Journal of Knowledge Society Research (IJKSR), 3, 45–57
26. Silva M. J., da Silva F. A. B., Lopes L. F., Couto F. M., (2010). Building a Digital Library for Epidemic Modelling, Proceedings of ICDL 2010 – The International Conference on Digital Libraries New Delhi, India, February.
27. Tolk A., (2010). Interoperability and composability, in: C. Banks, J. Sokolowski (Eds.), Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains, Wiley, New Jersey, pp. 373–402.
28. Varela-Candamio L., García-Álvarez M. T. (2012). Analysis of information and communication technologies in higher education: A case study of business degree. International Journal of Engineering Education, 28, 1301–1308.
29. Wainer G., (2009). Discrete-Event Modeling and Simulation: A Practitioner's Approach, CRC press, Taylor & Francis Group, Boca Raton, Florida.
30. Wainer G., Al-Zoubi K., Mittal S., Risco Martín J., Sarjoughian H., Zeigler B. (2010), in: G. Wainer, P. Mosterman (Eds.). Discrete-Event Modeling and

Simulation: Theory and Applications, CRC Press. Taylor and Francis, pp. 389–494 (Chapters 15–18).
31. Wainer G., Madhoun R., Al-Zoubi K. (2008). Distributed simulation of DEVS and Cell-DEVS models in CD++ using Web services, Simulation Modelling Practice and Theory 16 (9) 1266–1292.
32. Xiao Y, Zhou Y, Tang S. Modelling disease spread in dispersal networks at two levels. Math Med Biol 2011;28:227–44.
33. Yang R, Wang B-H, Ren J, Bai W-J, Shi Z-W, Wang W-X, Zhou T. Epidemic spreading on heterogeneous networks with identical infectivity. Phys Lett A 2007;364:189–93
34. Zeigler B., Praehofer H., Kim T., (2000). Theory of Modeling and Simulation, Academic Press, San Diego, CA.
35. Zhang H, Fu X. Spreading of epidemics on scale-free networks with nonlinear infectivity. Nonlinear Anal Theory Methods Appl 2009;70:3273–8.
36. Zhang J-P, Jin Z. The analysis of an epidemic model on networks. Appl Math Comput 2011;217:7053–64.
37. Zhang J-P, Jin Z. Epidemic spreading on complex networks with community structure. Appl Math Comput 2012;219:2829–38.