^{DEVS}Server: ambient intelligence and DEVS modelling-based simulation server for epidemic modelling

Mostefa Mokaddem* and Baghdad Atmani

Laboratoire d'Informatique d'Oran (LIO),
University of Oran 1 Ahmed Ben Bella,
BP 1524 El M'Naouer 31000 Oran, Algeria
Email: mokaddem.mustapha@univ-oran.dz
Email: atmani.baghdad@univ-oran.dz
*Corresponding author

Abdelmalek Boularas

Computer Information System Department,
Ahmed Bin Mohamed Military College,
P.O. Box 22988, Doha, Qatar
Email: boularas@abmmc.edu.qa

Chihab Eddine Mokaddem

Département d'Informatique et de R. Opérationnelle,
Groupe de Recherche GEODES,
Université de Montréal,
C.P. 6128 Succursale Centre-ville Montréal (QC) H3C 3J7, Canada
Email: mokaddec@iro.umontreal.ca

Abstract: To improve disease surveillance systems (DSS) with faster and accurate outbreak detection and epidemics propagation capabilities, the availability of fine-tuned models is required along with the design of server-based solutions that simulate the effects of public health authorities' measures and integrate ambient intelligence (AmI) capabilities to semantise epidemic models. Hosting discrete event system specifications (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. In this paper, we 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 a service oriented architecture (SOA) interface for the TSC operations. This interface complies with DEVS formalism and focuses on simplicity, conviviality and modularity, so that a single or many simulations that support different models can still interact. To assess ^{DEVS}Server, we provide a tuberculosis epidemic model simulation in time-varying temporal network with genetic programming immunisation strategy approach.

Keywords: ambient intelligence; AmI; triple space-based computing; TSC; service oriented simulation; parallel discrete-event simulation; PDES; disease surveillance system; epidemic modelling; temporal network; genetic programming; immunisation strategy.

Reference to this paper should be made as follows: Mokaddem, M., Atmani, B., Boularas, A. and Mokaddem, C.E. (2018) '^{DEVS}Server: ambient intelligence and DEVS modelling-based simulation server for epidemic modelling', *Int. J. Simulation and Process Modelling*, Vol. 13, No. 6, pp.557–581.

Biographical notes: Mostefa Mokaddem is an Assistant Professor at the Computer Science Department and a member of the Lab. of Informatics of Oran, University of Oran 1 Ahmed Ben Bella He received his Magister (2008) in Computer Sciences at the University of Oran, Algeria. His main research interests are DEVS, SOA and SOA simulation, epidemic modelling, AmI, data mining and decision support systems.

Baghdad Atmani is a Full Professor in Computing Science at the University of Oran 1 Ahmed Benbella. His field of interests are data mining and machine learning tools. His research is based on knowledge representation, knowledge-based systems and CBR, data and information integration and modelling, data mining algorithms, expert systems and decision support systems.

Abdelmalek Boularas is a Professor in Computer Information System Department, Ahmed Bin Mohamed Military College, Doha, Qatar. He received his PhD in Computer Science from the Rensselaer Polytechnic Institute, Troy, New York, USA, 1984. His main research field is fuzzy sets, fuzzy logic, computer architecture, transportation systems, simulation and intelligent systems.

Chihab Eddine Mokaddem is a PhD student at the DIRO and member of the GEODES group, Université de Montréal, Canada. He received his Master in Computer Science from the University Ahmed Ibn Badis of Mostaganem (Algeria), in 2015. His research fields are pattern detection and refactoring in model driven engineering, model transformation, search-based software engineering, simulation and machine learning tools related to software engineering and intelligent systems.

This paper is a revised and expanded version of a paper entitled '^{Devs}Server: ambient intelligence and DEVS modeling based simulation server' presented at 2016 Spring Simulation Multi-conference (SpringSim '16'), The Westin Pasadena, California, 3–6 April 2016; MSCIAAS '16 Proceedings of the Modeling and Simulation of Complexity in Intelligent, Adaptive and Autonomous Systems 2016 (MSCIAAS 2016), Pasadena, California, 3–6 April 2016; and Space Simulation for Planetary Space Exploration (SPACE 2016), Pasadena, California, Society for Computer Simulation International San Diego, CA, USA, 3–6 April 2016.

1 Introduction

In recent years, contamination and its interaction with huge flow of quantitative social, demographic and behavioural data are 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.

Demographic characteristics can explain differential health and disease transmission patterns within societies. Indicators of socioeconomic development (economic diversity, income disparity, social class fluidity), health care systems (refined measures of medical care), and public health characteristics are incorporated as health determinants rates. These characteristics, based in the social and economic organisation of societies, are postulated to have significant influences in shaping patterns of health and disease transmission. Community level characteristics are, as well, incorporated into the analysis of demographic and health behaviours. Communities provide a localised context for the social, economic, and political structures important to the interaction between these health and disease transmission determinants. But it is unclear which constellation of community variables has consistent effects. The absence of well-specified models has limited both theory building and the formulation of public health programs.

The service of epidemiology and preventive medicine (SEMEP, i.e., French acronym for Service d'Epidémiologie et de MEdecine Préventive) addresses ways that the environments in which we live shape health. The SEMEP always try to improve health care models and helps finding practical solutions to population health challenges by assessing assets and barriers, identifying evidence-based practices and innovations, and developing practical tools that guide action in promoting health and preventing chronic disease and disease transmission (Brahami et al., 2010; Amamra et al., 2012).

The distribution of health-damaging experiences is not in any sense a 'natural' phenomenon but is the result of a toxic combination of poor social policies, unfair economic arrangements, and bad politics.

With the evolution of technology and data (big data) and exchange standards, the SEMEP now has the opportunity to strengthen and modernise the infrastructure supporting its DSS.

As part of the surveillance strategy, this modernisation is underway to enhance the system's ability to provide more comprehensive, timely, and higher quality data than ever before for public health decision making. Through this initiative, the SEMEP seeks to increase the robustness of its DSS technological infrastructure so that it is based on interoperable, standardised data and exchange mechanisms that support internet of things (IoT) and ambient intelligence (AmI) technologies. AmI aims to enhance the way people interact with their environment to promote safety and to enrich their lives. The achievement of AmI largely depends on the technology deployed (sensors and devices interconnected through networks) as well as on the intelligence of the software used for decision-making (Augusto and McCullagh, 2007). AmI is an emerging discipline that brings intelligence to our everyday environments and makes those environments sensitive to us. AmI research builds upon advances in sensors and sensor networks, pervasive computing, and artificial intelligence (Bravo et al., 2016).

An interesting area concerns the integration with real data from large scale health sensors (AmI constrained devices and smartphones). Actually, only a sample of users will be carrying sensors, the accuracy of the estimation of epidemic propagation-based solely on such a sample needs to be assessed on a real deployment. Due to these previous significant changes, the SEMEP is involved in applying new simulation solutions to deal with these AmI-based Surveillance Strategies. These solutions embed progress in information, computing, and communication systems.

Applying abstraction concept further to the computational thinking model for system design and implementation, which include an architecture-driven approach, model-driven development, and workflow-based design process (Chen, 2018) (Figure 1) is related to this special issue of the International Journal of Simulation and Process Modelling (IJSPM).

Figure 1 Disease simulation in a virtual city (see online version for colours)

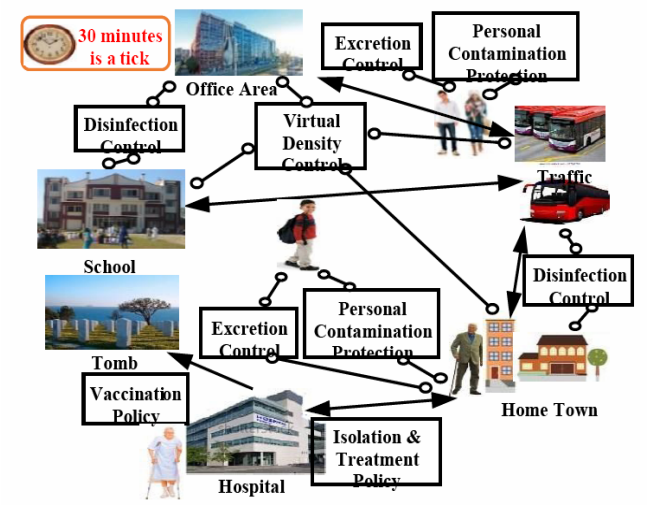


Figure 1 concerns a DSS strategy showing a simulation-based epidemiology with infection process model depending on social structure and filter type protection policy model with immunisation policy. The infection process model is divided into three basic models. The first one is a stage transition model of disease under the physical condition and medical treatment. The second (triangle arrowheads) is a city structure model and human activity model on the virtual city (smart city) that contains several types of social activities of agents. The third (circle arrowheads) is a contamination and infection models that contain protection policies against contamination and infection. Each individual carries/wears sensors during his moves. Human activity model deals with his temporal moves between home, traffic station, office area, school and hospital until tomb to detect if there are any face-to-face contacts patterns. The control of the virtual density is necessary. Many infected persons may be present. The disease stage model illuminates different stages of the process linking life events to disease.

Events are viewed as triggering affective states (susceptible, infected, or recovered) that in turn engender behavioural and biological responses having possible downstream implications for disease. A tick of 30 minutes is enough to transit from an S state to tomb.

Figure 1 also describes an experience of building and deploying a wearable system for automatically sensing, inferring, and logging a variety of physical activity to build high level simulation server platforms that reliably sense real world human actions and to develop intelligent simulations that automatically infer high level human behaviours from low-level sensor data.

The controls of excretion, disinfection, and virtual density are due to AmI platforms where sensors data are sent to service oriented architecture (SOA) servers (AmI technologies) to capture the previous characteristics generating spatial and social network structures influencing infectious disease transmission within populations. The DSS can indicate to people (personal smartphone notifications with alerts levels) to wear some personal contamination protection (maybe masks) before reaching some suspected areas.

Epidemic modelling and computational infrastructures, such as SOA, enable creating very detailed representations since restful web services (RWS) perform intelligently to generate the best accurate semantic-based model. With accurate models, we can predict the outbreak detection, the spread of diseases and simulate the effects of public health authorities' measures.

Parallel discrete-event simulation (PDES) has received increasing interest as simulations become more runtime 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 virtualised infrastructures (Fujimoto, 2000; Mittal, 2006; Mittal et al., 2007, 2009; Al-zoubi and Wainer, 2013; Jafer et al., 2013; Chen, 2018). Like popular PDES environments and their hybrid synchronisation techniques, in SOA oriented simulations (Mittal et al., 2009; Seo, 2009), 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.

Our simulation, integration and Fouille de données (SIF) (data mining) researchers involved with SEMEP in such design issues are developing a specific modelling platform to help model, simulate and evaluate DSS. An important challenge in designing DSS is to define immunisation strategies that discover a meaningful group of individuals (community) that are strongly related to the disease.

Once this community discarded, disease can be eradicated (Liu and Zhang, 2011). AmI-based approaches using semantic temporal network may be applied as novel intelligent and dynamical immunisation strategies. To simulate such strategies, a new AmI-based parallel and distributed discrete event simulation server solution is well-matched allowing the design of semantic atomic and coupled models integrated in epidemic modelling digital libraries (EMDL) (Silva et al., 2010) within simulation servers. EMDL hold models as resource description framework (RDF) graph to semantise modelling. The semantic associated to models describes the how-to of models and help simulation servers to perform intelligently as it is designed in our approach. Thus, providing Modelling and Simulation with semantics is prerequisite.

To pursue this goal, simulation servers use EMDL to manage models while describing them with ontologies. Knowledge may be shared between modellers and servers

themselves. 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 minimise the psychological impact of servers' use. Furthermore, AmI-based servers can be used in the same way in simulation to save modellers from doing low-level but yet time-consuming modelling tasks such as interoperability. Modellers can now focus on modelling with a high aggregate-value, where the importance of the human capital is vital (Pirkkalainen and Pawlowski, 2012; Gómez-Goiri et al., 2014). AmI Abstraction has been applied for representing a complex system in hierarchical layers, with a higher layer hiding certain information from a lower layer (Chen, 2018).

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 (Guinard, 2011), propose a straightforward integration of servers with the web using RWS. The problem with this model is that it couples the communication between nodes. This coupling can be avoided by using indirect communication styles (Gómez-Goiri et al., 2014). Indirect communication can be space and/or time uncoupling. Space uncoupling is achieved when the sender does not need to know the receiver or receivers and vice versa. Time uncoupling happens when senders and receivers do not need to exists in the same time. 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 specialised system converts and reinterprets them. A way to solve this problem is annotating the message semantically as proposed by the World Wide Web (WWW) (Berners-Lee et al., 2001; Pirkkalainen and Pawlowski, 2012).

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.

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

b it runs in different computational platforms, allowing Java SE, Java Mobile and Android interaction.

The underlying interface is based on SOA (Mokaddem et al., 2011) and covers isolated features such as discovery, maintenance or data access. Different simulations can provide only certain features and still interact with each other. In this way, it is possible to embed it in other real-time simulations.

Regarding the immunisation strategy simulation, we do not provide any specification; it is out of the scope of this paper which is only concerned with the design of ^{DEVS}Server over AmI interoperability.

The rest of the paper is organised as follows. Section 2 outlines related work. Section 3 describes the conceptual model for an SOA-based TSC solution, let us say ^{DEVS}Server principles. Section 4 details the implementation made to adapt it to the necessities of AmI requirements. Section 5 presents an epidemic modelling specification of the tuberculosis (TB) disease under ^{DEVS}Server involving temporal network and genetic programming (GP) capabilities. Since we are trying to target both epidemic modelling communities and general-purpose simulation communities, this example lets epidemic modelling readers be involved in SOA and AmI-based simulation servers. Therefore, the TB as temporal network, the propagation phase, and the immunisation phase need to be more detailed to fit their concerns and to envision what is the contribution of new simulation solutions in this topic. We hope that this initiative will make ^{DEVS}Server more attractive and getting more support. Discovering disease propagation and immunisation strategies may show the lack of related semantic-based simulation support tools and will certainly motivate simulation communities to investigate consequently. Finally, Section 6 concludes and discusses future work.

2 Related work

In the following two subsections, we analyse both semantic solutions of interoperability involving mobile and embedded devices and concluding with TSC paradigm, and the SOA-based modelling/simulation as synthesised by Al-zoubi and Wainer (2013). We compare our solution with the rest emphasising their strengths and weaknesses.

2.1 Ambient intelligence interoperability

Regarding representational state transfer (REST), its use in resource constrained devices is a current trend defended by the WoT initiative (Guinard, 2011). 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 (Gelernter, 1985). TS works with semi structured data, which is accessed in an associative manner. Several

TS solutions have used semantics to enhance the shared data (Khushraj et al., 2004). sTuples was conceived for scenarios (Nixon et al., 2008). In sTuples, the clients access a centralised space through a communication gateway. The centralisation completely simplifies the solution, but makes the whole system dependent on a single machine. Besides, Otsopack (Gómez-Goiri et al., 2014), a lightweight semantic framework for interoperable ambient intelligence applications, avoids the need of gateways by requiring a prominent protocol (i.e., HTTP) for the communication between the nodes. TripCom (IST-4-027324-STP, http://www.tripcom.org), a triple space communication, 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. In doing so, they can access to the most updated data and manage their own data. Finally, Smart-M3 (Honkola et al., 2010), an information sharing platform, 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 space access protocol (SSAP) (Honkola et al., 2010), a Smart-M3 fully integrated protocol, is used for the communication between both types of nodes. The SSAP can be implemented on top of different communication mechanisms. 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 centralised and 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.

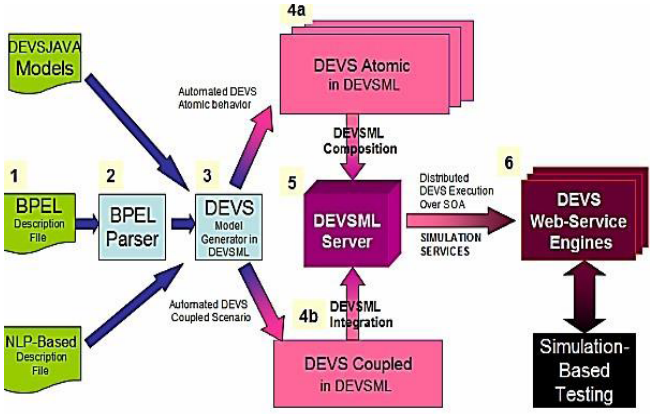
As was previously stated, our API is based on the TSC paradigm. TSC is a TS variation where the information is stored in RDF. Three key concepts (space, triples, graphs) are important at this point: models share information in a common *space*. A space is identified by a uniform resource identifier (URI). 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. Gómez-Goiri et al. (2014) present further discussion about knowledge distribution strategies.

2.2 Modelling and simulation interoperability

Mittal et al. (2007) present a test and development environment using discrete event system specification modelling language (DEVSML) and the SOA framework. DEVSML is built on XML and provides model interoperability among DEVS models hosted at remote network addresses. The Client application that communicates with multiple servers hosting DEVS simulation services and the underlying SOA for DEVS (SOADEVS) framework (Mittal et al., 2009). The authors show how SOADEVS is positioned to address the need for a Department of Defence Architecture Framework, DoDAF-based net-centric paradigm (Mittal, 2006) for test and evaluation at the system-of-systems and enterprise systems levels. The SOADEVS framework provides the needed feature of runtime composability of coupled systems using the SOA framework. The integration of DEVSML and SOADEVS is performed with the layout as shown in Figure 2. The manner in which DEVSJAVA/DEVS-Suite (ACIMS software site) models could be attained or developed by client can be manifold. The models can be created through natural language processing (NLP) methods, raw java format, or BPMN/BPEL files. The models rest with the client (Step 3, Figure 2). Once the client has DEVSJAVA models, DEVSML server can be used to integrate the client's model with model available at some other place on the web to get an enhanced integrated DEVSML file that can reproduce DEVSJAVA model in java format (Steps 4 and 5). The SOADEVS enabled server can either take this integrated DEVSML file directly or can ask the user to provide the top level coupled model through the SOADEVS client application. Finally, the remote simulation is conducted at various DEVS simulation engines located over the web (Step 6) and be used for simulation-based testing in a distributed environment.

Figure 2 DEVSML and SOADEVS integrated (see online version for colours)



However, Mittal et al. (2007) provide only platform interoperability that employs JAVA serialisation which converts JAVA objects into byte array to send messages to simulators. This restricts interoperation to simulators based on JAVA. To add the language interoperability to the platform interoperability, (Seo and Zeigler, 2009) apply neutral message passing and the SOA environment. Their interoperability on DEVS uses simulator level interoperability that uses common simulator interfaces to simulate DEVS models. The simulator interface describes a minimum agreement being able to implement a simulator class using different languages such as JAVA, C++, and C#. Their approach strengthens model reusability because DEVS modelling and simulation separates models and simulators. To increase model composability, they apply a new construct called the DEVS namespace which is a specific XML namespace to define unique message types used at DEVS models in the DEVS simulator services. Integrating different DEVS simulators provides semantic interoperability. The main contribution of their work was to design and implement interoperable DEVS simulation environment using SOA and DEVS namespace. The interoperable DEVS simulation environment is categorised to the design of DEVS simulator service and DEVS simulator service integrator. The DEVS simulator service provides not only simulator level interoperability, but also model level interoperability. Also, through the DEVS namespace, they can couple DEVS simulator services with same message types. In an interoperable DEVS environment, web services represent DEVS simulators embedding specific DEVS models. They have minimum agreement for simulator and information of input/output ports which have specific data types described in DEVS namespace (Seo, 2009).

In the Frontier design environment (Seo et al., 2012), authors exploit the reconfigurability inherent in a SOA with orchestration of the offered services to provide much greater flexibility. Such flexibility allows users to flow through the processing steps at will bypassing intervening steps if appropriate. To do so, they use the System Entity Structure which is a high-level ontology framework targeted to modelling, simulation, systems design and data engineering. SESs are pruned to pruned entity structures (PES) to be transformed to executable simulation models. A constrained natural language approach to pruning has been developed that not only allows easy manual pruning but also enables automated specification through input pruning scripts. This capability provides a key element in achieving the flexibility to adapt to user requirements. The family of PESs generated from a single pruning script constitutes the solution space. These PESs are encoded in XML, and with the help of an XMLToOWL converter, stored in a common data service. In this form, they are available on demand to Frontier SOA services where all information exchanges between services are mediated by a common data service within a web services environment that supports a 'semantic bus'. However, a more flexible orchestration is required to implement automatic invocation of modelling services. Eventually an intelligent learning system can implement such flexibility. However, initially Frontier must be seeded with some criteria for selecting services and invoking them in a particular order, with outputs from some services passed as input to others. Because Frontier is a semantics-based system, an OWL-based representation is most appropriate. This representation will reside in the common data service and implemented in TripleStore, thus providing support for orchestrating services. To level up in flexibility from a static OWL representation, intelligence is provided to perform simple matching between the declared capabilities of the basic Frontier services and the declared needs (including values) of the user. Eventually, an intelligent learning system can make better matches. The easiest and most simple-minded form of matching will look for explicit matches, i.e., equal values, between corresponding attributes (OWL properties) of the frontier services and the user requiring the services. Such matching will, at first, be crude, but it will be more than syntactic signature matching because the properties represent semantic information about both the demand and supply side. More advanced matching will involve dynamic orchestration decisions, such as examining the results from one service and inferring – at that point – whether they should be passed to another service, or whether perhaps the previous inputs should be passed to another service, etc. Whether such learned rules and/or choices can themselves be represented in OWL or must be kept in a sub-symbolic neural representation is not yet clear; but in either case, Frontier will provide the ability to adapt its orchestration of service's needs (Seo, 2012).

Zeigler et al. (2017) show that DEVS offers the ability, via mathematical transformations called system morphisms, to map a system expressed in a formalism suitable for analysis (e.g., timed automata or hybrid automata) into the DEVS formalism for the purpose of simulation. They discuss a probabilistic extension of the Finite Deterministic DEVS (FD-DEVS) (Mittal et al., 2007; Zeigler and Sarjoughian, 2012) formalism that enables a set of model classes and tools derived from Markov-type models. The MS4 (MS4 Me, 2013; Seo et al., 2013) modelling environment provides a suite of tools that support this extension, called finite probabilistic DEVS (FP-DEVS). This extension, also, is based on PROMELA models that are expressed in a high-level abstraction which does not consider explicit representation of time or events. But, the efficiency of the processes of verification and validation relies on the accuracy of the models. Yacoub et al. (2017) worked to develop a new extension of PROMELA for the modelling of discrete-event systems so that the verification of these models is then done by combining formal verification and simulation-based verification.

The implementation of Frontier Simulation Services is based on the author's extension of the open source ADEVS environment to support simulation using Web Services. The resulting environment, called ADEVS/SOA, allows ADEVS coupled models to be executed on an open-source Tomcat-based SOA platform. Due to the differences in the

underlying languages, C++ and Java, ADEVS/SOA (C++) does not support dynamic instantiation of ADEVS models, ADEVS/SOA (C++) does not support reflection functions for Object classes' variables, ADEVS/SOA does not support dynamic creation of XML DEVS messages from ADEVS messages, and ADEVS/SOA does not create ADEVS Simulator Services with uploaded ADEVS models. These limitations imply that work must be done in individually tailored, rather than automated, fashion to integrate an ADEVS model to execute on ADEVS/SOA. In particular, simulation servers must be individually provisioned with simulator services with pre-assigned atomic models. In contrast in the DEVS/SOA environment based on DEVSJAVA, atomic models can be downloaded to generic simulation servers and locally compiled. Much remains to be done including design and implementation of the semantics-based orchestration and an automated approach to mappings of the SES to incorporated abstractions. Authors also need to go beyond to the current pairing of models to populating the environment with services and models to address a full range of applications objectives. Their ontologies-based approach is under-development consistent with the development of the semantics-based data store discussed above (Seo, 2012).

Al-zoubi and Wainer (2013) present a better simulation interoperability concept background and describes their restful interoperability simulation environment (RISE) middleware that fits within this concept. Their objectives were to enhance interoperability by decoupling/hiding implementations. They highlighted some guidelines to be followed to a general Web-based middleware container. Interoperability, as they stated, enables two or more different software systems to interface and use each service correctly (Tolk, 2010). The complexity of interoperability arises when systems are heterogeneous, as in the case of some distributed simulations. 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 realised in software design and implementation, interoperability needs to be studied from the software perspective, in particular, at the API level (since this is how systems access and use other systems services).

Al-zoubi and Wainer (2013) 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 organised in the following layers: *middleware*, *simulation*, and *modelling*. 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 *modelling* layer operates above the simulation layer. This represents the system under study, which is simulated by a specific simulation environment. These RISE model layers match other existing interoperability conceptual layers, particularly the level of conceptual interoperability model (LCIM) (Tolk, 2010). The LCIM interoperability layer deals with the software implementation of interoperation, including simulation and middleware. In RISE, this layer is presented in two layers: middleware and simulation. The simulation layer contains the simulation engine implementation including simulation algorithms and formalism. This makes the simulation implementation and algorithms independent of the middleware layer. Thus, the simulation layer can have different types of simulation implementations and algorithms. The middleware layer provides management and common interoperability services to the simulation services in the upper layer. Al-zoubi and Wainer (2013) argued that SOAP-based WS simulations group the services as procedures in WS ports (addressed by a single URI). Thus, simulation data is exchanged and described in the form of procedure parameters while the data channels are described as procedures. SOAP messages in XML (describing RPCs) are not exchanged at the simulation level, but at the Web service technology layer. This is the case for all SOAP-based WS simulations such as (Seo, 2009). They however, realised that it was not possible due to the restrictions imposed by SOAP WS structural rules. For example, they cannot decouple interoperated systems implementations if the data channels (procedures) and the way data is described (programming parameters) are part of implementation itself. Composition scalability is complex if every service (implemented as procedure) at the simulations at the server side require stubs on each simulation client. Thus, this interoperability approach is difficult to achieve in open communities as in the case of the Web. This is because in such communities practice, systems need to be designed, implemented, and evolved independently. On the other hand, the SOAP-based WS ports (along with their procedures) had to be created and compiled before even starting up the system. This approach usually ends in a close community where software developers can discuss with each other to resolve systems API related design issues. The web services description language (WSDL) role is to describe the RPCs signatures (i.e., names and input/output parameters). However, once the published WSDL document is compiled to programming stubs (usually by a tool), programmers need to code those stubs and compile them with their software.

Finally, they presented two tables summarising the comparison of RISE to current interoperability approaches (Table 1) and comparing current Web-based simulation approaches (Table 2). Many references related to each case are mentioned.

In RISE, all functionalities are hidden in resources, named with URIs. Those resources (URIs) are connected to each other via uniform virtual channels in which the simulation synchronisation is done using XML messages. Thus, the RESTful interoperability approach allows system designers, to decompose the systems in components (i.e., called resources/URIs), and to hide the implementation within those components, hence separating component

interfaces from software implementation. These fundamentals were adapted by the RWS style, which was adapted by the 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 standardisation of DEVS (Zeigler et al., 2000; Wainer et al., 2008, 2010; Wainer, 2009). This standardisation effort is aiming at interoperating various DEVS-based implementations systems via the web (Wainer et al., 2010).

RISE is not a real SOA server-based solution since 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 develops updates and stores models in digital libraries. In RISE, the semantics of all synchronisation messages is described in XML.

On the other hand, how these messages are handled during implementation is out of the protocol scope (i.e., one does not need to be a programmer to design the synchronisation algorithms). Therefore, semantics is required to handle interoperability and synchronisation during implementation. RWS must be implemented using semantics to handle models.

Similarly, epidemiologists can develop epidemic models for a wide variety of diseases in different formalisms and store these models in digital libraries. These models are ready to be used by communities, users have just to integrate them through RWS in their own work. Therefore, we have to develop models in a way that they can be invoked by a RWS using java reflection. This RWS 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. RWS is an abstraction designed to call DEVS class methods using java reflection Figure 3 shows a simple RWS algorithm. The RWS 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 RWS asks the model for each method parameters. It must obtain parameters values from the TSC. Thus, these parameters must be annotated semantically. Obtaining parameters, the RWS 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 semantised, designed as graphs. Each RWS can easily find all graphs (messages) destined to it. So, a RWS accesses the TSC, looks for its input messages, performs its transitions and generates semantically outputs (graphs) that it writes back to the TSC. No direct communication between models at the low level is used; communication is only through the TSC and between RWS encapsulating models.

Figure 3 Algorithm of RWS model driving

1. Load the model from the Store (Java Class).
2. Obtain model parameters for each of its methods.
3. Choose Initialization parameters.
4. Read semantically the Initialization parameters from the shared space (TSC).
5. Ask the model to initialize itself by passing parameters to it.
6. When event occurs choose transition specific parameters and do:
   1. If external event, then
      1. Read semantically DeltaExt parameters values from TSC
      2. Pass values to the model class
      3. Ask the model to execute its DeltaExt function
   2. If internal event, then
      1. Read semantically DeltaInt parameters values from TSC
      2. Pass values to the model class
      3. Ask the model to execute its DeltaInt function.
      4. Ask the model to compute its outputs
7. Write semantically outputs to the TSC

3 ^{DEVS}SERVER principles

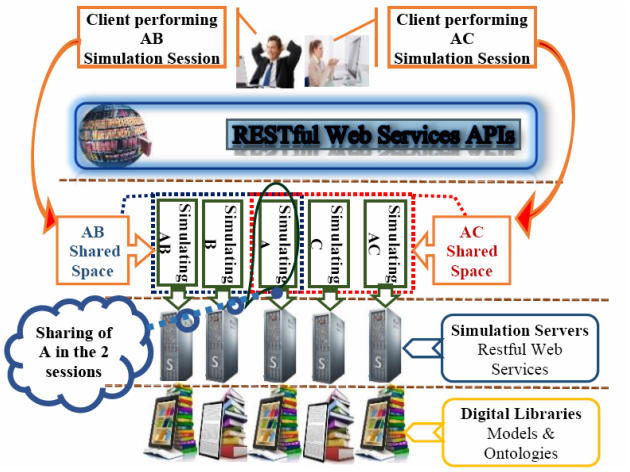
To deal with ^{DEVS}Server principles, let us consider two coupled epidemic models AB and AC with atomic models A, B and C according to DEVS formalism, Figure 4. Each of A, via RWS. A is participating in two parallel simulation sessions at the same time. AB and AC simulations are sessions performed on separate servers by two different clients (Figure 4). Servers store models in respective EMDL. Each epidemic model consumes information from its own EMDL (Silva et al., 2010). Thus, A, B and C are a data mining-based model where a mining of some data is applied to generate output (Barigou et al., 2010; Mokeddem et al, 2013, 2015). Data are in the same EMDL as the model using them. A and B are described with ontologies to allow a modeller to link servers hosting these models while trying to use each of them as sub-model of its own coupled model. Many modellers may realise a coupled model conjointly. AB, A and B and AC, A and C, respectively, represent exchanged information as a standard ontology such as friend of a friend (FOAF) to link models. This ontology is designed as a shared space between these models.

In the example of 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 modeller's server Triple Spaces. Since Triple Spaces is used as shared space, this knowledge is available for other servers using that shared space.

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 these other models (Varela-Candamio and García-Álvarez, 2012). Prior to information retrieving, the coordinator

semantises the information according to the FOAF ontology.

Figure 4 Simulation sessions with model sharing example (see online version for colours)



Finally, it periodically looks into the space to check who is friend of this model. Data mining-based model uses preview data as follow: data are organised in a view from dataset inside digital libraries. Each row of this view is an entity. Many columns are properties of this entity. We apply knowledge discovery from data (KDD) on these data to extract rules that will be used by the model (Barigou et al., 2010; Mokeddem et al., 2013, 2015). Therefore, a modeller 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.

A node, in DEVSServer architecture (Figure 5), is composed of a repository, a Collector and a Mediator. The Repository stores epidemic models, their datasets and ontologies to characterise 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 retrieves diseases models and their information from publicly available DEVSServer servers.

Interfaces enable the admin of the server and the client to perform Modelling 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 (coordinators) and RWS each one handling specific models. Many nodes interact to perform a distributed simulation execution.

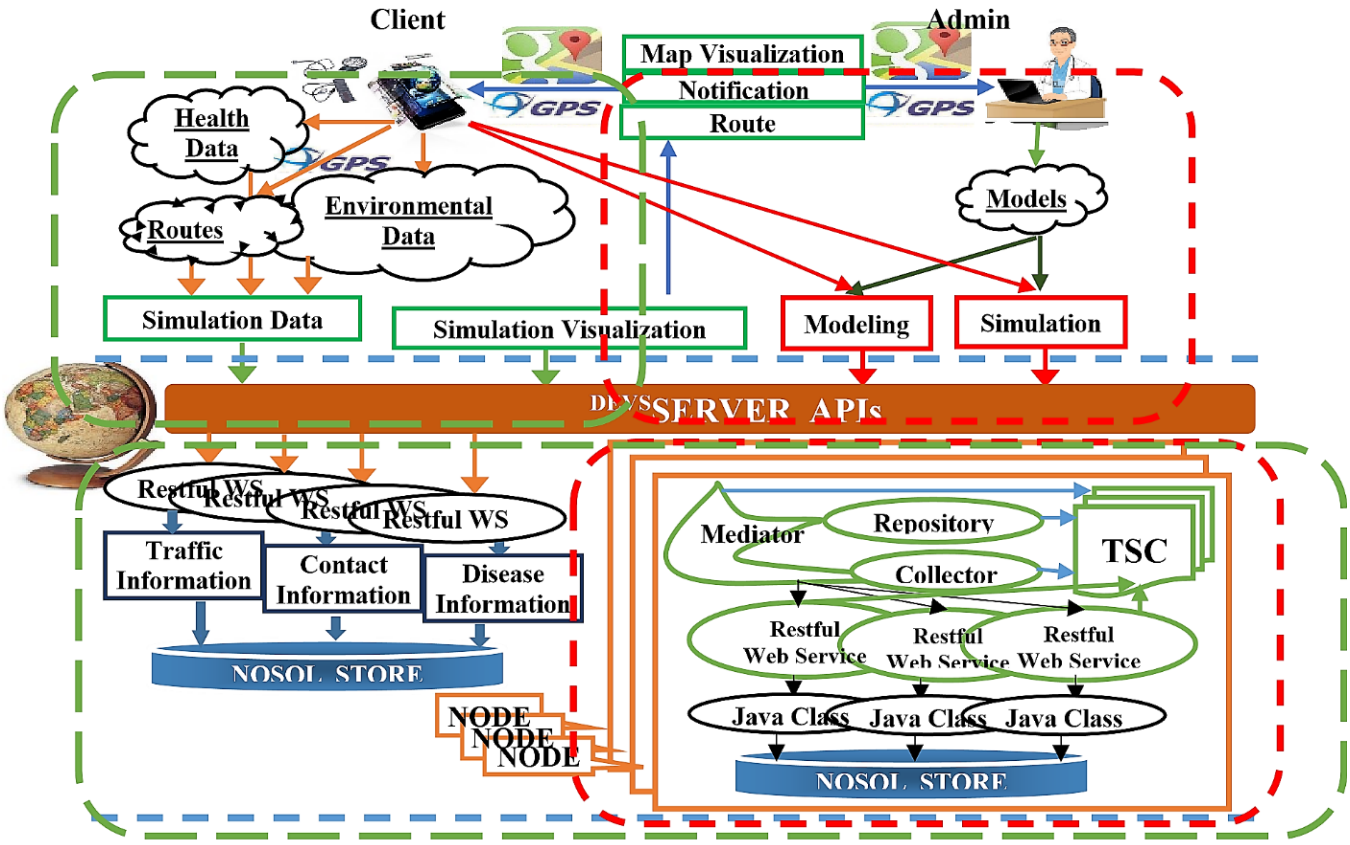
In this section, we detail how models are hosted by DEVSServer and how TSC is used to perform an SOA simulation. Models read inputs and write outputs to the TSC parameterised by the client (DEVS modeller). Two subsections are detailed. The 1st one concerns the admin of the server and the 2nd one the 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 two tasks in detail involving the mediator, the repository and the collector. This DEVSServer workflow and visual approaches, in addition to service-oriented architecture and web services fall down in a computational thinking when developing complex systems as described in Chen (2018).

3.1 The admin tasks

Figure 5, inside the two red squares, 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 utilised. A model is described as an RDF graph to be easily added to the NoSQL store. Such graph can be also converted to ontology. Figure 6 shows some RDF triples depicting the graph of a DEVS TB (tuberculosis). TB is the RDF subject, arrows define the RDF predicates, and helicoids are objects.

hasType, *hasOwner*, *hasInPort*, *hasCreationDate*, *hasJavaClass*, *hasInitialise*, *hasDeltaExt*, etc. are some TB predicates. *AtomicModel*, *Mokaddem*, 12/02/2016, parameter, etc. are some objects that may be used as new subjects to build a graph. Each model owns its functions such as *initialise*(), *deltaExt*(), *deltaInt*(), *out*() and the *conf*(). The *initialise*() function lets the model perform its initialisation. Such initialisation 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.

Figure 5 [DEVS]Server architecture (see online version for colours)



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 <u:hasJavaClass> refers to the java code that will be loaded by the RWS by java reflection. That means when calling a RWS handling any model to initialise itself then the RWS just does a java reflection to call at runtime the *initialise*() method of this java class included in this code. This class is inserted as a large object binary (LOB) in the NoSQL store. So, the related RWS must first load this class from the store then use it by reflection (Appendix A). A description of the model in natural language may also help understanding the model behaviour. The RWS always reads/writes from/to TSC.

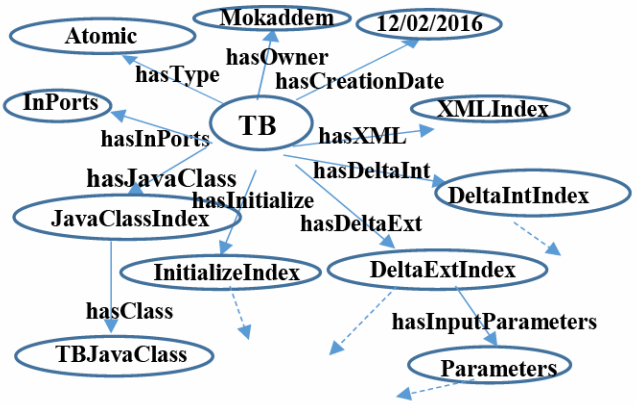
When an event occurs, the RWS detects the event type. If the event is an internal event, the RWS ensures that all parameters are ready to invoke the model *deltaInt*() transition. It does so for the external and initialisation events. The RWS looks like the driver of the model. *insertModel*(), *deleteModel*(), *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 and so on.

3.2 The client tasks

Figure 5, within the two green squares also, shows a client loop, depicting the steps followed by a client in a simulation session. A client may simulate an atomic or a coupled model. A coupled model execution is more complex.

In the following, we show the two executions respectively, commenting Figure 5 for each case.

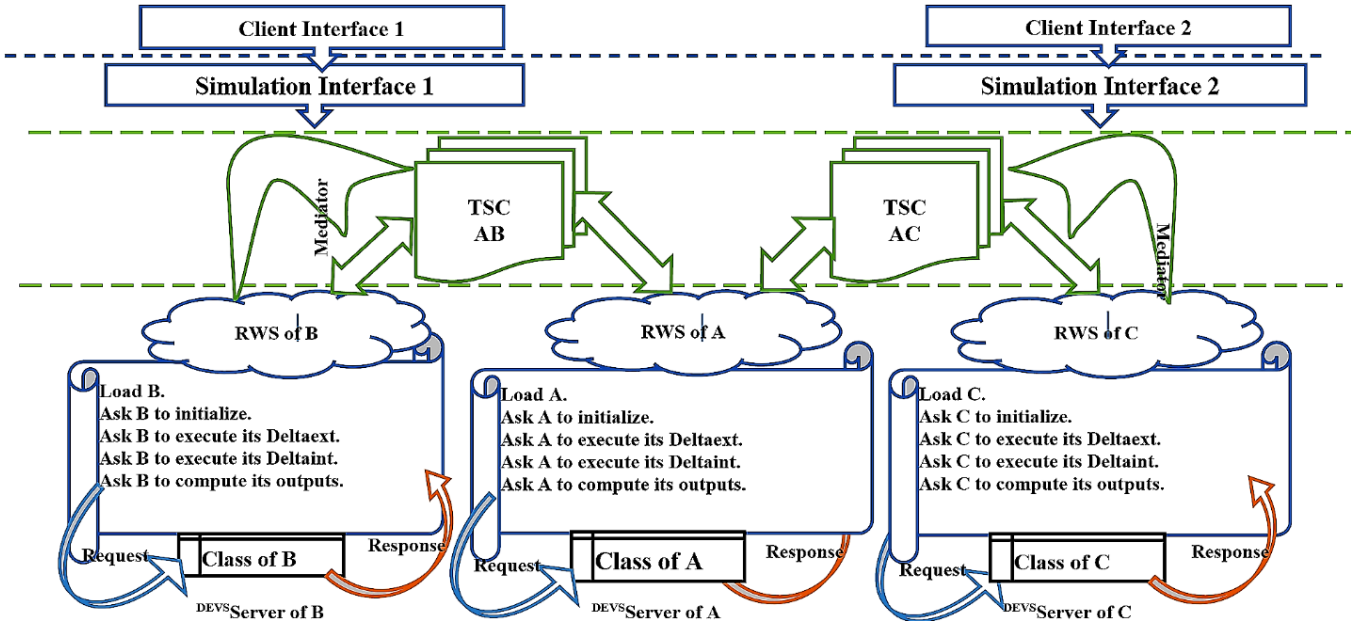
Figure 6 Graph model of a DEVS TB example (see online version for colours)



3.2.1 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.

Figure 7 A coupled model execution within three distant servers (see online version for colours)



The CI invokes the mediator which performs the search by starting its repository. The repository returns the URLs of the RWS handling the requested model which may be remote. The mediator passes these URLs to its collector which invokes the local/remote RWS to extract the requested model RDF graphs. The mediator places these graphs 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 unneeded models and adding the full information of the selected one as inserted by the admin (Figure 3). Since there is no coupling, the client can initialise parameters and start simulation. The mediator adds these parameters to the TSC and invokes the RWS to perform its simulation.

The RWS will follow a standard DEVS simulation of atomic model, according to Figures 3 and 7, but intelligently since it reads/writes from/to TSC and checks transition functions parameters.

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 semantises its results before inserting in the TSC. The Mediator sends this information to the CI for visualisation. This simulation driving is given in Figure 3.

The contents available on a node (DEVSServer) are completely browsable. The list of spaces a node is joined to are available under /diseases (e.g., Epidemic Modelling case). Each space is identified by an URI (e.g., http://diseases/diseaseName). All the resources of that space are listed under http://diseases/diseaseName/{space_uri}. For example, the graph related to the TB deltaExt() is available on http://diseases/TB/DeltaExt_TB, and its TSC is available at http://diseases/TB/TSC. After loading the TB in its TSC, *deltaExt*() becomes available at http://diseases/TB/TSC/DeltaExt_TB. If we make an HTTP DELETE to that resource, we will be taking that graph from the space (TSC respectively). Let sp denotes a space URI (http://diseases), g a graph URI, s, p and o-uri subject, predicate and object, then some HTTP requests are as follow:

GET {sp}/graphs/{?g} returns the graph {g}.

GET {sp}/graphs/{?g}/{s}/{p}/{o-uri} returns the graph of all triples such that the subject is {s}, the predicate is {p} and the object is {o-uri}.

GET {sp}/graphs/{?g}/{s}/{p}/{o-type}/{o-val} returns the graph of all triples such that the subject is {s}, the predicate is {p} and the object type is {o-val}.

DELETE {sp}/graphs/{g} returns the deleted graph {g}.

DELETE {sp}/graphs/{g}/{s}/{p}/{o-uri} deletes all triples from {g}such that the subject is {s}, the predicate is {p} and the object is {o-uri} then returns the deleted triples as graph.

DELETE {sp}/graphs/{g}/{s}/{p}/{o-type}/{o-val} same as previous but with object type.

PUT {sp}/graphs/{g}/{s}/{p}/{o-uri} adds the triple (s, p, o-uri) to the graph g.

PUT {sp}/graphs/{g}/{g'} adds the sub-graph g' to the graph g.

TSC provides some primitives to access to the semantic information hold in each graph as follows:

writeToTSC (space_URI, graph_URI) : URI

writeToTSC (space_URI, triple) : URI

readFromTSC (space_URI, graph_URI) : graph

readFromTSC(space_URI, query_parameters) : graph

takeFromTSC (space_URI, graph_URI) : graph

takeFromTSC (space_URI, query_parameters) : graph

The *writeToTSC*() primitive allows writing a graph/triple into a given graph (identified by its URI). It uses the previous HTTP PUT requests. It returns the URI of the updated graph. The *readFromTSC*() returns a graph

belonging to a given space which contains at least a triple matching the query parameters or has the given URI as its identifier. It should be remarked that it has been designed as a non-blocking operation. It uses the previous HTTP GET requests. The *takeFromTSC*() primitive behaves like a destructive read, deleting the graph returned from the space. It uses the previous HTTP DELETE requests.

Since all the resources of that space are listed under http://sp/modelName/{space_uri}, we can discover the http://sp/modelName/DeltaExt graph that contains available and semantised data associated to the *deltaExt*() transition. If the TSC is containing the following information for 2 models A and B:

<A> <hasDeltaExt> <sp:A/DeltaExt>
<B> <hasDeltaExt> <sp:B/DeltaExt>
<sp:A/DeltaExt><hasInputParameter><sp:A/DeltaExt/inport1>
<sp:A/DeltaExt><hasInputParameter><sp:A/DeltaExt/inport2>
<sp:A/DeltaExt><hasOutputParameter><sp:A/DeltaExt/outport1>
<sp:A/DeltaExt><hasOutputParameter><sp:A/DeltaExt/outport2>
<sp:A/DeltaExt/inport1><hasType> <u:message>
<sp:A/DeltaExt/inport2><hasType> <u:int>
<sp:A/DeltaExt/outport1><hasType> <u:double>
<sp:A/DeltaExt/outport2><hasType> <u:float>
<sp:A/DeltaExt/inport1><hasValue> value_of_parameter (data)
<sp:A/DeltaExt/inport2><hasValue> value_of_parameter (data)
<sp:A/DeltaExt/outport1><hasValue> value_of_parameter (data)
<sp:A/DeltaExt/outport2><hasValue> value_of_parameter (data)

<sp:B/DeltaExt><hasInputParameter><sp:B/DeltaExt/inport1>
<sp:B/DeltaExt><hasOutputParameter><sp:B/DeltaExt/outport1>
<sp:B/DeltaExt/inport1><hasType> <u:message>
<sp:B/DeltaExt/outport1><hasType> <u:double>
<sp:B/DeltaExt/inport1><hasValue> value_of_parameter (data)
<sp:B/DeltaExt/outport1><hasValue> value_of_parameter (data)

<A> <hasEvents> <sp:A/Events>
<B> <hasEvents> <sp:B/Events>
<sp:A/Events><hasEvent> <sp:A/Events/event1>
<sp:A/Events><hasEvent> <sp:A/Events/event2>
<sp:A/Events/event1><hasType> done
<sp:A/Events/event1><hasTime><t:time1>
<sp:A/Events/event1><hasPort> none
<t:time1><hasValue> 08:10:10:20
<sp:A/Events/event2><hasType> ext
<sp:A/Events/event2><hasTime> <t:time2>
<sp:A/Events/event2><hasPort><sp:A/DeltaExt/inport2>
<t:time2><hasValue> 08:10:10:40

To handle an external event, the RWS previously calls its *WSDeltaExt*() method which calls the *readFromTSC(sp:A, sp:A/DeltaExt)* to get the sub-graph sp:A/DeltaExt which contains related parameters. Now, the RWS knows that A has two input parameters and two other output parameters. Before invoking the java class *deltaExt*() method, the RWS get respectively the parameters values using:

readFromTSC(sp:A/DeltaExt/import1,<sp:A/DeltaExt/import1> <hasValue> {?o})

for the 1st parameter and so on. Parameters passing is now ready.

To compute its next event, the RWS invokes:

readFromTSC(sp:A/Events,{?s}<hasTime>{?t},{?t}<hasValue>{?o})

which returns all the remaining events times and the RWS takes the minimum. This event is event_t_N. When the simulation time is equal to t_N (next event time), the RWS calls *takeFromTSC*(*sp:A/Events, sp:A/Events/event_*t_N) to remove event_t_N from the graph sp:A/Events and calls *delta!ext*() to consume it. RWS do so for each occurring event.

Space management requires some additional primitives to join or leave a space using *joinSpace*(*space_URI*) or *leaveSpace*(*space_URI*). The query *query*(*space_URI, query_pameters*) aims to see the space as a whole triplestore, returning all the triples matching the given request.

The Mediator which may play the role of the coordinator of some coupled model begins by creating a new TSC using *createTSC*(). Then, the Mediator finds the user defined model and add its graph to the TSC graph. This is done with *query*(*space_URI*, *graph_of_model*) and *writeToTSC*(*graph_TSC*, *graph_of_model*). The mediator asks the model to join the TSC using *joinSpace*(*graph_TSC*). Now the RWS performs its simulation according the parameters initialisation provided by the user through its CI.

3.2.2 Coupled model execution

When a coupled model is invoked, this means that each atomic model involved must execute and synchronise itself with respect to the coupling rules. Only RWS supervising the models have to communicate and synchronise and each RWS routes the request to its own model. Figure 7 depicts two 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 starts, each Mediator creates a TSC and inserts the simulation parameters. These parameters involve the atomic models, the coupling information and the initialisation conditions. Now, RWS can join and read/write from/to TSC until the simulation ends. According to TSC events, the SI picks and sends to CI intermediate results for visualisation. Coupling rules are RDF triples added by the Mediator once the client finished his simulation configuration. These rules can be obtained according to the model ontology.

Figure 8 is a very simple example of coupling between AB, A, and B where in1 of AB is coupled to in2 of A, out1 of A is coupled to in3 of B, and out2 of B is coupled to out3

of AB. This is specified by the *<u:isCoupledTo>* predicate as follows:

<u: A> <u: hasInPort> <u: in2>.

<u: A> <u: hasOutPort> <u: out2>.

<u: B> <u: hasInPort> <u: in3>.

<u: B> <u: hasOutPort> <u: out2>.

<u: AB> <u:hasInPort> <u:in1>.

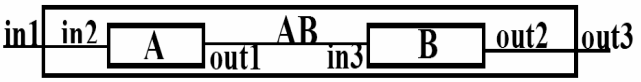
<u: AB> <u: hasOutPort> <u:out3>.

<u:in1> <u: isCoupledTo> <u:in2>.

<u:out1> <u: isCoupledTo> <u:in3>.

<u:out2> <u: isCoupledTo> <u:out3>.

Figure 8 A coupling examples



Since the RWS driving the model has the same functionalities as the driven model, the RWS may read semantics of each transition function before invoking it to synchronise itself within the simulation. The synchronisation is also designed as triples. Each model has and can compute its TL and TN times of its last and next events with respect to TNOW the current simulation time. When any transition function is invoked, the RWS 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 Modelling 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 produce its next output on OutPort1 at time equals *value*. The log of the events times is recorded in the TSC until the end of the simulation session. The next times of OutPorts events are also reported according to T_{NOW}.

4 Ambient intelligence adaptation

Nowadays constrained devices such as smartphones are furnished with global positioning system (GPS). Free Android/iOS applications can record contact locations in data stores (NoSQL) using RWS. In the same way, other health data, temperature, blood pressure, etc. can be captured and recorded. These data may drive real-time simulation.

IoT is a general concept that can be interpreted in different contexts. Different protocols and standards can be used for the communication between the devices and the Internet. As described in this paper, IoT connects to Internet through Internet protocols, such as HTTP, TCP and IP. The data received from IoT is represented as Web data in forms such as HTML, JSON, XML, and URI. This data is then organised into ontology presented in RDF graphs, for storing, analysing, and reasoning. The data is typically processed in service-oriented and Web-based computing environment. Repository, Mediator and Collector help reduce RDF graphs handling and complexity and play a main role in the tasks scheduling since related tasks are scheduled in groups and are running simultaneously. At this end, the IoT and its data are fully integrated into the Web and the virtual world. A convenient way of programming the IoT devices is critical to the success of the current organisation stack, which allows the massive proliferation of applications (Chen, 2016). Neto et al. (2017) presented a study on the use of industrial internet of things (IIoT) currently in industry context, its basic differences from IoT and its expansion possibilities pointing out some challenges related to a new approach within the industry. They argue that the complex interconnection which is made possible through the IIoT is able to optimise resources and reduce exponentially the costs of production processes in most stages and is gradually changing the direction of society in labour relations. Their advances in manufacturing processes are feasible as the internet of things is not simply inserting intelligence in equipment, but allowing interconnection, reconfiguring functions and anticipating loss of productivity or failures that might occur in real-time. Within this context, the IIoT can be understood as a broad and complex concept that encompasses asset and performance management areas, availability of increased data and intelligent corporate. Brozek and Jakes (2017) presented a study that addresses the potential of using mobile devices in a distributed simulation. The study also focused on the possibility of applying the various technologies and architectures in context of using mobile devices in simulation. Lamamra et al. (2018) presented an acquisition and processing of coded data from temperature sensors used in radiotherapy rooms of the Hospital Pierre and Marie Curie Centre (PMCC). Their aim was to acquire and check remotely the temperatures of rooms to trigger alarms and their control thereafter in order to avoid mistakes of manipulation which are deadly for patients if they happen or arise. Their system modelling was made before proceeding to the implementation in practice using an artificial neural network to acquire and decrypt the temperature data received from the sensors placed in the treatment rooms.

^{DEVS}Server can be considered as an autonomous decentralised system (ADS) that enables the system to continue to function in the event of component failures.

The mathematical modelling of infectious disease spreading has been extensively studied for a long time (Hethcote, 2000). A lot of epidemic models, such as the compartmental models that are composed of differential equations, have been developed and analysed (Hethcote, 2000; Xiao et al., 2011). 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 modelled and its transmission over complex heterogeneous networks where a node is an individual and an edge stands for

interaction between nodes allowing disease transmission (Hethcote, 2000; Pastor-Satorras and Vespignani, 2001; Moreno et al., 2002; Olinky and Stone, 2004; Liu and Hu, 2005; Yang et al., 2007; Zhang et al., 2009; Liu and Zhang, 2011; Xiao et al., 2011; Zhang et al., 2011, 2012). It is shown that the SIS (Barabási and Albert, 1999) and the SIR (Moreno et al., 2002) 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 capabilities are provided.

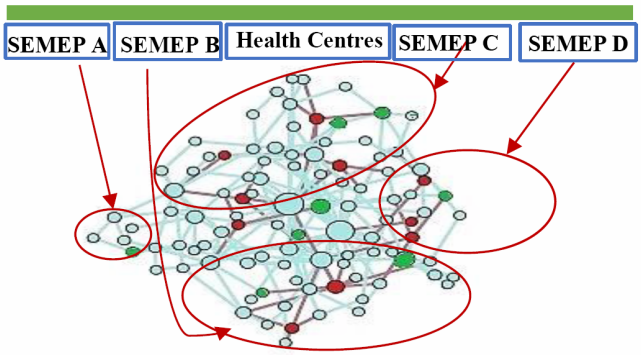
Spreading processes are strongly interacting with huge flow of quantitative social, demographic and behavioural data that may be used to improve the immunisation strategy. The topology of the pattern of contacts between individuals plays a fundamental role in determining the spreading patterns of epidemic processes embedding the mechanism of diverse infection periods and is an impact on the properties of the dynamical behaviours of the spreading process. The existing immunisation strategies are limited by their computational requirements and still have the problem of scaling in large networks. Optimal immunisation strategies shed light on how the role and importance of nodes depend on their properties and can yield importance rankings of nodes.

Contacts positions, social and environment information are dynamically and periodically gathered by devices before applying any strategy. The simulated strategy detects the appropriate contacts once running on ^{DEVS}Server. Contacts on constrained devices are dynamically advised to avoid a critical and unsafe location as a prediction due to AmI servers' simulation. Contacts can move safely avoiding disease contamination. Notification, data gathering and geographical visualisation applications for mobile are part of the platform and may be installed on contacts constrained devices. Typical information on disease can also be visualised and critical regions coloured on a map (admin and CI of Figure 5). Parallel diseases surveillance is captured so that contacts related to a specific disease are advised with their disease state.

5 Epidemic modelling

We consider a typical network (Figure 9). Nodes represent different individuals of a real system and links represent the connection between nodes. A connection is designed as a set of routes intersections. A route is the sequence of visited positions by each node of the network. Routes intersection design a pair of participating individuals which were in face-to-face close proximity (≈1–2m), with a temporal resolution of 20 s inside the radius of a designed area. Since the mobility of individuals can play an important role in the epidemic spreading process, we must gather such information using AmI devices. From AmI technological advances, we are now in a position to gather data describing the contacts in groups of individuals at several temporal and spatial scales and resolutions. Contacts are connected to the system via Smartphones with GPS and Health Sensors. Figure 9 shows the relation between contacts and their respective stores. Some contacts are stored in 4 SEMEPs stores (SEMEP A, B, C, and D). A city like Mostaganem has four SEMEPs. For example, a contact X from Mostaganem District is vaccinated at SEMEP A and SEMEP B with some vaccines, and he is vaccinated with many other vaccines at SEMEP C. He has been admitted in a public hospital of another city, reported in SEMEP D. To gather X information, we have to connect to the SEMEPs A, B, C, and D, in a distributed and parallel manner. Moves of individuals are also captured in their respective stores (Figure 9).

Figure 9 Global network dispatching (see online version for colours)



We must investigate the network dynamics affected by individual diversity of susceptibility and infectivity that may be caused by individuals' intrinsic differences that may lead in infections periods. For each node i, $\tau_i(t)$ and $\acute{\imath}_i(t)$ correspond to its contamination and immunity degrees and $S_i(t)$ is its state. To address the heterogeneous connectivity, we measure these two important degrees $\tau_i(t)$ and $\acute{\imath}_i(t)$. The state $S_i(t)$ may be susceptible 'S', infected 'I', recovered 'R' to correspond to a SIR model (Figure 9, blue ='S', red ='I', green='R') in widely held situations. Since it is computed dynamically, we can adopt any other model SI, SIS or SEIR (Liu and Zhang, 2011) as the Tuberculosis model below. We compute dynamically $\tau(t)$, the disease contamination threshold, and $\acute{\imath}(t)$, its immunisation threshold. $\tau_i(t)>\tau(t)$ means the contamination of i is strong otherwise it is weak and $\acute{\imath}_i(t) < \acute{\imath}(t)$ means i has a weak immunity otherwise it is strong. When a contact occurs between i and j, if $\tau_i(t)$ is strong, then j is strongly affected and if $\tau_i(t)$ is weak then j is weakly affected. A strong immunity means a susceptible individual will not be infected but his immunity degree will decrease and a weak immunity means a susceptible individual will be infected resulting in a state change. Each disease is associated to rules that change dynamically the states and the degrees of its related individuals and its thresholds.

Figure 10 The fitness computation

```
public void fitness(){
  double immunity_degree=0, contamination_degree=0;
  if(age<=40 & weight <=60 & socialLevel==0.2 & malnutrition==0.8 & smoking<=20 & alcoholism<=15 & itinerancy==1 & idu==1
    & prevTB==1 & physActivity==0 & corticoidTherapy==1 & chronicStress==1 & hiv==1 & diabetes==1 & cancer==1)
    immunityDegree=0.1;
  if(age<=40 & weight <=60 & socialLevel==0.3 & malnutrition==0.8 & smoking<=20 & alcoholism<=15 & itinerancy==1 & idu==1
    & prevTB==1 & physActivity==0 & corticoidTherapy==1 & chronicStress==1 & hiv==0 & diabetes==1 & cancer==1)
    immunityDegree=0.2;
     ……
  if(age<=30 & weight <=50 & socialLevel==0.8 & malnutrition==0.4 & smoking<=10 & alcoholism<=5 & itinerancy==0 & idu==0
    & prevTB==0 & physActivity==1 & corticoidTherapy==0 & chronicStress==0 & hiv==0 & diabetes==0 & cancer==0)
    immunityDegree=0.8;
  if(age<=30 & weight <=50 & socialLevel==0.8 & malnutrition==0.2 & smoking<=4 & alcoholism<=0 & itinerancy==0 & idu==0 &
    prevTB==0 & physActivity==1 & corticoidTherapy==1 & chronicStress==0 & hiv==0 & diabetes==0 & cancer==0)
    immunityDegree=0.9;
  if(age<=30 & weight <=60 & socialLevel==0.9 & malnutrition==0.1 & smoking==0 & alcoholism==0 & itinerancy==0 & idu==0 &
    prevTB==0 & physActivity==1 & corticoidTherapy==0 & chronicStress==0 & hiv==0 & diabetes==0 & cancer==0)
    immunityDegree=1.0;
  contaminationDegree= 1-immunityDegree;
  if (immunityDegree<=0.4 & previousState=="Ill") this.setState("Infected");
    else if(immunityDegree>0.4 & immunityDegree<=0.6) this.setState("Ill");
    else if(immunityDegree>0.6 & immunityDegree<=0.7 & previousState=="Ill") this.setState("UnderTreatment");
    else if(immunityDegree>0.7 & immunityDegree<=0.9 & previousState=="UnderTreatment") this.setState("Recovered");
    ……
    else this.setState("Susceptible");   }
```

Table 1 Data of Mostaganem City corresponding to 2015 used in simulations

	Masculine	*Feminine*	*Individuals*	*Percentage*
Total population	445,992 (50.21)	442,262 (49.79)	888,254	-
Urban population	203,551	201,848	405,399	45.64 %
2nd urban population	61,235	60,722	121,957	13.73 %
Sparse population	181,207	179,691	360,898	40.63%
Population <= 5 years old	49,170	47,495	96,655	10.88%
Population 6-19 years old	104,501	102,249	206,750	24.77%
Population 19 - 59 years old	264,790	262,420	527,210	63.17%
Population > 60 years old	27,635	29,992	57,627	6.90%
Detected cases of TB*			303	0.2%
Estimation of infected individuals *			413	0.3%
Cavitation forms*			104	34.32%
Detected cases of HIV*			3	0.99%
Risk factors*			-	65.4%
Total annual mortality*			-	0.03%
Negative Treatment*			-	0.1%
Failed Treatment*			-	0.2%
Relapsed*			12	3.96%
Treatment abandonment rate*			-	1.2%
Diagnosis delay (days)			48	-

Notes: All percentages are with respect to the total population of Mostaganem, except (*) which is with respect to the total number of TB patients of Mostaganem district.

Table 2 Population repartition of Mostaganem district according to age range

DISTRICT	*0–5 years*		*6–19 years*		*16–59 years*		*60 years and +*		*Total*	
	M	*F*	*M*	*F*	*M*	*F*	*M*	*F*	*M*	*F*
MOSTAGANEM	9,141	8,754	18,211	17,891	49,610	51,083	5,640	7,184	78,055	80,391

For example, if $\tau_i(t) > \tau(t)$ and $\acute{\imath}_i(t) < \acute{\imath}(t)$ and $S_i(t)$ = 'S' then $S_i(t)$ = 'I'. Some rules change the thresholds values of $\tau(t)$ and $\acute{\imath}(t)$. Some rules compute the values of $\tau_i(t)$, $\acute{\imath}_i(t)$ for each node i. These degrees depend of environmental, social and health data like sex, age, weight, climate, humidity, etc. Data mining tools help discretise these data more efficiently. Figure 10 is a set of some rules related to Tuberculosis. The periodic change of temperature, humidity profiles, or even the succession of school terms and holidays, can lead to periodic phenomena of epidemics and affect these degrees. Some mutated viruses propagate again, rendering a new epidemic outbreak. This process occurs repeatedly; the immunisation strategy needs to handle this phenomenon to give a better description of these seasonal epidemics.

We follow Montañola-Sales et al. (2015) and Prats et al. (2016) to consider a typical tuberculosis case study inside Mostaganem Virtual City. Tables 1 and 2 give some related data to Tuberculosis in Mostaganem. This model is based on general knowledge about the natural history of tuberculosis gathered from the previous SEMEPs stores as RDF graph. There are two essential characteristics of TB that must be taken into account in any epidemiological model. On the one hand, an infected individual does not necessarily develop an active disease. On average, only 10% of infected people become ill. Moreover, a person remains infected for a long time and may develop active tuberculosis after several years. Infected people are usually not diagnosed. On the other hand, only an ill individual can disseminate the infection. The infection rate increases if the patient has TB with cavitation. Once an individual is diagnosed with TB, the pharmaceutical treatment lasts six months. Once the treatment is finished, the possibility of getting sick again remains at 1% for two years post-diagnosis (Zhang et al., 2011, 2012).

Once infected, the individual may develop active TB according to a certain annual probability that decreases with infection time following a quadratic during the seven years' post-infection. It is neglected for the subsequent years ($t > 7$ years). The discrete time step of one day is assumed, as mentioned. Since simulation time does not cover periods longer than ten years, the approximation is good enough. This probability has a ten-fold increase for immunosuppressed people, and it is multiplied by a factor of 1.5 if there are other risk factors. The chance of becoming an individual ill with TB is evaluated at each time step for all infected persons. Globally, the average of 10% of the infected developing an active disease is satisfied. The possibility of relapse (getting sick again) for recovered patients is also evaluated daily according to the individual relapse probability. Once an individual gets sick, the disease time counter starts running until the assigned individual diagnosis time is reached. If the disease is developed, then the individual enters the sick state. In this state, the agent not only presents symptoms but also becomes infectious (that is, can spread the tuberculosis among healthy or treated people). This state considers the time since the disease became active, whether the lesions include cavitated tissue, and the number of days that will pass until the individual is diagnosed (diagnosis time delay).

Each individual has a particular diagnosis time that is randomly assigned when becoming ill, following a truncated exponential with mean diagnosis delay (MDD) of 45 days. Once diagnosed, the agent enters the medical treatment state and stops spreading TB. The treatment state accounts for the number of days since the treatment started and finishes in 180 days. There is a certain probability that an individual may abandon the treatment before finishing it. This possibility is evaluated at every time step for each patient under treatment, according to the input abandonment probability. If an individual abandons the treatment during the initial 15 days post-diagnosis, the patient becomes sick again. If the patient abandons the treatment after 15 to 180 days post-diagnosis, then he or she is considered to be recovered but with a certain probability of relapse the following two years. This probability is considered to linearly decrease from 100% at 15-day abandonment to 1% at the 180-day treatment period.

The basic entities of the model are individuals. We consider five possible states according to TB infection dynamics: healthy, infected, ill (i.e., with active TB), under treatment and recovered. The state variables of the individuals refer mainly to their status in the infection cycle as well as the time spent in such phases and individual diagnosis time when ill.

Other individual state variables and parameters are age, weight, itinerancy, possible risk factors (e.g., smoking), and possible immunosuppression (mainly AIDS). Once a person is infected, the presence (or not) of pulmonary cavitation is also considered.

We used individuals' dataset with the following attributes to compute the fitness function (Figure 10) that defines the immunity $\acute{\imath}_i(t)$ and contamination $\tau_i(t)$ degrees and the state $S_i(t)$ of an individual. These attributes are: weight, age, smoking, alcoholism, HIV, corticoid therapy, previous TB, itinerancy, physical activity, We used individuals' dataset with the following attributes to compute the fitness function (Figure 10) that defines the immunity $\acute{\imath}_i(t)$ and contamination $\tau_i(t)$ degrees and the state $S_i(t)$ of an individual. These attributes are: weight, age, smoking, alcoholism, HIV, corticoid therapy, previous TB, itinerancy, physical activity, injecting drug user (IDU), chronic stress, diabetes, cancer, social level, malnutrition, immunity

degree, contamination degree, and state. Some initial conditions are initialised at the beginning of simulation.

Data shown in Tables 1 and 2 are used to generate the initial population of the Virtual City of Mostaganem (Figure 1). Number of healthy, infected, sick, under treatment and recovered individuals; MDD; mean treatment abandonment rate; and individuals with risk factors and with AIDS. Some other initial variables were assigned randomly: individual age, weight, and time spent in the infection state (Zhang et al., 2011, 2012).

The time step is set to one day, and the simulation may cover up to a period of one or more years. A healthy individual is assumed to not have any bacteria in his organism and therefore is not able to infect others as well as not presenting symptoms. Generally, this individual can be seen as a person that does not smoke, not drink, not having caught a previous TB, performing a sport, not having any form of stress, having a healthy home with high social level (job + vacation, etc.), not having used corticoids, having a stable eating and sleeping regimes, with no diabetes or cancer or HIV. On the other side, a risky individual may have some disturbance in some of these factors. However, common properties such (age, weight, itinerancy, risk factors, diabetes and previous TB) must be well suited if randomly generated.

Individuals may get infected with TB by their close contacts. This depends on the type of TB disease that the sick person has either cavitated or non-cavitated. A cavitation is considered to double the infection probability. These probabilities are fixed to reproduce the ratios of ten infections/cavitated TB sick and five infections/ non-cavitated TB sick in 60 days. Once infected, an individual may develop active TB according to a certain annual probability that decreases with infection time following a quadratic during the seven years' postinfection. It is neglected for the subsequent years ($t > 7$ years). Since simulation time does not cover periods longer than ten years, the approximation is good enough. This probability has a ten-fold increase for immunosuppressed people, and it is multiplied by a factor of 1.5 if there are other risk factors. The chance of becoming an individual ill with TB is evaluated at each time step for all infected persons. Globally, the average of 10% of the infected developing an active disease is satisfied. The possibility of relapse (getting sick again) for recovered patients is also evaluated daily according to the individual relapse probability. Once an individual gets sick, the disease time counter starts running until the assigned individual diagnosis time is reached (Prats et al., 2016). To deal with the previous assumptions, we define a chromosome with risk factors as follows:

60	77	0.3	0.8	15	10	0	0	1	1	0	1	0	1	0

The used risk factors are respectively: age, weight, social level, malnutrition, smoking, alcoholism, itinerancy, IDU, previous TB, physical activity, corticoid therapy, chronic stress, HIV, diabetes, and cancer (composing rules of fitness in Figure 10). This individual is 60 years old, 77 kg, a poor man, with high malnutrition, smoking 15 cigarettes/day, drinking 10 litres/month, with no itinerancy, no drug user, previous TB, sport practice, no corticoid experience, no stress, with diabetes and no cancer.

A contamination is no more than a crossover between chromosomes of individuals in contacts. We compare these chromosomes gene by gene. If the respective genes are close, we move to the next. Otherwise, we use this gene as the crossover point and apply a permutation between the remaining genes as shown below.

In the example below, we assume all genes differences in the interval [0, 5]. A max of five years for the age is not significant, so 1st genes (ages) are close since ($|32–29| \leq 5$). Also, weights ($|48–56| \leq 5$) are close. A social level equal to 0.3 or 0.8 means that the 1st person is poor and the 2nd one is, may be, rich. The same thing for malnutrition for the values 0.8 (high malnutrition) or 0.4 (almost healthy). We, also, assume that a difference of more than five cigarettes ($|12–5| \geq 5$) is important and produces a crossover. The cut is at smoking genes.

32	48	0.3	0.8	12	15	1	0	1	0	1	1	1	1	0
29	46	0.8	0.4	5	4	0	0	0	1	0	0	0	0	0

According to these values and the rules of Figure 10, the 1st and the 2nd persons have an immunity degree = 0.2 (0.8) and a state = 'Infected' ('Susceptible') respectively.

After applying a crossover, a permutation occurs as follow:

32	48	0.3	0.8	5	4	0	0	0	1	0	0	0	0	0
29	46	0.8	0.4	12	15	1	0	1	0	1	1	1	1	0

This scenario means that a behaviour contamination occurred, in other words these individuals became friends and changed habits (behaviours). If one of them was a smoker or a drinker, then the second has taken this habit and became also a smoker or a drinker.

Since some values such as previous TB, diabetes, etc. cannot change, a mutation must be performed on the initial chromosome, according to the last result concerning each chromosome. Only, the genes in red squares, below, are may change.

1st individual result

32	48	0.3	0.8	12	15	1	0	1	0	1	1	1	1	0
32	48	0.3	0.8	5	4	0	0	0	1	0	0	0	0	0
32	48	0.3	0.8	5	4	0	0	1	1	0	0	1	1	0

2nd individual result

29	46	0.8	0.4	5	4	0	0	0	1	0	0	0	0	0
29	46	0.8	0.4	12	15	1	0	1	0	1	1	1	1	0
29	48	0.8	0.4	12	15	1	0	0	0	1	1	0	0	0

The first individual does not change his weight, nor his social level, nor his malnutrition, but his smoking and his drinking have decreased. He gets a home. He still has a previous TB, he starts playing physical activity. He stops taking corticoid. He has now no stress.

The second individual does not change his weight, nor his social level, nor his malnutrition, but his smoking and his drinking have increased. He loses his home. He still has not a previous TB, he stops playing physical activity. He starts taking corticoid. He has now a stress.

According to these new values and the rules of Figure 10, the immunity degree and the health state become respectively for the two persons 0.7 (0.5) and 'recovered' ('Ill') respectively. So, we can see that the first individual grants from an immunity degree of 0.2 to 0.7 and from a state of 'Infected' to 'Recovered', only after changing his residence, taking a treatment, and moving to a new best neighbourhood. On the other side, the second individual which had an immunity degree of 0.8 and a state 'Susceptible' is got decreased to an immunity degree of 0.5 and becomes 'Ill'.

These changes take effect over many years, by starting with low/high values and increasing/decreasing to new values. This risk factors effect is straightforward.

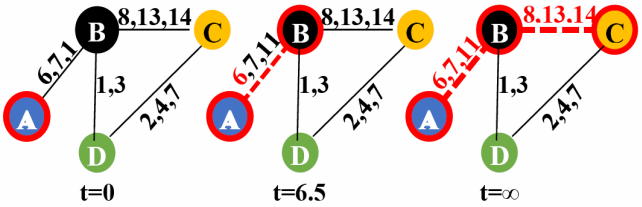
To apply crossover and mutation, let us consider that a district is a set of individuals in the same neighbourhood. The within individuals are permanently changing behaviours over chromosomes crossovers and mutations. We define a district chromosome which is the mean chromosome of the chromosomes of individuals of the same district. This mean chromosome is updated dynamically. It shows the state of the district. It gives a quick idea on the effects of the related disease. It shows, also, if a district is a poor or rich district, a district of delinquents, and so on. When a new individual is introduced in such district for a long time, he performs, periodically, a crossover with this mean chromosome which changes instantly. His chromosome changes slowly until he became a delinquent, a rich or a poor.

The set of RDF triples of any contact defines his RDF graph (his ontology). This ontology includes his previous vaccinations, nurses, centres, virus related to vaccinations, weights, heights, type of home including humidity, climate, temperature, pressure, etc. These data are required to compute an individual's immunity and contamination degrees and disease thresholds. Some of these data are temporal, which means that they are not significant after a deadline time. We need to collect the network topology and its knowledge to apply efficient acquaintance immunisation strategies.

We consider datasets describing the contacts occurring in a population during a time interval $[T_B, T_E]$ and a predefined space area, T_B and T_E are respectively the beginning and the end times. Only contacts inside these area and time interval are included in the temporal network. Contacts can reach or leave dynamically the area at that time interval. Such information is gathered via contacts smartphones/sensors using ^{DEVS}Server APIs deployed on the SEMEPs. This network is fully distributed over many ^{DEVS}Servers. Each one builds its own subgraph. The full network is obtained by assembling these subgraphs into a TSC. Semantic is embedded in the RDF triples of each node and its links. The final temporal network of concerned contacts defines the full intelligent temporal network. Figure 11, extracted from the global network of Figure 9, corresponds to the subnet of SEMEP A. The times of the contacts between vertices A–D are indicated on the edges. Assume that, for example, a disease starts spreading at vertex A and spreads further as soon as a contact occurs. The dashed lines and vertices show this spreading process for three different times. The spreading will not continue further than what is indicated in the $t = \infty$ picture, i.e., D cannot get infected. However, if the spreading started at vertex D, the entire set of vertices would eventually be infected. Aggregating the edges into one static graph cannot capture this effect that arises from the time ordering of contacts. To apply an immunisation strategy, we use and increment a global time T_{NOW} between T_B and T_E. The next discrete event time is the increment step. When a move or an information effect occurs, the system state (individuals' state and degrees) is updated. Accordingly, we focus on rules that do not ignore the consequences of the time and space ordering by, e.g., projecting out the interaction times.

When an epidemiologist connects to ^{DEVS}Server to apply an immunisation strategy, he begins by creating the TSC in his own node by calling the Mediator *createTSC*() method. After that, his Mediator add the area and the time interval data inputs to the TSC, using the *writeToTSC*() method. The Mediator executes its *constructTemporalGraph*() primitive to ask the Repository to retrieve all concerned Mediators URLs. A Repository is updated every time information is added to a node, telling that this node is owning that information. Once getting mediators URLs, the Mediator allows these Mediators to access its own TSC using the *readFromTSC*() and *writeToTSC*() primitives.

Figure 11 Illustration of reachability issue and the intransitivity of temporal networks (more specifically a contact sequence) (see online version for colours)



Each Mediator invokes its own *findAllContactsAround Position*() primitive to collect its subgraph forwarding this to its own Collector that invokes its *getContactRoutes*() and *getContactInfo*() primitives to get routes and information of concerned contacts. Finally, the temporal network is ready to be used in the TSC, it summarises all the events occurring in the system. Immunisation strategy can now be applied.

For instance, consider two epidemiologist mobile applications. The first one consumes information from its

TSC on SEMEP A using standard ontologies to link epidemiologists working jointly on some epidemics. Since the application uses TSC, this information is available for the Mediators in the shared space from SEMEPs B, C, D. The Mediators in the same party (session) populate dynamically the contacts moving in the case of real time immunisation. The second epidemiologist independent application, which try to immunise another disease, may notify the first epidemiologist when the two diseases have common contacts moving which means that immunisation strategies can affect each other. The second application may populate the first TSC by dumping information retrieved from its TSC. Prior to the dump, the application semantises the information according to the ontology. Finally, it periodically looks into the space to check the contacts leaving the space area and remove them from the network. TSC provides space decoupling, so applications do not know where the information is physically located. They just access the space requesting, removing or providing information. Therefore, the conceptual scheme the developer should have in mind is that multiple nodes interact through different spaces (represented by clouds). Each space contains multiple graphs (represented by temporal networks). These graphs are composed by a set of triples represented by nodes and links within each SEMEP and corresponding to the area and time interval provided. However, AmI environments are mainly populated by mobile devices and sensors. These devices frequently join and leave the spaces and the information they hold constantly changes. Thus, AmI environments are highly dynamic. As a consequence, we decided to adopt a distributed strategy which locally stores, or even generates on demand, the information necessary to answer a query. Doing so, we ensure the freshness of the responses regarding the sensed data. The main drawback is that whenever a node is temporarily unavailable its contents become unavailable for the rest of the nodes too. However, this faithfully represents the actual state of the space. Our TSC design does this by allowing each node, no matter how complex or simple it is, to manage its own information. Besides, it establishes a communication channel with the space it wants to join to, i.e. with each of the nodes belonging to it. Queries are propagated to other nodes which previously joined that space (regardless of who they are at each moment). Possible responses are received from them using the same communication channel. In this scheme, each node actually has the sets of graphs locally allowing knowledge distribution strategies.

5.1 DEVS Specification

Let us consider the example of Figure 11, four contacts (A, B, C, D) are considered as atomic temporal DEVS models. The coupling is defined temporally by the network topology. When simulation time advances, each model looks in TSC to find if it has a coupling or data updates at that time. If so, it generates its outputs according to that dynamic coupling and writes its outputs to the TSC. According to Figure 11, the start time is zero and the end time is 14. So, we consider the time interval = [0, 14] to be the window in which we observe the simulation. The DEVS global coupled model of the example is shown in Appendix B. Let us detail the simulation steps ordered as follows:

1 Generate the virtual city according to data of table.

2 Apply TB propagation.

3 Apply immunisation strategy.

5.2 Virtual city generation

This step ends with the temporal network building in TSC. Since a city is a set of districts and a district is a set of individuals and each individual has a chromosome and a route that is a sequence of positions, we need a DEVS model *VirtualCity* that generates the overall virtual city population, a DEVS model *GenerateDistrict* that generates a district population, a DEVS model *GenerateContact* that generates an individual and an DEVS model *GenerateChromosome* that generates a chromosome. Each contact has his own routes composed of positions giving a DEVS model *route* of DEVS model *position*.

These models are typical generators with two states. The *deltaExt*() changes the generator to busy. Since the generator is busy, the *deltaInt*() generates an output and the *out*() writes it as RDF triples in the TSC. Let say that a generated contact is a graph that is a set of RDF triples describing each contact. The *initialise*() function of the *VirtualCity* reads the data of Table 1 from a file, initiates the number of districts and their names. For each district, it initiates also the numbers of individuals. The first and last names of each individual are selected randomly from files of names given as input to the *VirtualCity* model, masculine and feminine names are considered. The address of each individual is obtained by calling a Google RWS by giving a well randomly selected position (longitude and latitude).

All the chromosomes data are randomly generated using rules, considering that a child is not a smoker or drinker. These rules are used to help obtaining useful chromosomes. We used some previous individuals' data to generate these rules as described in Section 3, paragraph 3.

Route generation consists of choosing a centre and a radius, and generates a position in that area. A position is defined by a longitude, latitude, a start time (the time the individual reached the position), a last time (the time the individual left the position), and an address.

Google(http://maps.googleapis.com/maps/api/geocode/json),Geonames(http://api.geonames.org/findNearbyPlaceName?),Gisgraphy(http://services.gisgraphy.com/geoloc/geolocsearch?) RWS are used conjointly to help route generation. Addresses and routes fit in the area of Mostaganem districts.

A result of an individual random generation extracted from a full virtual city of Mostaganem generation is presented in Appendix B. The simulation uses the Mostaganem District population that is 158,446 respecting the number of men and women and the data of Table 1. Routes were randomly chosen to be fewer than ten positions

each. We used three storage nodes to store data assuming that Mostaganem has only three SEMEPs. Each node is deployed on a distinct computer following the topology of Figure 9. At each node, a [DEVS]Server (Figure 7) is also deployed. Individuals are stored according to their respective SEMEP.

The woman 'Fallak Abou' of the example (Appendix B) is 19 years old and lives at 'Avenue Khemisti Mohamed, Mostaganem'. Her weight is 55 kg, she is poor (social level very low) but has no malnutrition. She has no stress, no cancer, no previous TB, no itinerancy, no corticoid therapy. She is not alcoholic but drinks sometimes, same for smoking. She is not diabetic but presents a little glucose's rate. She has a home and does not undertake any physical activity. She has a medium immunity and she is ill but not infectious. She leaves her home to go to an urban station (position_1), then to the university (position_2) then to the city centre (position_3), then back to the urban station (position_4) and to home. We can validate this itinerary using the start and end time to confirm that it is serial. In a real-life system, health data and moves of that woman are gathered dynamically using her smartphone and sent to her respective store. Some health data are already in the SEMEP store.

5.3 Propagation phase

In this step, the first thing to do is to capture the temporal network in the area where the disease is suspected. This area is specified with a centre position and a radius. We also need the start time when the disease begins to propagate and the last time when the propagation is estimated to end. So, we have to look for each individual that reaches this area in this interval time. This search is done on each SEMEP whom individuals are included. The whole temporal network is composed of subnetworks related to these SEMEPs.

The simulation of the propagation starts at the previous start time and ends at the previous last time. The epidemiologist starts his CI and ask his DEVS model, *propagation*, to start building the network. Since we have three instances of [DEVS]Server, we have three *propagation* models, each one on its respective server. Each *propagation* model calls its respective Mediator that put the related information in its TSC. *Propagation* model asks the Mediator to collect the list of its own individuals that are in that area and interval time. This list is put by the Mediator in its TSC.

Once *propagation* model has the list of concerned individuals, it starts RWS to meet the number of individuals. Each RWS is seen as an Individual DEVS model. Now that we have *individual* models running, *propagation* models acting as a coordinator, and the first *propagation* as root-coordinator, the simulation is ready to start. *Propagation* models read/write from/in the TSC of the root. *Individual* models and each *propagation* model read/write from/in their local TSC.

At each node, *propagation* model initialises itself and asks its *individual* models to do so. Each *individual* computes the time of its next event TN that is the time of its next move. *Propagation* asks eminent *individual* models, models with the minimum next event time, to perform their moves, searches from the eminent *individual* models which were at the same place and the same time, applies a crossover operation between their chromosomes and then applies the fitness computation. Each *propagation* model reports its contacts into the global TSC as follow:

1st *propagation* model

<u: Birth_270000> <u: hasPosition> <u: Position_3>.
<u: Birth_270004> <u: hasPosition> <u: Position_5>.
<u: Birth_270008> <u: hasPosition> <u: Position_1>.

2nd *propagation* model

<u: Birth_270002> <u: hasPosition> <u: Position_3>.
<u: Birth_270005> <u: hasPosition> <u: Position_4>.
<u: Birth_270009> <u: hasPosition> <u: Position_5>.

3rd *propagation* model

<u: Birth_270001> <u: hasPosition> <u: Position_3>.
<u: Birth_270003> <u: hasPosition> <u: Position_6>.
<u: Birth_270007> <u: hasPosition> <u: Position_4>.

The root-coordinator deduces and adds these triples

<u: Birth_270000> <u: isWith> <u: Birth_270002>.
<u: Birth_270000> <u: isWith> <u: Birth_270001>.
<u: Birth_270002> <u: isWith> <u: Birth_270001>.
<u: Birth_270005> <u: isWith> <u: Birth_270002>.
<u: Birth_270007> <u: isWith> <u: Birth_270001>.

Five (5) crossover operations occur between the above individuals producing change of their chromosomes. These operations increase or decrease the respective immunity and contamination degrees, states are changing accordingly by the fitness performance. If a state is affected with 'Infected', then a propagation occurred. This contact is added to the 'Infected' list. Ill individuals are added to the 'Ill' list and so on. *Propagation* models stay in this loop until *individuals* consumes all their moves or until the simulation reaches the interval last time. At the end of the propagation, the number of infected gives the prevalence and the incidence factors. Holland (1975, 1992) has shown that crossover and mutation operators on chromosome can destruct it, resulting in the death of the individual. He used a probability of this destruction.

5.4 Immunisation phase

The immunisation strategy process follows the propagation process. Since the propagation produces lists of infected, ill, under treatment, recovered and healthy individuals, the immunisation strategy consists of producing solutions to eradicate the disease. It proposes a solution for each category. In this paper, we only deal with the infected category.

A simple solution consists of vaccinating or quarantining all the infected individuals. An important challenge is to define immunisation strategies that discover a meaningful group of individuals (community) that are strongly related to the disease. Once this community discarded, disease can be eradicated. To find the strongly related individuals to a disease, we need to use protocols as stated by Starnini et al. (2013) where the authors consider that an immunisation strategy is defined as the choice of a set of nodes (individuals) who cannot catch nor transmit the disease. They argued that this choice is performed according to a ranking of the nodes of the temporal network. They consider various ranking strategies, focusing in particular on the role of the training window during which the nodes' properties are measured in the time-varying network. We follow their work but we use the immunity and contamination degrees to rank nodes. The node with the higher contamination degree and the lower immunity degree is the best ranked. To be more careful, a contamination degree in the interval [0.8–1] is considered to be very strong and an immunity degree in [0–0.4] is very weak. We can test other least values for immunity and highest values for contamination respectively. The best choice is designed by the Epidemiologist under the disease circumstances.

To rank nodes the previous intervals, the *immunisation* DEVS model computes the rank of every infected individual in the 'Infected' list, then it applies a mutation operation on the best ranked ones found in the interval [0–0.4] for immunity and [0.8–1] for contamination. The *immunisation* model restarts the propagation process and compares the obtained list with the previous one. If they have the same size, the immunisation process stops. If the two lists are different, the immunisation process loops again until the list sizes are equal. After each *individual* move, *immunisation* model sends a message to the CI to visualise this move with the corresponding *individual* information showing their behaviour on the map. Map colouring indicates the gravity of the propagation. Another message (notification) is sent to the individual smartphone preventing him to perform the move if a contamination is simulated to occur.

6 Conclusions

A new concept and ideas for distributed simulation are implemented as [DEVS]Server showing that the design of [DEVS]Server principles (TSC paradigm, mediator, collector, and repository) can achieve interoperability. The TSC interoperability style at the web level (RWS) allows [DEVS]Server to take advantage of new Web-based features or technologies. On the other hand, [DEVS]Server provides better interoperability applying RWS principles among the TSC paradigm.

[DEVS]Server is specially designed to deal with DSS but it can be applied to similar systems. DSS with Epidemic Modelling 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, [DEVS]Server 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 [DEVS]Server cloud implementation, we aim to consider implementing the visualisation process with its web interface. A real-time simulation with online data under AmI assistance to show the [DEVS]Server ability to a wide range of interaction and pursue its intelligent interoperability aspect is to be considered. We hope to choose a testbed of volunteers to validate this ability.

Since distributed intelligence is a part of the IoIT that has adequate computing capacity to perform complex computations, and the gang scheduling and real-time scheduling techniques in ad hoc distributed systems and on the elastic cloud environment ensure the coordination of these intelligent devices. We can improve [DEVS]Server performance with replacing IoT constrained devices with the so called IoIT objects and increase the Mediator with these scheduling capabilities. The recent Intel IoT-enabled architecture, such as Galileo and Edison, makes it easy to program these devices as Web services (Chen, 2016).

We also consider integrating the immunisation strategies of the other categories such as 'Ill', 'UnderTreatment', 'recovered' and 'healthy' so that 'Ill' recovers soon, 'under treatment' does not relapse or abandon treatment, and 'recovery' does not relapse. Such proposals are related to artificial intelligence and are part of actual AmI systems.

The tuberculosis example with its GP application is under work and will be another paper to be submitted soon. We also aim to use its agent-based simulation implementation using GP. Others diseases are under study to make [DEVS]Server a fully integrated epidemic modelling tool.

Acknowledgements

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 SIF Team under the supervision of the LIO labs and the SEMEP services of the Algerian Health Ministry.

Funding

This research received no specific grant from any funding agency in the public, commercial or not-for-profit sectors.

References

ACIMS software site [online] http://acims.asu.edu/software (accessed May 2018).

Al-zoubi, K. and Wainer, G. (2013) 'RISE: a general simulation interoperability middleware container', *Journal of Parallel and Distributed Computing*, Elsevier, Vol. 73, No. 5, pp.580–594.

Amamra, L., Mokaddem, M. and Atmani, B. (2012) *Mesure de la qualité de la vaccination guidée par les données*, Sixième Atelier sur les Systèmes Décisionnels ASD'2012, 1–3 Avril, Université Saad Dahlab, Blida, Algérie, ISBN: 978-9947-0-3416-3.

Augusto, J.C. and McCullagh, P. (2007) 'Ambient intelligence: concepts and applications', *Computer Science and Information Systems*, Vol. 4, No. 1.

Barabási, A.L. and Albert, R. (1999) 'Emergence of scaling in random networks', *Science*, Vol. 286, No. 5439, pp.509–512.

Barigou, F., Mokaddem, M., Atmani, B. and Beldjilali, B. (2010) 'Towards an automated system for extracting named entity from medical reports', *International Congress on Models Optimization and Security of Systems*, Du 29–31 May, Tiaret, Algeria.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) 'The semantic web', *Scientific American*, Vol. 284, No. 5, pp.34–43.

Brahami, M., Atmani, B. and Mokaddem, M. (2010) 'CARTOCEL: un outil de cartographie des connaissances guidée par la machine cellulaire CASI', *Conference Extraction et Gestion des Connaissances EGC*, pp.625–626.

Bravo, J., Cook, D. and Riva, G. (2016) 'Ambient intelligence for health environments', *Journal of Biomedical Informatics*, Elsevier, Vol. 64, pp.207–210 [online] https://doi.org/10.1016/j.jbi.2016.10.009 (accessed May 2018).

Brozek, J. and Jakes, M. (2017) 'Application of mobile devices within distributed simulation-based decision making', *Int. J. Simulation and Process Modelling*, Vol. 12, No. 1, pp.16–28.

Chen, Y. (2016) 'Analyzing and visual programming internet of things and autonomous decentralized systems', *Simulation Modelling Practice and Theory*, Elsevier, Vol. 65, pp.1–10 [online] https://doi.org/10.1016/j.simpat.2016.05.002 (accessed May 2018).

Chen, Y. (2018) *Service-Oriented Computing and System Integration: Software, IoT, Big Data, and AI as Services*, Kendall Hunt Publishing Company, ISBN: 9781524951658 [online] https://he.kendallhunt.com/product/service-oriented-computing-and-system-integration-software-iot-big-data-and-ai-services (accessed May 2018).

Fujimoto, R. (2000) *Parallel and Distribution Simulation Systems*, John Wiley and Sons, New York.

Gelernter, D. (1985) 'Generative communication in Linda', *ACM Transactions on Programming Languages and Systems (TOPLAS)*, Vol. 7, p.80112.

Gómez-Goiri, A., Orduña, P., Diego, J. and López-de-Ipiña, D. (2014) 'Otsopack: lightweight semantic framework for interoperable ambient intelligence applications', *Computers in Human Behavior*, Elsevier, Vol. 30. pp.460–467, DOI: 10.1016/j.chb.2013.06.022.

Guinard, D. (2011) *A Web of Things Application Architecture Integrating the Real-world into the Web*, PhD ETH Zurich.

Hethcote, H.W. (2000) 'The mathematics of infectious diseases', *Society for Industrial and Applied Mathematics SIAM Rev*, Vol. 42, No. 4, pp.599–653.

Holland, J.H. (1975/1992) *Adaptation in Natural and Artificial Systems*, Second ed. (First ed., 1975), MIT Press, Cambridge, MA.

Honkola, J., Laine, H., Brown, R. and Tyrkko, O. (2010) 'Smart-M3 information sharing platform', *IEEE Symposium on Computers and Communications ISCCs*, pp.1041–1046.

Jafer, S., Liu, Q. and Wainer, G. (2013) 'Synchronization methods in parallel and distributed discrete-event simulation', *Simulation Modelling Practice and Theory*, Elsevier, Vol. 30, pp.54–73 [online] https://doi.org/10.1016/j.simpat.2012.08.003 (accessed May 2018).

Khushraj, D., Lassila, O. and Finin, T. (2004) 'sTuples: semantic tuple spaces', *The First Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services*, MOBIQUITOUS, pp.268–277.

Lamamra, K., Allam, A. and Afiane, M. (2018) 'Artificial neural networks for acquisition and processing of sensors data in a radiotherapy application', *International Journal of Simulation and Process Modelling (IJSPM)*, Vol. 13, No. 1, pp.15–23.

Liu, J. and Zhang, T. (2011) 'Epidemic spreading of an SEIRS model in scale-free networks', *Communication in Nonlinear Science and Numerical Simulation*, Elsevier, Vol. 16, No. 8, pp.3375–84.

Liu, Z. and Hu, B. (2005) 'Epidemic spreading in community networks', *Europhys Lett*., Vol. 72, No. 2, pp.315–21.

Mittal, S. (2006) 'Extending DoDAF to allow DEVS-based modeling and simulation', *Special Issue on DoDAF, Journal of Defense Modeling and Simulation*, Vol. 3, No. 2, pp.95–123.

Mittal, S., Risco-Martín, J.L. and Zeigler, B.P. (2007) 'DEVS-based simulation web services for net-centric T&E', *Proceedings of the 2007 Summer Computer Simulation Conference SCSC*, San Diego, CA, USA, pp.357–366.

Mittal, S., Risco-Martín, J.L. and Zeigler, B.P. (2007) 'DEVSML: automating DEVS execution over SOA towards transparent simulators', *Proceedings of the 2007 Spring Simulation Multi-conference SpringSim '07*, San Diego, CA, USA, pp.287–295.

Mittal, S., Risco-Martín, J.L. and Zeigler, B.P. (2009) 'DEVS/SOA: a cross-platform framework for net-centric modeling and simulation in DEVS unified process', *Simulation*, Vol. 85, No. 7, pp.419–450.

Mokaddem, M., Bahnes, A. and Atmani, B. (2011) 'Création dynamique orientée services de contenu pédagogique en e-learning', *Journées Doctorales du Laboratoire d'informatique d'Oran, JDLIO '2011*, Oran, Algérie.

Mokeddem, S., Atmani, B. and Mokaddem, M. (2013) '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, pp.53–64, ISSN: 2231-5403, ISBN: 978-1-921987-23-6.

Mokeddem, S., Atmani B. and Mokaddem, M. (2015) 'An effective feature selection approach driven genetic algorithm wrapped Bayes Naïve', *Int. Nat. Journal of Data Analysis Technics and Strategies*, ISSN: online: 1755-8069 ISSN print: 1755-8050.

Montañola-Sales, C., Gilabert-Navarro, J.F., Casanovas-Garcia, J., Prats, C., López, D., Valls, J., Cardona, P.J. and Vilaplana, C. (2015) 'Modeling tuberculosis in Barcelona, a solution to speed-up agent-based simulations', *Proceedings of the 2015 Winter Simulation Conference*, Yilmaz, L., Chan, W.K.V., Moon, I., Roeder, T.M.K., Macal, C. and Rossetti, M.D. (Eds.).

Moreno, Y., Pastor-Satorras, R. and Vespignani, A. (2002) 'Epidemic outbreaks in complex heterogeneous networks', *Eur Phys J B*., Vol. 26, No. 4, pp.521–9.

MS4 Me (2013) *MS4 Me Software V 3.0 (Markov Modeling Capability)* [online] http://www.ms4systems.com (accessed May 2018).

Neto, A.R., Ribeiro, M.F.G., Cunha, G.G. and Landau, L. (2017) 'The industrial internet of things and technological innovation in its applications for resources optimisation', *Int. J. of Simulation and Process Modelling*, Vol. 12, No. 6, pp.525–534.

Nixon, L.J., Simperl, E., Krummenacher, R. and Martin-Recuerda, F. (2008) 'Tuple space-based computing for the semantic web: a survey of the state-of-the-art', *Knowledge Engineering Review*, Vol. 23, No. 2, p.181–212.

Olinky, R. and Stone, L. (2004) 'Unexpected epidemic thresholds in heterogeneous networks: the role of disease transmission', *Phys Rev E*, Vol. 70, p.030902(R).

Pastor-Satorras, R. and Vespignani, A. (2001) 'Epidemic dynamics in scale-free networks', *Phys Rev Lett*., Vol. 86, No. 14, p.3200.

Pirkkalainen, H. and 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)*, Vol. 3, No. 3, pp.45–57.

Prats, C., Montañola-Sales, C., Gilabert-Navarro, J.F., Valls, J., Casanovas-Garcia, J., Vilaplana, C., Cardona, P.J. and López, D. (2016) 'Individual-based modeling of tuberculosis in a user-friendly interface: understanding the epidemiological role of population heterogeneity in a city', *Frontiers in Microbiology*, January, Vol. 6, Article 1564.

Seo, C. and Zeigler, B.P. (2009) 'Interoperability between DEVS simulators using service oriented architecture and DEVS namespace, a joint symposium devs integrative M&S (DEVS) and high performance computing (HPC)', *Proceedings of the Spring Simulation Conference.*

Seo, C. (2009) *Interoperability between DEVS Simulators using Service Oriented Architecture and DEVS Namespace*, PhD dissertation, University of Arizona, Tucson, Arizona.

Seo, C., Zeigler, B.P., Coop, R. and Kim, D. (2013) 'DEVS modeling and simulation methodology with MS4 Me software tool', *Proceedings of the Symposium on Theory of Modeling and Simulation: DEVS Integrative M&S Symposium (TMS-DEVS)*, April, San Diego, CA.

Silva, M.J., Da Silva, F.A.B., Lopes, L.F. and Couto, F.M. (2010) 'Building a digital library for epidemic modelling', *Proceedings of ICDL 2010 – The International Conference on Digital Libraries*, New Delhi, India.

Starnini, M., Machens, A., Cattuto, C., Barrat, A. and Pastor-Satorras, R. (2013) 'Immunization strategies for epidemic processes in time-varying contact networks', *Journal of Theoretical Biology*, Elsevier, Vol. 337, pp.89–100.

Tolk, A. (2010) 'Interoperability and composability', Banks, C. and Sokolowski, J. (Eds.): *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*, pp.373–402, Wiley, New Jersey.

Varela-Candamio, L. and 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*, Vol. 28, No. 6, pp.1301–1308.

Wainer, G., Al-Zoubi, K., Mittal, S., Risco Martín, J.L., Sarjoughian, H. and Zeigler, B. (2010) *Discrete-Event Modeling and Simulation: Theory and Applications* in Wainer, G. and Mosterman, P. (Eds.), pp.389–494 (Chapters 15–18), CRC Press, Taylor and Francis.

Wainer, G., Madhoun, R. and Al-Zoubi, K. (2008) 'Distributed simulation of DEVS and Cell-DEVS models in CD++ using web services', *Simulation Modelling Practice and Theory*, Elsevier, Vol. 16, No. 9, pp.1266–1292.

Wainer, G. (2009) *Discrete-Event Modeling and Simulation: a Practitioner's Approach*, CRC Press, Taylor and Francis Group, Boca Raton, Florida.

Xiao, Y., Zhou, Y. and Tang, S. (2011) 'Modelling disease spread in dispersal networks at two levels', *Math Med Biol*., Vol. 28, No. 3, pp.227–44.

Yacoub, A., Hamri, M.E.A., Frydman, C., Seo, C. and Zeigler, B.P. (2017) 'DEv-PROMELA: an extension of PROMELA for the modelling, simulation and verification of discrete-event systems', *International Journal of Simulation and Process Modelling (IJSPM)*, Vol. 12, Nos. 3/4, pp.313–327.

Yang, R., Wang, B.H., Ren, J., Bai, W.J., Shi, Z.W., Wang, W.X. and Zhou, T. (2007) 'Epidemic spreading on heterogeneous networks with identical infectivity', *Phys Lett A*., Vol. 364, Nos. 3–4, pp.189–93.

Zeigler, B.P. and Sarjoughian, H.S. (2012) *Guide to Modeling and Simulation of Systems of Systems*, p.393, Springer, Berlin, Germany.

Zeigler, B.P., Praehofer, H. and Kim, T. (2000) *Theory of Modeling and Simulation*, Academic Press, San Diego, CA.

Zeigler, B.P., Nutaro, J.J. and Seo, C. (2017) 'Combining DEVS and model-checking: concepts and tools for integrating simulation and analysis', *Int. J. Simulation and Process Modelling*, Vol. 12, No. 1, pp.2–15.

Zhang, H. and Fu, X. (2009) 'Spreading of epidemics on scale-free networks with nonlinear infectivity', *Nonlinear Anal Theory Methods Appl*., Vol. 70, No. 9, pp.3273–8.

Zhang, J.P. and Jin, Z. (2012) 'Epidemic spreading on complex networks with community structure', *Appl Math Comput*., Vol. 219, No. 6, pp.2829–38.

Zhang, J.P. and Jin, Z. (2011) 'The analysis of an epidemic model on networks', *Appl Math Comput*., Vol. 217, No. 17, pp.7053–64.

Appendix A

Restful web service model invocation

```
@XmlRootElement public class ModelSchema {
String name, owner, creationDate,
String description, initialiaze, deltaExt, deltaInt;
int inports; int outports;
byte[] modelclass ;

@XmlRootElement
public class ModelAccess {
private static Key key = null;
private static KVStore mystore;
public static final String MOD_PREFIX = "DEVS";
public static KVStore getStore() { }
public static void insertModel(String username,
                               String modelName,
                               int inports, int outports,
                               String owner,
                               String creationDate,
                               String description,
                               String inisliaze,
                               String deltaExt,
                               String deltaInt,
                               File modelFile){
}
public static ModelSchema getModel(String username,
                               String modelName){
ModelSchema model = new ModelSchema( nameOfModel,
                               inports, outports,
                               owner,
                               dateCreation,
                               description,
                               initialiaze,
                               deltaExt, deltaInt,
                               modelClass);
return model;
    }
public static void deleteModel(String username,
                               String modelName){
     }
}
@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 not found ");
         }
         catch (InstantiationException e) {
                  System.out.println("instance not found ....");
      }
      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);
      for (Field field : o.getClass().getDeclaredFields())
      for (Method method : o .getClass().getDeclaredMethods())
            o.getClass().getDeclaredField("state").equals("S");
Method method =
o.getClass().getDeclaredMethod("DeltaExt",String.class);
      Object r = method.invoke(o, "infection" );
      } catch (Exception io) {
      io.printStackTrace();
      System.out.println("model "+ modelName+" not found");
      } finally { }
      ModelSchema mm = mv.readModel(username,
      modelName);
……..
}
System.out.println("model "+ modelName+" not found");
} finally { }
ModelSchema mm = mv.readModel(username,
modelName);
……..
}
```

Appendix B

A snapshot of a random contact generation with his route

```
<u:Birth_270000>  <u:hasAddress>  <Avenue Khemisti Mohamed, Mostaganem, Algeria>
<u:Birth_270000>  <u:hasBirthDate>  <Mon Sep 14 00:00:00 WAT 1998>
<u:Birth_270000>  <u:hasChromosome>  <Chromosome_Birth_270000>
<u:Birth_270000>  <u:hasPhone>  <0554611902>
<u:Birth_270000>  <u:hasMail>  <Falak    _Abou @gmail.com>
<u:Birth_270000>  <u:hasLastName>  <Abou >
<u:Birth_270000>  <u:hasFirstName>  <Falak         >
```

```
<u:Chromosome_Birth_270000>  <u:hasContaminationDegree>  <0.4^^http://www.w3.org/2001/XMLSchema#decimal>
<u:Chromosome_Birth_270000>  <u:hasSex>  <F>
<u:Chromosome_Birth_270000>  <u:hasDiabetes>  <1>
<u:Chromosome_Birth_270000>  <u:hasSmoking>  <1>
<u:Chromosome_Birth_270000>  <u:hasPhysicalActivity>  <1>
<u:Chromosome_Birth_270000>  <u:hasIDU>  <1>
<u:Chromosome_Birth_270000>  <u:hasCancer>  <0>
<u:Chromosome_Birth_270000>  <u:hasItinerancy>  <0>
<u:Chromosome_Birth_270000>  <u:hasPreviousTB>  <0>
<u:Chromosome_Birth_270000>  <u:hasCorticoidTherapy>  <0>
<u:Chromosome_Birth_270000>  <u:hasChronicStress>  <0>
<u:Chromosome_Birth_270000>  <u:hasAlcoholism>  <4>
<u:Chromosome_Birth_270000>  <u:hasImmunityDegree>  <0.6^^http://www.w3.org/2001/XMLSchema#decimal>
<u:Chromosome_Birth_270000>  <u:hasAge>  <19>
<u:Chromosome_Birth_270000>  <u:hasMalnutrition>  <0.9>
<u:Chromosome_Birth_270000>  <u:hasState>  <Ill>
<u:Chromosome_Birth_270000>  <u:hasSocialLevel>  <0.2>
<u:Chromosome_Birth_270000>  <u:hasWeight>  <55>
```

```
<u:position_1>  <u:hasAddress>  <station_urbaine>
<u:position_1>  <u:hasEndDate>  <Fri Jan 01 23:42:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_1>  <u:hasLongitude>  <0.092078^^http://www.w3.org/2001/XMLSchema#decimal>
<u:position_1>  <u:hasStartDate>  <Fri Jan 01 23:19:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_1>  <u:hasLatitude>  <35.931482^^http://www.w3.org/2001/XMLSchema#decimal>

<u:position_2>  <u:hasAddress>  <universite_kharouba>
<u:position_2>  <u:hasLongitude>  <0.099867^^http://www.w3.org/2001/XMLSchema#decimal>
<u:position_2>  <u:hasEndDate>  <Sat Jan 02 01:01:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_2>  <u:hasLatitude>  <35.951777^^http://www.w3.org/2001/XMLSchema#decimal>
<u:position_2>  <u:hasStartDate>  <Sat Jan 02 00:57:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>

<u:position_3>  <u:hasEndDate>  <Sat Jan 02 03:00:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_3>  <u:hasStartDate>  <Sat Jan 02 02:40:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_3>  <u:hasAddress>  <boulevard_benghettat_mohamed>
<u:position_3>  <u:hasLatitude>  <35.932759^^http://www.w3.org/2001/XMLSchema#decimal>
<u:position_3>  <u:hasLongitude>  <0.081677^^http://www.w3.org/2001/XMLSchema#decimal>

<u:position_4>  <u:hasAddress>  <station_urbaine>
<u:position_4>  <u:hasStartDate>  <Sat Jan 02 03:09:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_4>  <u:hasLongitude>  <0.092078^^http://www.w3.org/2001/XMLSchema#decimal>
<u:position_4>  <u:hasEndDate>  <Sat Jan 02 03:43:22 WAT 2016^^http://www.w3.org/2001/XMLSchema#string>
<u:position_4>  <u:hasLatitude>  <35.931482^^http://www.w3.org/2001/XMLSchema#decimal>
```