Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 43 (2015) 18 – 25

Procedia Computer Science

ICTE in Regional Development, December 2014, Valmiera, Latvia

Architecture for Distributed Simulation Environment

Artis Aizstrauts[a]*, Eglis Ginters[a], Mikelis Baltruks[a], Marjan Gusev[b]

[a]*Sociotechnical Systems Engineering Institute, Vidzeme University of Applied Sciences, Cesu street 4, Valmiera LV - 4200, Latvija*
[b]*Faculty of Information Sciences and Computer Engineering, University Sts Cyril and Methodius, blvd. Goce Delcev 9, 1000 Skopje, R. Macedonia*

Abstract

The article describes a custom made architecture for distributed simulation. Initially intended for Skopje Bicycle Inter-modality simulation, it has proven itself as being universal, adaptable and usable for many different purposes. It supports not only the set of simulation models, GUIs and data representations, but also many users running simulations at the same time. Such architecture brings certain challenges forth. The authors discuss methods for dealing with these challenges.

Peer-review under responsibility of the Sociotechnical Systems Engineering Institute of Vidzeme University of Applied Sciences

Keywords: Distributed simulation; ECE; Repast Symphony

1. Introduction

Policy planning and decision making are among those domains that could benefit from possibilities granted by modeling of potential outcomes. But policy modeling as the object of research is versatile and complex. EC FP7 project No.287119 FUPOL „Future Policy Modelling" (http://www.fupol.eu) aims at a new approach to traditional policy modelling[1]. The simulation and visualization tools assist to governments and policy makers in the whole policy life cycle and help avoiding voluntary and wrong decisions. In FP7 FUPOL the set of policy domains as the object for analysis and simulation involves community facilities (area design, open space), urban transport and economics, land use planning, sustainable tourism and other. Each of them comprises some use cases.

* Corresponding author.
E-mail address: artis.aizstrauts@va.lv

Peer-review under responsibility of the Sociotechnical Systems Engineering Institute of Vidzeme University of Applied Sciences
doi:10.1016/j.procs.2014.12.004

Usually to simulate sociotechnical problems there are several models involved to solve particular problem and not all of them may be designed with the same simulation type. It is common to design Multi-agent system (MAS) as a distributed simulation[2]. Previously for distributed simulation communication Easy communication environment (ECE) was designed to provide communication among simulation models[3], but it cannot be used in this case as it is. ECE is more focused on direct data exchange between simulation models, but Skopje Bicycle Inter-modality simulator requires wider functionalities within the communication environment.

At first this article briefly introduces Skopje Bicycle Inter-modality but after discusses the architecture of distributed simulation environment in detail. Finally some concluding remarks are given and further research directions are defined.

2. Skopje Bicycle Inter-modality simulator use case

The main objective of this project is the development of a software solution that will offer the City of Skopje and its citizens the opportunity to simulate the occupancy and usage of bicycle stations and parking lots. The overall goal is to increase the number of people that use bicycles as transport by taking several different measures such as establishing bicycle inter-modality, initiating the development of parking lots, rent-a-bicycle facilities, new bicycle paths and improving existing bicycle paths. Such project would help the authorities of City of Skopje in improving the scheduling and resource planning, initiation and creation of new projects involving the bicycle stations and parking lots. The citizens of City of Skopje will also be involved in the decision making process by constantly communicating and expressing their opinion to the authorities, making the whole process more transparent and efficient[4]. Public simulation is accessible at http://prod.fupol.lv:8080/skopjebicycle. Skopje Bicycle Inter-modality simulator ensures validation of potential activities of Skopje Municipality using multi-agent systems simulation (ABM/MAS) in Repast Symphony[5] environment. It is based on statistical data provided by Skopje Municipality. It not only gives possibility for citizens to participate directly in policy planning and crafting, but also provides municipality with valuable feedback through ticketing[6].

3. Simulator description and requirements

This section describes the simulation types, user roles and overall functionality provided by Skopje Bicycle Inter-modality simulator. The explanations will help to understand the specific requirements of the architecture in the following section. Skopje Bicycle Routes Simulator software is created to find optimal solution for bicycle station and track building location in the City of Skopje. To encourage the intermodal transport, the citizens are also integrated in the collection of ideas on how bicycle intermodality can be fostered. The citizen participation is supported including the use of a simulation tool.

3.1. Simulation types

There are two types of simulations proposed in the simulator: Simulation 1 simulates bicycle usage for one week with given custom parameters (set of tracks, set of bicycle stations, weather conditions and month of the year). Based on these parameters simulation will generate bicycle usage for each track and station; Simulation 2 is a way of combining multiple object changes to see which changes affect the bike usage the most. Simulation 2 includes multiple project definitions, affordable project set, combination generation and best result displaying. Each combination is a mix of one or more projects added to existing simulation settings. Each Simulation 2 consists of one or more one year simulations with conditions set by projects. These types of simulation were defined by Skopje Municipality.

3.2. User roles

The user roles that were defined by customer are:

- Administrator – creates and manages users;

- Master user – setup (using import) of initial simulator configuration, change configuration, export configuration, and realize both simulation 1 and 2;
- Manager – can view ticket reports, view and answer tickets sent by users, suggest new configuration and projects that will improve the biking experience in Skopje.
- Public user – can access the visual simulation interface for Simulation 2.

These are the main requirements to the simulator. Whatsoever it should be noted that the greatest constraints are the public access to the simulator – simulation can be run many times by many users at the same time – that means, the architecture has to be flexible and able to process large amount of requests. The next section describes the distributed simulator architecture in more detail.

4. Distributed simulator architecture

The simulator is part of FP7 FUPOL project and the requirements are obtained partly from project technical guidelines. Each simulator can be viewed as one of FUPOL functionalities, i.e. end-user can use either one simulator or several. Simulator within the context of this paper is model and GUI that helps to control the model and view simulation results. The project proposes that simulation GUI is a web application, and the model based on ABM/MAS technologies. At the same time the models have to be executable also by third party software that is developed within other work packages of FP7 FUPOL. That means the architecture has to be able to run all models together, and each model might have different GUI. Based on previous research is it known that there must be a central element for such an architecture that is responsible for the data exchange between architecture components, such as Simulator GUI and model[7,8]. Simulation GUI and Model can be considered as plugins within the architecture. The architecture has to ensure that these components can be changed dynamically (plugged or unplugged) without interrupting the operation and without interfering with other components. At the same time it has to be scalable and ensure that its components can be distributed and operate within a cloud. A component (for example, a model) can also exist in several instances, to maximize the amount of simulation sessions within one unit of time. Another important issue to be considered is the fact that simulators can be publicly available. That means that the architecture has to be able to process large amount of requests generated by large amount of users. In this case the amount of users exceeds the number of models (model instances) and it is technically impossible to run all the requested simulation instantly with existing model instances. The architecture has to be able to handle this issue by introducing some waiting queue for requested simulation. From the viewpoint of system integrity the architecture has to ensure mechanisms to be able to operate also in the situations when one of elements is not working properly due to different problems. For example when the connection with the database is interrupted and models have nowhere to store the data. The architecture has to embody mechanisms to ensure that these data are not lost.

Taking into account everything mentioned previously, it was decided to construct the architecture according to Fig.1. The architecture can be divided into logical elements that communicate with each other and some of the parts can be distributed over the cloud technologies.

The further sections describe the elements and operations of the architecture shown in Fig.1.

4.1. Simulation GUI

Simulation GUI is a tool or platform that ensures user interaction with model, lets them define configuration of the simulation, execute it and view the results. The architecture is designed so that the simulation GUI could operate as an independent element, that has to be able to communicate with the architecture using Advanced Message Queuing Protocol (AMQP)[9]. From the viewpoint of architecture, there is no difference, how the simulation GUI is organized, what programming language is used, what additional functionalities it has etc. However it is important that GUI is able to communicate with Message broker/Message queue. So far the FP7 FUPOL experience shows that different additional options are usually annexed to the simulation. For example, user management (including authorisation), simulation reports, advanced analytics and ticketing functionality (public participation). Fig. 2 shows a single view from Skopje Bicycle Inter-modality simulator.

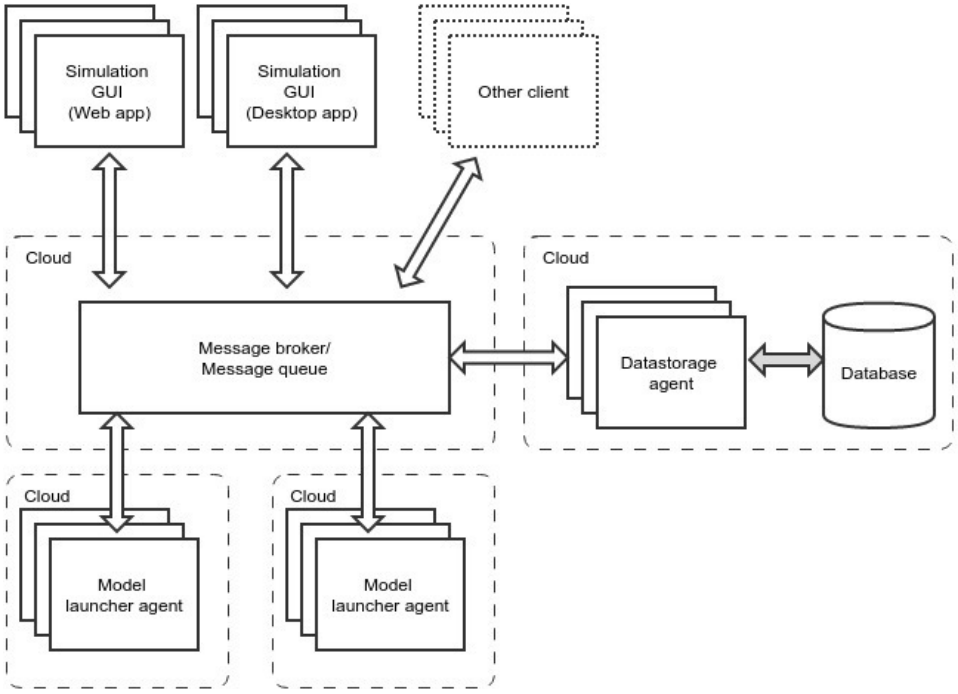

Fig. 1. Distributed simulation architecture.

The architecture is designed in a way that allows the Skopje Bicycle Inter-modality simulator GUI (see Fig. 2) to be fully customizable. Thus GUI is adjustable to specific cases and groups of users and their needs. For example (see Fig. 2), different user-friendly features as interactive maps sometimes are helpful for public use, but not always. Sometimes it is much more important that GUI is orientated towards field specialists that need to be able to fully control simulation and do not need decorative appearance or moving objects.

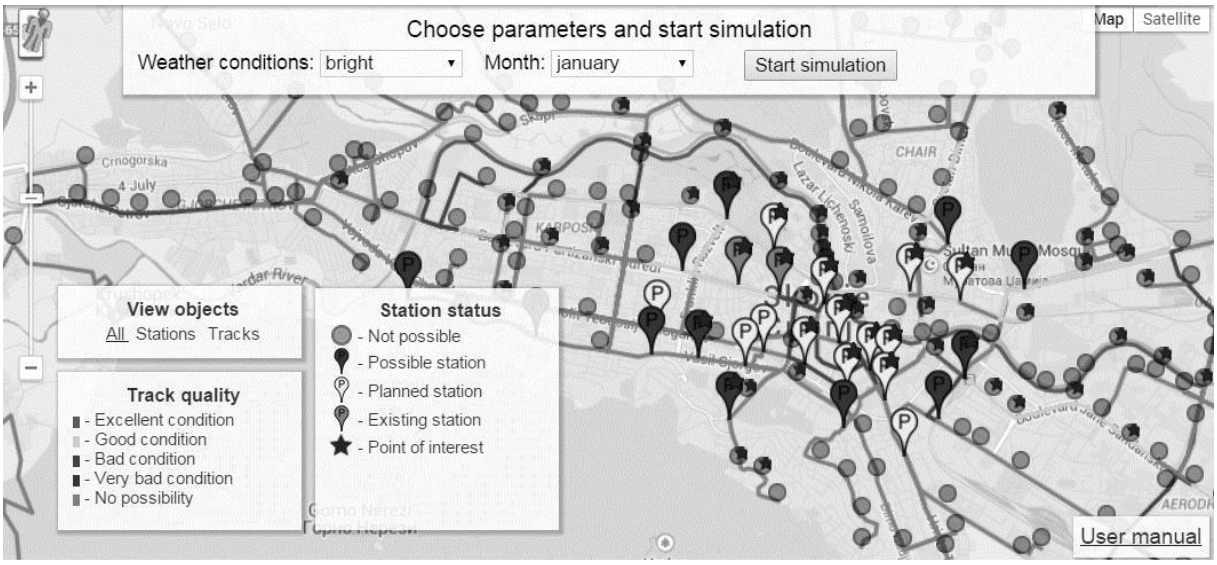

Fig. 2. Skopje Bicycle Inter-modality simulator GUI.

It is worth mentioning that the architecture has to be able to support both mentioned solutions – they are equal from the point of view of the architecture. They both send specific commands to start a simulation with specific parameters. This architecture is built to ensure only the infrastructure for designing of simulator (simulation GUI + simulation model). In other words, the developers of simulator have to define the communication themselves. The architecture has to ensure only two main functions - the delivery of message from simulation GUI to simulation model and also the storage (and retrieval) of messages in database (see Fig.1). The content of the message that is sent from simulation GUI to run the simulation will be explained further.

4.2. Message broker/Message queue

Message broker/Message queue (MQ) ensures all communication within the architecture. It also ensures that simulation GUI is able to deliver message to Model launcher agent to start respective model. And the Model launcher forwards the simulation parameters to the model. MQ may be used also for any custom communication among the architecture components if needed. The message broker is a message-oriented middleware (MOM) - software or hardware infrastructure supporting sending and receiving messages between distributed systems. MOM allows application modules to be distributed over heterogeneous platforms and reduces the complexity of developing applications that span multiple operating systems and network protocols[10]. In fact it would also be possible to use any other task/job queuing tools, but the Message Broker was chosen because it is able to ensure equivalent functionality as task/job queuing tools and is more orientated to message delivery, that is important in this case, because simulation results are sent to database also using MQ.

Before integration into architecture, several MQ were analysed. The summary of conclusions and the mayor differences can be seen in Table 1.

Table 1. Message broker solutions.

MQ	Open source or similar license	Administration/ monitoring panel	Self-hosted	Client language support	Previous experience
ActiveMQ[11]	Yes	Yes	Yes	Many[a]	Yes
Kafaka[12]	Yes	Yes	Yes	Many[b]	No
RabbitMQ[13]	Yes	Yes	Yes	Many[c]	Yes
Amazon SQS[14]	No	Yes	No	Java, .Net, Node.js, PHP, Python, Ruby	Yes

Within FP7 FUPOL project open-source products are considered first. If it is not possible to find an open source product, then the commercial counterparts are considered. At the same time in this case it was important that MQ is self-hosted, and does not use open source cloud platform. Thus Amazon SQS was not further considered.

For the system to be easily maintained it is important that there is a convenient monitoring tool that enables the management of communication – helps understand which nodes are alive/dead, monitor bottlenecks, analyse throughput, etc. Therefore Administration/monitoring panel is used as a criterion and all MQs included in the analysis do have this panel. It is important that MQ has ready libraries for as many programming languages as possible. That allows the expansion of the use of architecture and there are fewer constraints to simulation GUI. Mostly all MQ tools included in the analysis support many programming languages except Amazon SQS.

---

[a] ActionScript, Ajax, C, C++, C#, .Net, Delphi, Erlang, Flash / ActionScript, Haskell, Perl, PHP, Pike, Python, Ruby

[b] Python, Go (AKA golang), C, C++, .net, Clojure, Ruby, Node.js, Storm, Scala DSL , HTTP REST, JRuby, Perl, Java, PHP, Python, Go, Erlang, etc.

[c] Java, Ruby, Python, .NET, PHP, Perl, C / C++, Erlang, Node.js, Go, Common Lisp, Haskell, Ocaml, COBOL, etc.

Taking into account all above mentioned arguments, only RabbitMq and ActiveMQ were analysed in more detail. A throughput test was performed to make the final decision. The test was carried out with out-of-box configuration for each tool (both had persistent mode on). The parameters of the experiment were following:

- Size of message – 5KB;
- One message receiver;
- 1, 5 and 10 message senders;
- Computer - Intel® Core™ i5-3320M CPU @ 2.60GHz × 4, 11.4 GB RAM.

Table 2. MQ performance test results

MQ	1 sender	5 senders	10 senders
ActiveMQ	10000 msg/s	2000 msg/s	1800 msg/s
RabbitMQ	30000 msg/s	6500 msg/s	3500 msg/s

Based on performance test results (see Table 2) it was decided to use RabbitMQ as message broker for simulation environment architecture designing under the framework of FUPOL project.

4.3. Model launcher

Model launcher is custom made software that launches Repast Symphony models. The command to launch the model is received from MQ. To launch the simulation Simulation GUI has to send a message to MQ with a reference that this message has to be included in queue and named “new simulation”. Respectively every Model launcher one by one receives messages from this queue and each simulation is designated to different Model launcher. This kind of architecture design allows to dynamically increasing the number of Model launchers, if it is necessary (in case queue "new-simulation" is starting to get crowded). Fig. 3 shows the elements that are involved in launching a simulation and storing the data/results of it in the database. As previously mentioned MQ contains a queue that is named “new-simulations” and “stores” simulation launch requests. To store the data in database Model Launcher sends data to queue named „save-simulation-data".

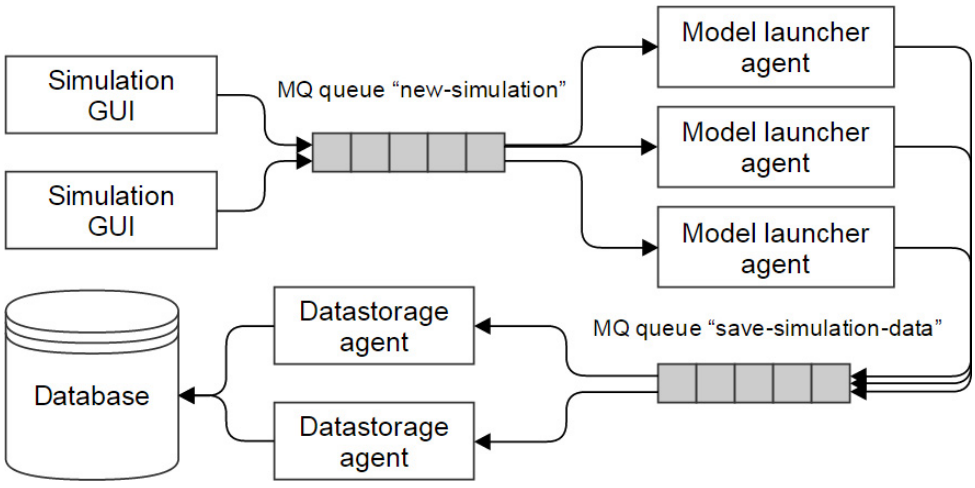

Fig. 3. Start simulation message flowchart.

Simulation GUI inserts message in MQ queue "new-simulation". The message is in JSON (JavaScript Object Notation)[15] format.

The Model launcher determines which model has to be launched according to message attribute “simulation”. The data that can be found in the message attribute “params” are delivered (by Model launcher) to the model before the launch of it. Within this architecture the developers of Simulation GUI and the developers of the model have to decide what data the attribute “params” will contain. It is a closed agreement as the architecture does not provide any mechanism for third parties to find out what parameters are needed for each model. Such mechanisms are provided similarly by HLA with Federation Object Model[9].

4.4. Datastorage agent and Database

Architecture allows data storing in the database. The main function of database is to accumulate simulation results created by model. Datastorage agent’s task is to read these data from MQ queue “save-simulation-data” and store them in database. The architecture anticipates that database vendor has no crucial role – with datastorage agent there are many possibilities to create different database adapters, thus widening the scope of architecture realizations. For now, adapters that support PostgreSQL and MongoDB have been created. Message broker has two queues that are checked by Datastorage agent on a regular basis:

- "Save-simulation-data" – in this queue those JSON format messages are inserted, that consist of data that has to be stored in the database;
- "Get-simulation-data" – in this queue those JSON format messages that include inquiry for simulation data. Datastorage agent understands inquiries with AND and OR comparisons.

Fig. 4 shows all steps, how data is retrieved from database. In this case it shows that Simulation GUI creates data request in QM queue "get-simulation-data” that is then read by Datastorage agent. The message is being parsed and then processed within database. When Datastorage agent has obtained all necessary data, the results are formatted in JSON and returned to Simulation GUI, using MQ queue "reply queue".

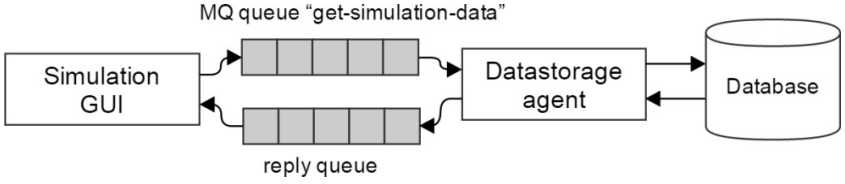

Fig. 4. Fetch data from database.

The architecture provides also the possibility to run many Datastorage agent nodes, thus improving performance.

5. Conclusion

This article describes distributed simulation modelling architecture that was developed for the purposes of FP7 FUPOL project. The requirements defined by project posed certain challenges for the developing of such architecture.

Easy communication environment (ECE) is designed to provide communication among simulation models. Although ECE has some great design ideas and features, it proved to be too tightly bounded for direct simulation model communication. In this case ECE could not be used as it is and for this project a new architecture had to be designed, taking into account previous gained experience with communication in distributed simulation environments[8, 2]. This architecture had to balance between two integration aspects – being easy and universal. The architecture had to be flexible to implement various simulation models, regardless of simulation models and

software (AnyLogic, RePast, etc.). At the same time the architecture had to be simple to implement for developer of simulators.

The authors describe the architecture that correspond the various requirements defined by stakeholders. The main elements of the architecture are – Simulation GUI, Message broker/Message queue, Model launcher and Datastorage agent and database.

Further research and developments are needed to make the input parameter structure and data structure generated by model available publicly within the architecture. Another perspective for further work is the Message broker attachment to each Model launcher. This would ensure that data will not be lost in cases when connection with main Message broker is failed.

References

1. FUPOL project homepage. Retrieved: 05.10.2014, URL: http://www.fupol.eu/en.
2. Silins, A., Ginters, E., Aizstrauta. D. Easy Communication Environment for Distributed Simulation. In World Scientific Proceedings Series on Computer Engineering and Information Science 3. "Computational Intelligence in Business and Economics" Proceedings of the MS'10 International Conference. ISBN 978-981-4324-43-4. 15-17 July, 2010. Barcelona, Spain. pp.91-98.
3. Aizstrauts, A., Ginters, E., Aizstrauta, D. Step by step to Easy Communication Environment for Distributed Simulation. In: Annual Proceedings of Vidzeme University of Applied Sciences „ICTE in Regional Development 2009/2010". ISBN 978-9984-633-20-6. 2011. Valmiera, Latvia. pp. 1-13.
4. Ginters, E., Aizstrauts, A., Dreija, G., Ablazevica, M., Stepucev, S., Sakne, I., Baltruks, M., Piera, M-A., Buil, R., Gusev, M., Velkoski, G., 2014. Deliverable 4.6 – Intermediate Release FUPOL Simulator Software advanced prototype. Skopje Bicycle Inter-modality Simulator. 224 p.
5. Repast Simphony, Retrieved: 05.10.2014, http://repast.sourceforge.net/repast_simphony.php
6. Ginters, E., Aizstrauts, A., Dreija, G., Ablazevica, M., Stepucev, S., Sakne, I., Baltruks, M., Piera, M-A., Buil, R., Gusev, M., Velkoski, G. Skopje bicycle inter-modality simulator – E-involvement through simulation and ticketing. Proceedings of the European Modeling and Simulation Symposium. 2014. 978-88-97999-38-6. pp 557-262
7. Aizstrauts, A., Ginters, E., Aizstrauta, D., Sonntagbauer P. Easy Communication Environment on the Cloud as Distributed Simulation Infrastructure. In: Proceedings of the 5th WSEAS World Congress on Applied Computing Conference (ACC '12). Recent Advances in Computing Engineering Series 2. ISBN: 978-1-61804-089-3. ISSN: ISSN: 1790-5109. 2-4 May, 2012. Faro, Portugal. pp. 173-179.
8. Ginters, E., Silins, A., Andrusaitis, J. "Communication in distributed simulation environment". In Proceedings of 6th WSEAS International Conference on Systems Science and Simulation in Engineering. ISBN 978-960-6766-14-5. 21-23 November, 2007. Venice, Italy. pp. 217-221.
9. Advanced Message Queuing Protocol - Wikipedia, the free encyclopedia, Retrieved: 05.10.2014 http://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol
10. Message-oriented middleware - Wikipedia, the free encyclopedia, Retrieved: 05.10.2014, http://en.wikipedia.org/wiki/Message-oriented_middleware.
11. Apache ActiveMQ, Retrieved: 01.10.2014, http://activemq.apache.org/
12. Apache Kafka, , Retrieved: 01.10.2014, http://kafka.apache.org/
13. RabbitMQ - Messaging that just works, Retrieved: 01.10.2014, http://www.rabbitmq.com/
14. Amazon Simple Queue Service, Retrieved: 01.10.2014, http://aws.amazon.com/sqs/
15. JSON homepage, Retrieved: 05.10.2014 , http://www.json.org/