SoS-centric Middleware Services for Interoperability in Smart Cities Systems

Frederico Lopes, Stefano Loss
Metropole Digital Institute
Federal University of Rio Grande do Norte
Natal - Brazil
fred@imd.ufrn.br, momoloss10@gmail.com

Altair Mendes, Thais Batista
Department of Computing and Applied Mathematics
Federal University of Rio Grande do Norte
Natal - Brazil
altairbmendes@gmail.com
thais@ufrnet.br

Rodger Lea
School of Computing and Communications
Lancaster University, UK
rodger@comp.lanc.ac.uk

ABSTRACT

Modern cities are supported by many IT systems managed by distinct public and private agents. Such legacy systems are often incompatible since, in general, they use old, dependent and non-standardised technologies. This results in an environment in which there is no interoperability among smart city systems, preventing richer and more interesting applications to be used by citizens, companies, and city administration. An alternative to solve the lack of interoperability is the adoption of a System-of-Systems (SoS) approach. A SoS is a set of independent and heterogeneous *constituent systems* that interoperate to accomplish a global mission. The collaboration among such constituent systems enables a SoS to offer new functionalities that cannot be provided by any of these systems working as individual entities. The goal of this paper is to propose SoS-centric middleware services to support the management and execution of SoS in Smart Cities environments in a dynamic, transparent and scalable way. The proposed services, once integrated into a smart city platform, support interoperability among different systems operating in a city. Moreover, this paper also presents a motivational case study to make it clear the issues that must be addressed when multiple independent systems are brought together to provide a new Smart City service or application.

Categories and Subject Descriptors

C.2.4 **[Computer-Communication Networks]:** Distributed Systems – distributed applications.
D.2.12 **[Software Engineering]:** Interoperability

Keywords

System-of-Systems (SoS); Interoperability; Middleware; Smart Cities.

1.INTRODUCTION

Nowadays, there are a lot of IT systems (for traffic management, smart grid, efficient buildings, healthcare, public safety, among others areas) deployed in cities that collaborate to make cities smarter. However, many of those systems originate from different providers, are managed by distinct public and private agents, often on their own computational infrastructure, and work in isolation. In addition, such systems are often incompatible since in general they use old, dependent and non-standardised technologies. This results in an environment in which there is no interoperability among the systems and no holistic comprehension of the smart solutions, hampering the goal of fully addressing the urban development challenges.

The lack of integration brings some problems, for example, how distinct, heterogeneous, fully decentralized and independent systems can closely collaborate with each other to achieve application goals. Despite the wide range of middleware [1, 2, 3, 4, 5, 6] providing sophisticated abstractions to develop smart city applications, in general, they do not fully address the requirements presented in this document. For instance, some middleware focuses on IoT environments and in others abstractions for smart city [1, 2, 3]. Others middleware focuses on the integration of public and private cloud platforms for smart cities environments [4, 5] rather than interoperability. In contrast, OverStar [6] does have a strong focus on interoperability but its generality fails to address Smart City situations.

In fact, the plethora of systems for making cities smarter can be regarded as a SoS, and can be supported by well-known SoS approaches that support different heterogeneous, stand-alone and largely independent constituent systems, that need to interoperate to achieve a common goal [7, 8]. In SoS scenarios, the cooperation between many systems offer emergent behaviour that cannot be provided by a constituent system in isolation and thus offers a promising approach to support Smart City systems interoperability.

While there has been significant research into SoS middleware [9, 10, 11] these works have not addressed the needs, and complexities of Smart Cities.

The goal of this paper is to propose SoS-centric middleware services to support the creation, composition, deployment, and execution of *SoS in Smart Cities* environments. The SoS-centric approach presented in this work aims at supporting interoperability among different systems operating in the city by providing a set of services and tools to allow, in a dynamic, transparent and scalable way, the management of SoS in the smart cities environments. Once developed, the proposed middleware will enable smart cities systems to talk to each other to provide richer and more interesting applications to solve city issues and provide better services to citizens.

SmartCities '16, December 12-16, 2016, Trento, Italy
 ISBN 978-1-4503-4667-2/16/12…$15.00
DOI: http://dx.doi.org/10.1145/3009912.3009917

2.BACKGROUND

This section presents some important concepts to allow the understanding of this paper. Section 2.1 discusses about interoperability. Section 2.2 presents SoS concepts.

2.1.Interoperability

According to IEEE [12], systems interoperability is "the capacity of two or more systems or components to exchange information and use the information exchanged". Interoperability involves much more than making systems communicate with each other, it requires significant effort to reach some degree of semantic compatibility and common interpretation among the interoperable systems, regarding exchanged data and messages [13]. Another definition, stated by [14], is that "interoperability is the ability of organizations and users to utilize the interconnectivity co-exhibited by the systems that serve them, in order to cooperate and collaborate, carry out business or operational transactions, and share and exchange goods and information".

Interoperability can be achieved in two ways [13]: (i) systems can natively include interoperability issues in their designs; or (ii) systems can be retrofitted. The first option is easier and less expensive. However, considering that smart cities systems usually are proprietary systems and that they are often owned by distinct organizations, in general, they are developed in an isolated way without considering interoperability in their design.

Conversely, while the second option (retrofitting) is a tractable task, it requires enormous effort since: (i) it affects many aspects of the existing systems (for instance, external APIs, user interface, use of standards and communication protocols, and so on), and (ii) it involves understanding a broad range of systems, which is a hard task for developers. Unfortunately, this second option is currently the only viable option to interoperate existing smart cities systems.

2.2. System-of-Systems (SoS)

SoS can be defined as a set of independent and heterogeneous *constituent systems* that interoperate to accomplish a global mission [15, 16]. This mission can only be achieved with the contribution of each system that constitutes the SoS, that is, it cannot be performed by only one of its systems. The collaboration among such constituent systems enables a SoS to offer new functionalities that cannot be provided by any of these systems working as individual entities, the so-called *emergent behaviour*. Besides emergent behaviour, there are other intrinsic characteristics that make SoS distinct from other distributed complex and large-scale systems, such as (i) the *operational and managerial independence* of constituent systems, which provide their own functionalities even when they do not cooperate within the scope of the SoS and can be managed independently from it, and (ii) the *evolutionary development of the SoS*, which may evolve over time to respond to changes on its execution environment, on the constituent systems, or on its own mission. Together, these characteristics have posed a set of challenges mainly related to the development, dynamicity, and evolution of SoS, thereby making traditional system engineering processes no longer suitable for constructing these systems [16].

2.2.1.Missions in the context of SoS

SoS missions are typically viewed as features or a set of tasks to be performed by constituent systems [17]. Such missions are categorized into two types: *individual missions* and *global missions*. Considering that in a SoS, each constituent system is independent from any other system, individual missions refer to the features or tasks performed by each constituent system isolated. Such missions are related to a specifically constituent system. In turn, global missions are related to new features and/or functionalities achieved by the integration of SoS's systems. Global missions are dependent of a set of constituent systems, and it is not possible to achieve global missions considering the contribution of only one constituent system.

Figure 1 presents a conceptual model of missions in SoS [17], showing the relationship among concepts related to missions in SoS. It is possible to observe that SoS is composed by many constituent systems. SoS accomplishes global missions, whereas constituent systems accomplish individual systems and contribute to global missions. Missions have tasks, priorities, and parameters. Moreover, emergent behaviours are those behaviours created by the cooperation between constituent systems of a SoS. Missions are indirectly related to such emergent behaviours.

2.2.2.Interoperability through SoS

According to the SoS concept, there is a strong relationship between SoS and interoperability since this is one of SoS main characteristics. Global missions are achieved by interoperability among constituent systems, resulting in emergent behaviours created by such interoperability.

Once a specific system participates in a SoS, it is necessary to ensure that the system can exchange and understand data, messages, and policies of the other constituents systems of the same SoS. This implies the need for models, protocols, services, and tools to support the development of systems interoperability through a SoS approach.

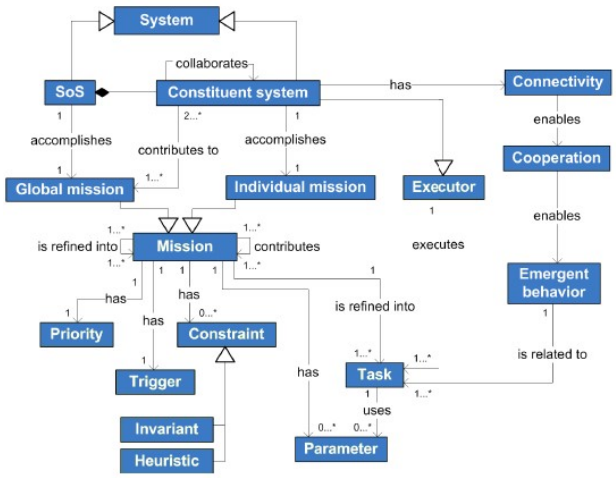

Figure 1. Relationship among SoS concepts [17]

3.MOTIVATING CASE STUDY

In order to better understand the issues that must be addressed when multiple independent systems are brought together to provide a new Smart City service or application, we have studied a motivational case study, composed of three distinct and independent systems, related to the process of waste collection and management. Typically, such systems are owned by different (public or private) organizations and have already been developed using distinct technologies. The purpose of this case study is to motivate the creation of some abstractions and services to facilitate the interoperability among systems in smart cities environments.

To better understand how to build a new service composed of multiple independent city systems, we explore three distinct approaches: (i) Non-interoperable systems – where existing systems are not capable to exchange data automatically. (ii) A loosely coupled *approach* - a naive solution to provide direct interoperability between systems using specific adapters for each pair of systems, allowing their interactions. (iii) *Smart City platform* – this approach uses some form of Smart City platform to act as an orchestration platform for the distinct city systems.

It should be noted that to facilitate the analysis, we have not used actual city systems, rather we have modelled and, where needed, built our own versions of existing systems.

3.1. An overview of the garbage management service in a non-interoperable approach

This subsection presents the three systems developed in this case study, in which each system is an isolated legacy system. It means that, in this first approach, there is no any type of interoperability among them.

System 1, owned by the City Hall department responsible for collecting the garbage produced in the whole city, is in charge of monitoring the quantity of waste inside each garbage container. Specifically, this system monitors the occupation level and the weight of waste inside the container. To achieve this functionality, each container has a balance weight and level sensors. All data (weight and occupation level) collected by the sensors are periodically sent, via wireless communication, to the system server. Thus, the server can store that data in the system database. It was developed in PHP and relates each garbage container to a unique identification, geographic location and the type of waste (e.g. plastic, organic, paper, glass, etc.) that it stores. Moreover, since each container stores only one specific type of waste, one collection point could be associated to more than one container.

System 2, called *Eco-Feedback system*, is responsible for calculating and presenting statistical information about the waste production in the city. Owned by a local non-governmental organization (NGO) focused on environmental conservation, this Java Web-based system provides data about the waste production considering many granularity levels. For example, data related to waste produced per capita, considering his/her neighbourhood, city region, whole city, etc. This system also reports the most produced waste type in the last week. The data processed by such a system is entered by a system operator of the NGO and provided weekly by the City Hall department responsible for collecting the garbage. This data is delivered in a spreadsheet document, resulting in a manual process to insert such data in the second system. Besides being a manual and error-prone strategy, this process makes real-time data visualization impossible. Moreover, the statistical information produced by the system is public and can be viewed by all citizens on the second system's Web page.

The last independent system (*System 3*) is responsible for managing the routes of the garbage trucks. Under the responsibility of the private company that provides garbage collecting service to the City Hall, this system was developed in the C++ programming language and it does not provide external communication with others systems. Its main function is to define garbage truck routes to collect the garbage from across the city. The system associates a route to a specific truck and each truck collects one or more types of waste or recyclable material. Truck routes are static since their definition is directly dependent on the manager's decision that creates the routes according to him/her feelings and experience. Thus, route definition is not based on real-time or statistical quantity of waste in each collection point. Once created, routes are printed and delivered to the truck drivers. This is not the most productive way to collect the garbage since the truck can visit sites without garbage to be collected or might be full before completing the route. Obviously, this is not a good strategy but it still used by many cities.

3.2 A loosely coupled approach to system integration

The systems previously described are completely independent and owned by distinct organizations. Moreover, they use different technologies and each one has very specific and well-defined functions. Although useful, some of those systems work with out-dated data since they work in an isolated way. Moreover, the data feeding process in the second and third systems is made in a manual way. These characteristics result in an error-prone environment since they can use out-dated data. Another important and negative aspect of this first approach is that, isolated, they cannot be integrated to the smart city ecosystem. This is an important drawback since the smart city platform could facilitate data processing, data analysis, data sharing, etc. To overcome this drawback, it was necessary to develop new software components (*bridges*) to achieve integration among the aforementioned systems, in which each bridge interconnects pair of systems. To achieve this, the bridge's developer has to consider and understand characteristics of both systems.

This new case study's approach included three bridges: (i) bridge to interconnect *System 1* and *System 2*, in which *System 1* provides, to *System 2*, data about the quantity of waste in each collection point; (ii) bridge that provide the same data to *System 3*; and (iii) bridge that provides statistical data from *System 2* to *System 3*. Each bridge has the responsibility of automatically delivering data from one system to another system. Moreover, *System 1* and *System 2* also provide REST APIs. This is necessary since such systems were developed using distinct technologies. Figure 2 presents such bridges and the REST APIs.

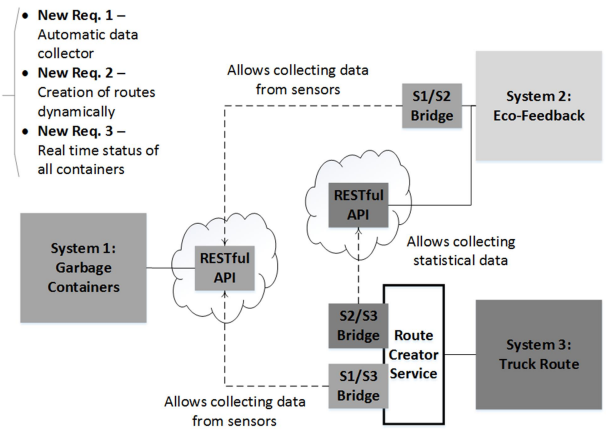

Figure 2. Systems directly integrated through APIs and bridges among them.

This new version of our case study directly implemented three new functionalities: (i) automatic data collection in *System 2* and *System 3*; (ii) dynamic creation of truck routes; and (iii) status information of each waste container in real time. It is important to mention that the creation of these new functionalities may not be

possible without the integration of those three systems. Thus, it means that system integration really can add important value for the environment they are engaged on. Although this solution is relatively simple to be developed, it is clear that there are some considerable difficulties in its development. For instance, it is necessary that the bridges' developers know some details of each integrated system. Other problem is the coupling between such systems, making reusability difficult to be reached. Such problems are better detailed in the Section 4.

3.3 Using a Smart City platform

A third approach to support interoperability between existing Smart City systems in order to deliver new services or applications is to leverage a Smart City platform. These platforms [1, 3, 4, 5] have been developed in recent years with the goal of making the development of new city services and application easier. An example of this is the CityHub platform [3] developed by one of the authors.

Integrating existing systems using a smart city platform allows the systems to use sophisticated abstractions provided by those platforms. However, integrating interoperable systems to a smart city environment is not a trivial task. In addition, based on our knowledge in the area, the platforms currently available do not fully address systems interoperability. Thus, it is necessary to include some source code in the application side to simulate any degree of interoperability.

For example, Figure 3 shows the CityHub platform which uses CKAN to manage Open Data (static data) and an IoT platform (WoTKiT [18]) to provide dynamic (sensors, etc.) data for smart cities applications. CityHub offers a state of the art solution for data integration in Smart Cities. If we were to use CityHub as a basis for an SoS approach to building our Smart Garbage application, it would require that these applications implement the requirements within their source code. This is shown in Figure 3 by circles inside applications cubes, which would increase the coupling of applications. However, this approach reduces the reusability since new requirements deployed in a specific application cannot be used by other applications. So, although this 3rd approach, exploiting a Smart City platform for example CityHub, does make it possible to build Smart City applications, it is still more complex and less flexible than our proposed approach of using a SoS-centric middleware.

4.LESSONS LEARNED

This section discusses some learned lessons from our experiences trying a direct integration or a middleware platform approach for building new services based on existing systems. More specifically, we discuss problems related to systems heterogeneity, lack of systems control by the integration developer, systems changes issues, complexity to develop applications and integration of those systems to smart cities environments.

Lack of systems control. In addition to the systems heterogeneity, typical systems deployed in the city are independent and owned by distinct organizations, as SoS constituent systems. Considering that the system control of each system is concentrated on its owner, the developers of the systems integration is strongly dependent of such systems owner. Thus, there is a clear need of a change in the approach in the direction of having well-defined standards and protocols to allow developers to integrate such systems in an easy way. Such an approach is not restricted to only provide systems APIs. Even more than that, it is needed to provide new ways to support the integration developer in the control of messages flows, constraints, missions, etc. However, that support cannot undermine the integrity and independence of such systems.

Change management. The organizations activities are dynamic in nature. Thus, systems must be prepared for changes and adaptations in the environment. However, in general, systems are not designed to reflect such dynamism, resulting in serious problems. For example, the solution illustrated in Figure 2 integrates three systems that were not prepared to interoperate. Thus, since such three systems are now integrated, any new change (the change in the requirements of a constituent system, the emergence of a new systems or the exchange of one system for an equivalent system) will require a large effort. For instance, in case of substitution of the company responsible for garbage collection for another company, it could be necessary to also substitute the system responsible for generating the dynamic routes for the system adopted by the new company. This type of environmental changes will require unexpected efforts to adapt: (i) bridges between the other two systems; (ii) features of the new system; (iii) technologies used by new systems, among others. Similarly, if the change occurs in any existing system, or if there is a need of inserting a new system into the environment, it will be necessary to create new bridges. The growth of this problem is proportional to the growth of the number of systems to be integrated and any changes will require a huge programming effort, increasing the costs involved.

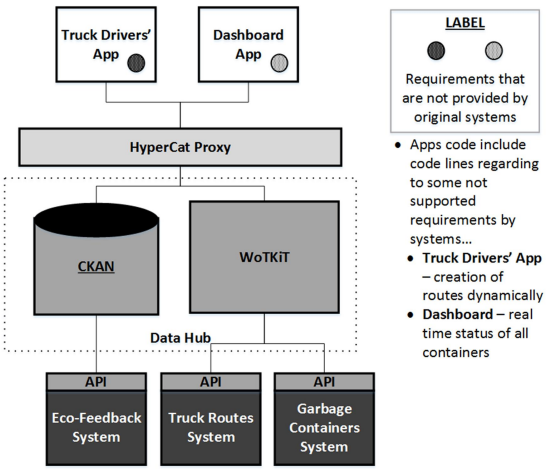

Figure 3. Requirements implemented directly by applications in a Smart City Environment.

High complexity when develop applications. Considering that one specific application needs to use data and requirements provided by many smart cities systems, the complexity in developing such application is also a challenge. The reason is because the application developer has to understand details of each system. The problem is not the need of knowing how to use the systems APIs, since nowadays many systems offers RESTful APIs. The real problems are how to: (i) control the order of calls for each system; (ii) know specific messages protocols adopted by each system; (iii) handle data from distinct systems, sometimes transforming data format from one system to another system; among others. Thus, it is possible to observe that these problems hamper the application development since the code necessary to integrate such system is embedded and coupled within

applications. Consequently, applications have a low level of reuse, high maintainability costs, and this problem grows in proportion to the number of systems.

The challenges described above may be solved through some specific middleware services and tools. They would be responsible for creating, managing, and deploying the SoS to integrate smart city systems. Moreover, middleware components will abstract away some very complicated tasks, for example, take the control of messages flowing among integrated systems. Although, as a first impression, current Web Service compositions solutions [19, 20] would be used to coordinate such messages flow, our understanding is that, alone, such solutions are not enough due to the full independence of smart cities systems. Such middleware services have to support a higher level of abstraction, considering messages flow, individual and global missions, emergent behaviours, priorities, among others SoS characteristics.

5. PROPOSED SOLUTION

Taking into account the drawbacks of the integration approach presented in Section 3.2 and Section 3.3, and the lessons learned (Section 4), this section presents a new abstraction layer and some middleware services to provide interoperability, based on the SoS approach, among smart city systems. Such solution can be integrated to smart cities platforms to provide an easier and reusable way to create and reuse SoS. This approach maintains the independence and goals (individual mission) of each constituent system, while providing the conditions to the emergence of new features and/or functionalities (global mission).

Concerning the global missions, it is important to mention that, unlike the directly integrated approach (Section 3.2), a SoS approach offers greater flexibility because new systems, once developed, are available for future composition into other SoS. Thus, the source code responsible for implementing global missions should be consumed by smart cities applications as a service. This strategy will improve the reusability of global missions. For instance, Section 3.2 presented three new functionalities of the integrated version (automatic data collect in systems 2 and system 3; truck routes dynamically created; and status of each wast container in real time). However, as showed in Figure 2, those functionalities were implemented by bridges and/or directly in the applications. In the SoS platform approach, such functionalities are implemented as a new system, decoupling functionalities from application-specific SoS.

Another advantage of this approach is the clear separation of responsibilities among constituent system developer and SoS developers. Taking into account that smart cities environments are composed of a huge number of systems, systems developers would be motivated to integrate their systems to the SoS platform since it will allow those systems be a constituent system of SoS.

5.1.Interoperability requirements

This subsection discusses about some requirements to allow an interoperability approach based on SoS.

(i)Integrating Systems into the Environment - In order to make it easier to integrate a system in the SoS the developer of system must: (i) implement an specific API to allow the communication between the system to be integrated and its broker. The proposed solution in this paper offers an abstract interface component to be easily extended by constituent system's developer; (ii) define the individual missions of his system; and (iii) register his system in the metadata repository of proposed solution.

(ii)SoS Creation - Integration developers can create a new SoS through a GUI and using systems' metadata stored in a metadata repository. In this phase, they can specify which are the SoS's constituent systems, SoS's global missions, SoS's business rules, SoS configuration, among others tasks. Each system can be a constituent system of many SoS and each SoS has, at least, two constituent systems.

(iii)SoS Management - Once created, all SoS are managed by the SoS platform. This activity is responsible for: (i) deploy SoS; (ii) make the constituent systems aware of the new SoS; (iii) update the metadata repository including SoS metadata; (iv) end the SoS.

(iv)SoS Deployment - Each SoS, according to its global missions, can interact (consume functionalities) of its constituent systems and can include some new business rules. A good example of this characteristic is the *Route Creator Service* presented in Figure 2, since this service includes some source code directly related to the systems integration. All new SoS have to be deployed in the environment as a new system. Thus, the SoS would be able to be consumed by applications.

(v)SoS Ending - When the SoS lifecycle ends, it is necessary to make the constituent systems aware about the end of SoS operation.

5.2.Architecture

This section proposes a middleware architecture that supports the interoperability among systems, in the context of smart cities, through a SoS approach. It also allows the creation, development, management and tear-down of a SoS. Figure 4 shows the proposed components integrated to the CityHub platform, smart cities systems, and the applications.

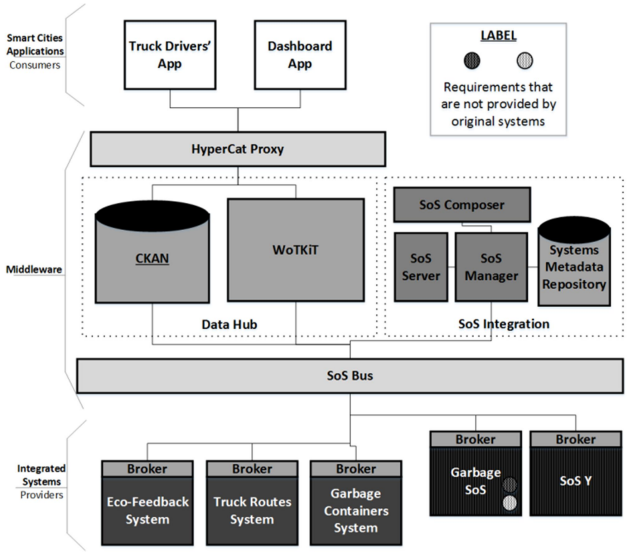

Figure 4. Proposed Architecture.

Each system to be integrated to the environment, must have its own *Broker* instance. Brokers are responsible for mediating the communication between the smart city system and the middleware. Internally, each broker includes sub-components to support the communication with brokers of other systems and with others middleware components. Brokers also have some sub-components to control the SoS execution flow in a distributed and choreographed way. Consequently, they have to be aware of all SoS in which the system is participating. Thus, each broker has its

own repository to store metadata (SoS's behaviours, global missions, flows, RESTful APIs, etc.) about the SoS. It is important to mention that even the SoS has a broker instance since each SoS can offer new requirements to be mediated by CityHub components, and consequently, those new requirements can be consumed by applications through CityHub components.

The SoS Bus defines protocols to allow the communication among brokers, *SoS Manager* and some CityHub components (*CKAN* and *WoTKiT*). *SoS Manager* is the component responsible for controlling the SoS creation, deployment and ending. It provides metadata of systems integrated to the platform for the *SoS Composer* GUI and deploys/withdraws SoS in the *SoS Server* as a new system, always updating brokers of concerned systems. *SoS Composer* is the GUI used by SoS developers to create and administrate SoS. SoS developers can semantically or syntactically describe SoS to be deployed in the environment. Finally, *Systems Metadata Repository* stores metadata regarding all constituent systems and SoS deployed in the platform environment. That repository stores, for instance, the endpoints of systems brokers, missions, among other information.

5.3.Next Steps

This work is still under development. Future work includes: (i) designing the SoS abstractions; (ii) designing and developing each component related to the SoS-centric middleware services; (iii) select technologies to describe missions execution flow, to communicate brokers with each other, to describe interfaces between brokers and respective systems, to describe interfaces between brokers and other smart cities platforms' components; and (iv) to develop case studies in order to validate and evaluate the SoS centric middleware services presented in this paper.

6.CONCLUSION

In Smart Cities environments it is expected that systems work with others, exchanging data and features. However, it is not easy to create conditions for an efficient exchange among those systems since they are not usually designed to be integrated. Thus, composing them to create a new complex system is a non-trivial task. Moreover, the divergence of adopted technologies and the need for knowledge about third-party systems complicate the already difficult task of developing such interoperability.

This paper proposed a SoS-centric middleware services to support the creation, deployment, execution and tear-down of SoS for Smart Cities environments in a dynamic, transparent and scalable way. The proposed services, once integrated to a Smart City platform, support interoperability among different systems operating in a city. Moreover, this paper also presented a motivational case study to highlight the issues that must be addressed when multiple independent systems are brought together to provide a new Smart City service or application.

7.ACKNOWLEDGEMENT

This research was partially supported by Brazilian's National Counsel of Technological and Scientific Development (CNpQ) and Smart Metropolis Project (IMD/UFRN/Brazil).

8.REFERENCES

1. Villanueva, F. J., Santofimia, M. J., Villa, D., Barba, J., and López, J. C. 2013. Civitas: The smart city middleware, from sensors to big data. In *Proc. of int.. conf. on IMIS*, 445-450.
2. Delicato, F., Pires, P., Batista, T., Cavalcante, E., Costa, B. and Barros, T. 2013. Towards an IoT ecosystem. In *Proceedings of the International Workshop on SESoS*. 25-28.
3. Lea, R. and Blackstock, M. 2014. CityHub: a cloud-based IoT platform for Smart Cities. In *International Conf. on Cloud Computing Technology and Science, IEEE*. 799-804.
4. Mitton, N., Papavassiliou, S., Puliafito, A. and Trivedi, K. 2012. Combining Cloud and sensors in a smart city environment. *EURASIP Journal on Wireless Communications and Networking*, 1–20.
5. Petrolo, R., Loscri, V., and Mitton, N. 2014. Towards a Smart City based on Cloud of Things. *WiMobCity - International ACM MobiHoc Workshop on Wireless and Mobile Technologies for Smart Cities*. Philadelphia, USA.
6. Grace, P., Bromberg, Y. D., Réveillere, L., and Blair, G. 2012. Overstar: An open approach to end-to-end middleware services in systems of systems. In *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing.* Berlin, 229-248.
7. Maier, M. 1996. Architecting Principles for SoS. In *Sixth annual International Symposium of the International Council on Systems Engineering*. Boston, MA, 567–574.
8. Jamshidi, M. 2011. *System of Systems Engineering: Innovations for the Twenty- First Century*. Vol 58.
9. Blair, G., Bromberg, Y., Coulson, G., Elkhatib, Y., Réveillere, L., Ribeiro, H., Riviére, E., and Taiani, F. 2015. *Holons:Towards a Systematic approach to composing SoS.*
10. Coulson, G., Blair, G., Elkhatib, Y., and Mauthe, A. 2015. The design of a generalised approach to the programming of SoS. In *WoWMoM.* 1-6.
11. Curry, E. 2012. System of systems information interoperability using a linked dataspace. In *7th International Conference on System of Systems Engineering.*101-106.
12. Dictionary of IEEE Standards Terms, Seventh Edition. 2000 New York, NY: IEEE.
13. Madni, A. M., and Sievers, M. 2014. Systems integration: Key perspectives, experiences, and challenges. In *International Council on Systems Engineering.* 37-51.
14. Mordecai, Y., and Dori, D. 2013. 6.5. 1 I5: A Model-Based Framework for Architecting SoS Interoperability, Interconnectivity, Interfacing, Integration, and Interaction. In *INCOSE International Symposium*. 1234-1255.
15. Billaud, S., Daclin, N., and Chapurlat, V. 2015. Interoperability as a key concept for the control and evolution of the System of Systems (SoS). In *International IFIP Working Conf. on Enterprise Interoperability*. 53-63.
16. Kazman, R., Schmid , K., Nielsen, C., and Klein, J. 2013. *Understanding Patterns for SoS Integration.* Technical Note.
17. Silva, E., Cavalcante, E., Batista, T., Oquendo, F., Delicato, F., and Pires, P. 2014. On the Characterization of Missions of SoS. In *Proceedings of the 2014 European Conference on Software Architecture Workshops, ACM.* 26.
18. Blackstock, M., and Lea, R. 2012. IoT mashups with the WoTKit. In *International Conference on the IoT.*159-166.
19. Pautasso, C. 2009. *Restful web service composition with BPEL for REST*. Data Knowledge Engineering. 851–866.
20. Hatzi, O., Vrakas, D., Nikolaidou, M., Bassiliades, N., Anagnostopoulos, D., and Vlahavas, I. 2012. An integrated approach to automated semantic web service composition through planning. *IEEE. 5*(3), 319-332.