A Bigraphical Multi-scale Modeling Methodology for System of Systems

Amal Gassara, Ismael Bouassida Rodriguez, Mohamed Jmaiel, Khalil Drira

A Bigraphical Multi-scale Modeling Methodology for System of Systems

Amal Gassara

ReDCAD, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia
amal.gassara@redcad.org

Ismael Bouassida Rodriguez

ReDCAD, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia
Digital Research Center of Sfax, B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia

Mohamed Jmaiel

ReDCAD, University of Sfax, B.P. 1173, 3038 Sfax, Tunisia
Digital Research Center of Sfax, B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia

Khalil Drira

CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France

Abstract

In this paper, we present a multi-scale modeling methodology for software System of Systems (SoS) using the formal technique of Bigraphical Reactive System. This methodology provides a correct by design approach ensuring the correctness of the SoS architectures. A first scale is defined by the designer. Then, it is refined by successively adding lower scale details. The transition between scales is implemented following a rule-oriented refinement process. The executed rules respect the system constraints ensuring, in this way, the correctness of the obtained scale architectures. Moreover, we address the dynamic aspect of SoS by providing model-based rules of reconfiguration actions. We illustrate our approach with a Smart Buildings case study.

I. Introduction

The increasing integration of software, hardware and network has raised a new class of software systems, the so-called System of Systems (SoS) [Maier, 1998]. A SoS is composed of large scale integrated systems that are heterogeneous and independently operable on their own, but are networked together for a common goal. Systems of Systems are increasingly involved in different areas [Zhou et al., 2011] like emergency coordination and crisis management, health care, smart cities, etc.

These systems are characterized by geographic distribution, operational and managerial independence of their elements, evolutionary development, and emergent behavior. These characteristics have therefore brought new challenges to software development including software architecture design. In fact, a software architecture represents the structure of a system which comprises software components, their externally visible properties, and the relationships among them [Garlan, 2003]. Hence, by providing a high-level model of the

system architecture, it allows to understand the system in a simple way and to reason about its key properties.

However, the complexity of SoSs makes their design more difficult especially in ensuring the correctness of their architectures. By correct, we mean that architectures respect a set of functional requirements like structural constraints related to the hierarchy of their constituents and the connections between them. Therefore, an existing open issue is how to facilitate the modeling of SoS architectures and how to represent them rigourously in order to guarantee their correctness.

Another major issue is to address the dynamic aspect of such systems by ensuring reconfigurations. A reconfiguration describes how an architecture can evolve by adding or removing not only components and connections inside a system but also the containing systems themselves considered as larger components in their own.

In recent years, some research activities like [Bryans et al., 2013, Khlif et al., 2014] addressed the description of SoS software architectures. But they provide informal description and can be extended with a solid formal foundation to ensure the correctness of the designed architectures. Other approaches [Petitdemange et al., 2015, Riddle, 2012, dan,] were proposed including formal models and techniques. However, these approaches do not provide a solution for facilitating the modeling of SoS architectures and mastering their complexity. They addressed the dynamic aspect of such systems by focusing only on correctness by evolution and not on correctness by design. Whereas, our approach ensures in addition the correctness of the initial architecture.

In this direction, the main goal of this work is to provide a rigourous solution for SoS modeling based on the formal technique of Bigraphical Reactive Systems (BRS) [Milner, 2009] with an inspiring vision from multi-scale modeling [Ingram et al., 2004, Gassara et al., 2013, Gassara et al., 2015, Khlif et al., 2014].

Therefore, we proposed B3MS, a Bigraphical Multi-scale Modeling Methodology for System of Systems. This formal methodology aims, first, to help the designer to model correct SoS architectures. Instead of modeling the whole SoS and verifying it with respect to the defined constraints, we rather propose a correct by design approach using a multi-scale modeling. Actually, multi-scale modeling is a suitable solution since it is based on a refinement process. Following B3MS, a coarse grained scale is defined by the designer. Then, it is refined automatically by successively adding lower scale details until detail goals are satisfied. This refinement is ensured by applying specific rules that comply with the structural constraints of ths system ensuring, in this way, the correctness of the obtained scale architectures.

Moreover, we address the dynamic aspect of SoS by giving some reconfiguration meta-rules which help the designer to model reconfigurations. These meta-rules are modeled as BRS and encompass the essential SoS reconfigurations such as adding, removing and replacing system constituents. In fact, we envision that modeling SoS reconfigurations can benefit, on the one hand, from multi-scale modeling since there is a need to work sometimes at coarse-grained levels and other times at fine-grained levels. On the other hand, modeling these reconfigurations benefits from the expressive power and visual representation of BRS to describe architectures and reconfiguration. It benefits also from the ability of BRS to abstract the constituent systems via parameters. Hence, the difficulty of dealing with SoS reconfigurations can be reduced by focusing on coarse-grained level without going into details of fine-grained levels.

The remainder of this paper is organized as follows. We present in Section II some research activities dealing with modeling SoS architectures. Then, in Section III, we present B3MS, our proposed modeling methodology for SoS that is illustrated by the Smart Buildings case study in Section IV. Finally, Sec-

tion V concludes this paper and presents future work.

II. Related work

Recent systematic literature reviews [Nakagawa et al., 2013, Guessi et al., 2015b] on the description of SoS architectures presented the use of some architecture description languages (ADLs) such as UML (Unified Modeling Language), SysML (System Modeling Language), CML (COMPASS Modeling Language), and X-UNITY. So, we identified some research activities that are based on these ADLs. For instance, Khlif et al. [Khlif et al., 2014] proposed a multi-scale modeling approach for SoS architecture description using UML and SysML notations. They ensure the description of SoS architectures (their constituents and the connections between them). To do so, they adopt a rule-oriented description technique to manage the refinement process as a model transformation between the coarse-grain and the fine-grain descriptions. Moreover, Bryans et al. [Bryans et al., 2013] used SysML to specify interfaces among the constituents and then they enriched such a specification with CML to specify contracts in SoS software. CML [Woodcock et al., 2012] is specifically designed for SoS modeling and analysis. It is a collection of process definitions and interact with the environment via synchronous communications. The use of interface descriptions enables the specification of pre and post conditions for operations.

However, we can notice that these two studies support only the static aspect of SoS and do not address dynamic aspect. This is explained by the fact that ADLs lack evolution feature of SoS, as it is claimed by Guessi et al. [Guessi et al., 2015a] after assessing fours ADLs used for SoS architecture description. Actually, SoS architectures should support dynamic evolution by adding new constituents and connections or removing existing ones. Moreover, despite the advantages of semi-formal representation, mainly with regard to comprehension, formal methods are interesting since considering that many SoS can address critical domains.

In this direction, Flavio Oquendo and Axel Legay [Oquendo and Legay, 2015] defined SosADL, a novel formal ADL which is an evolution of Π-ADL, used to describe static and dynamic architectural specifications. It extends Π-ADL with new architectural concepts and notations for SoS. SosADL allows to declare a set of architectural constraints which are solved at runtime in order to construct architectures. Using SosADL, Petitdemange et al. [Petitdemange et al., 2015] proposed an approach to describe SoS architectures and to manage their reconfigurations using the example of a fire emergency system. They proposed a pattern-based approach that provides a set of reconfiguration patterns in order to assist dynamic reconfigurations. According to these patterns, reconfigurations can be implemented and applied at runtime, while maintaining the consistency in the SoS.

Based on formal models, we identified also two major European projects (COMPASS [Riddle, 2012] and DANSE [dan,]). DANSE [dan,] proposed a methodology for the development of SoS architectures based on architectural patterns. This methodology starts with an initial SoS model that is created using the Model SoS solution method. Then, this model is changed to encompass a pattern that offers improvement. Through simulation and statistical model checking, the changed SoS model is tested to conform the SoS goals while meeting the desired evolutionary performance. To deal with dynamicity and evolution, DANSE abstracted models from SoS characteristics and used a contract approach to define semantics on behaviors, evolutions and structural parts of SoS. COMPASS [Riddle, 2012] aims to develop a modelling framework for SoS architectures and to provide a formal semantic foundation to support analysis of SoS models. This framework provides different levels of description, starting with SysML that is easy for stakeholders to understand. SysML is linked to CML

by extending it with SoS-specific features to describe the assumptions and the guarantees of constituent systems. This extended SysML can then be readily processed by static analysis tools including theorem provers and model checkers, allowing automated detection of inconsistencies, and of potential deviation from contract conformance.

Although these latter work are based on formal techniques, the correctness of architectures is based on verification using model checkers which would cause computational problems due to the complexity of SoS architectures. Whereas, in our work, we ensure SoS correctness by construction instead of verification.

Furthermore, these approaches do not provide a methodology that facilitates the design of SoS architectures and masters their complexity. In this context, Clark [Clark, 2009] considered that SoS consists of elements that are traditional systems and then classical modeling methodologies can be applied. Indeed, he used two Building Block and V-Model to model them. The Building Block method demonstrates that a SoS could be decomposed into subsystems until every subsystem becomes an individuals system, and then the system engineering methods can be applied. The V-Model method describes the SoS life cycle. It can be used to each subsystem until every subsystem is validated according to its requirements. However, using these classical methods, essential characteristics of SoS such as communication are ignored [Zhou et al., 2011]. This appeals to develop a more SoS-specific methodology. In this direction, we aim in our work to provide a methodology for SoS. We adopt a multi-scale modeling approach which supports a continuous and a coherent architecture description from the coarse grained level down to the fine grained level. This approach facilitates the design of such system architectures through a refinement process as well as ensures their correctness by construction instead of verification.

Also, Valerdi et al. [Valerdi and Lane, 2004] affirmed that to understand the SoS model, one can view the SoS as a set of systems, with the systems at each level comprised of some combination of other systems and system elements. Much of the total development effort for this type of system can be estimated using current system engineering and software development models. So, Valerdi et al. proposed COSOSIMO (Constructive System-of-Systems Integration Cost Model) that is a newest addition to the Constructive Cost Model suite (COCOMO) [Madachy, 2009]. COCOMO is designed to help users estimate software engineering effort for requirements analysis, design, construction, and verification and validation at the software configuration item level. However, COSOSIMO is designed to estimate the SoS architecture definition and integration effort for SoSs. Comparing with our approach, this work focuses on estimating the cost of modeling and integrating SoS. However, our approach focuses on providing a methodology of describing SoS architectures ensuring their correctness.

III. B3MS: a Bigraphical Multi-scale Modeling Methodology for SoS

We aim, in our work, to model correct SoS architectures that respect a set of structural constraints. To ensure correctness, it is necessary to study the consistency of a system at a given time, more precisely, its conformance to an architectural style. An architectural style defines a vocabulary of component and connector types, and some constraints on how they can be combined [Garlan, 2003]. So, an architecture can be considered as correct if it is an instance of its architectural style (i.e., it uses the defined component types and it preserves all the defined constraints).

B3MS, our proposed formal methodology, supports the correct modeling of the static aspect as well as the dynamic aspect of SoS architectures.

i. Modeling the static aspect of SoS

Modeling the static aspect of SoS is based on a multi-scale modeling approach. So, we follow the three steps of multi-scale modeling [Ingram et al., 2004]:

i.1 Step 1: Scales identification

The first step of multi-scale modeling is identifying and selecting scales. The architecture of a SoS is characterized by a hierarchical description of systems containing physical, hardware and software components. So, we define the relation between scales by a hierarchical decomposition. In other words, a finer scale represents the constituent systems of the previous scale systems:

- Scale i: represents coarse grained entities, such $i \in [0, n]$ where n corresponds to the level of composition (i.e., when it is impossible to have more decomposition).
- Scale $i + 1$: represents the sub-entities of scale i entities.
- Scale n: describes the communication entities deployed in each hardware component and how they are connected. We used the publish/subscribe paradigm [Meier and Cahill, 2005]. This communication model allows the interconnection of components loosely coupled, in a synchronous or asynchronous mode. It offers three types of entities: event producers (EP), event consumers (EC) and channel managers (CM). For every determined session, a CM is created in order to manage, store and deliver exchanged data flow between multiple producers and consumers. Actually, the EPs and ECs can be connected to CMs, but they can not be directly interconnected. The EPs send data to the CM to which they are connected. The CM returns a copy of the received data to all ECs connected to it.
  The main advantage of the publish/subscribe is the opportunity for better scalability than traditional client-server pattern through message caching and network-based routing. Moreover, publishers are loosely coupled to subscribers and they operate independently of each other.

i.2 Step 2: Submodel identification

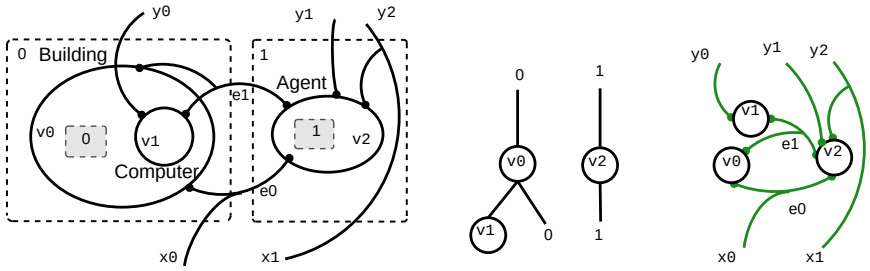
The second step of multi-scale modeling is identifying the appropriate submodels that will be adopted at each scale.

Actually, correct architectural design motivates the need for suitable formal language to avoid ambiguities. This formal language should (1) be able to describe systems, constituent systems as well as software and physical components. It should (2) emphasize both hierarchy and connectivity of constituent systems and components. It should also (3) provide information on both static and dynamic aspect of the system. We have noticed that Bigraphs and BRS [Milner, 2009] are the most appropriate languages that support these requirements. Moreover, Chang and al. [Chang et al., 2007] recommended that BRSs are suitable for describing architectural styles. Indeed, bigraphs are equipped with a sorting logic that allows to discipline bigraphs with some constraints. In our methodology, we used this language as a submodel for all scales.

A BRS consists of a set of bigraphs and a set of reaction rules that may be applied to rewrite these bigraphs. In the following, we give an overview of bigraphs, reaction rules and their sorting logic.

(a) Bigraphs A bigraph consists of hyperedges and nodes which can be nested (for example in Figure 1(a), v_1 is nested in v_0). Each hyperedge can connect many nodes via ports. For example, v_0, v_1 and v_2 are joined by the hyperedge e_1 in Figure 1(a) where ports are represented by black dots. A bigraph combines two graphical structures -a *place graph* and a *link graph*- based on the same node set, hence the term bigraph.

The *place graph* (cf. Figure 1(b)) is a hierarchical tree that describes the locality of



(a) A bigraph G

(b) *place* and *link* graphs

Figure 1: *A bigraph G and its graphs*

the nodes. In this graph, trees are rooted by regions represented by dashed rectangles (cf. Figure 1(b)). There can also be sites, represented as grey rectangles (cf. Figure 1(b)). A site is a hole that can host new nodes.

Whereas the *link graph* (cf. Figure 1(b)) is a hypergraph that describes the connectivity of nodes. Within this graph, there can be outer names like y_0 and inner names like x_0 represented as open links (cf. Figure 1). These names give bigraphs the possibility to be composed by joining the inner names of one bigraph with the corresponding outer names of another one.

Each node in the bigraph is assigned a control. Controls (*Building*, *Computer* and *Agent* in the bigraph G in Figure 1(a)) indicate the node type and the node ports' number. The set of controls forms the so called signature. We use the notation "X-node", which means a node that has been assigned the control X.

(b) Reaction Rules A reaction rule consists of two bigraphs: a *Redex* R (the pattern to be changed) and a *Reactum* R' (the changed pattern). The application of the rule consists in identifying the image of R in a bigraph and replacing it by the corresponding R'. For example, in Figure 2(b), the rule allows to add a floor (*Floor*-node) in a building (*Building*-node). The site (grey rectangle) in the *Redex* represents all other possible occupants of the *Bluiding*-node which are unchanged after applying this rule.

(c) Sorting logic Bigraphs can be associated with a sorting logic which consists of a set of sorts (i.e., types). So, all nodes and/or ports can be then enriched with such a sort. Further, a sorting stipulates some conditions on bigraphs enabling, thus, to restrict our attention to the bigraphs which satisfy these conditions and outlaw the others.

Definition III.1. (Sorting [Milner, 2009]). A sorting $\sum = (\Theta, \mathcal{K}, \Phi)$ has:
Θ: a set of sorts
$\mathcal{K} = \{K_1 : \theta_1, K_2 : \theta_2, ..\}$: a signature which is $\sum$-sorted over Θ (i.e., assigning a sort θ_i to each control K_i)
Φ: a formation rule defining conditions on the structure of nodes.

As an example of a formation rule, we can cite the following: "nodes having the sort θ_2 should be children of nodes having the sort θ_1"

i.3 Step 3: Multi-scale model construction

In order to construct a multi-scale model, our methodology adopts the top-down strategy. This latter helps to follow a correct by design approach based on our formal language (i.e., BRS).

Instead of modeling the whole SoS architecture and verifying it with respect to the defined system constraints (i.e., its conformance to its architectural style), we rather ensure the correctness of each scale architecture obtained by a refinement process.

Therefore, at first, the designer defines the architectural style by identifying the different components types (i.e., locations, devices, hosts and constituent systems, etc.) as well as their structural constraints (i.e., hierarchical and connection constraints). Then, he/she defines the first scale architecture. After that, this architecture is refined automatically by adding the next scale entities. The obtained architecture is refined in turn until reaching the last scale (i.e., where all system entities are defined).

With a BRS, a scale architecture is represented as a bigraph where nodes represent systems, constituent systems or locations, etc. and nesting structure represents their hierarchy. Whereas the hyperedges represent connections between the CSs. A hyperedge connects a set of nodes via ports where the ports represent the interfaces of CSs and the hyperedge means that these CSs belong to the same system and communicate together.

Regarding the refinement process, it is performed by applying specific rules that should respect the defined architectural style (or system constraints) ensuring in this way the correctness of the obtained scale architectures. To do so, we proposed some types of rules (i.e., denoted as meta-rules) ensuring this refinement.

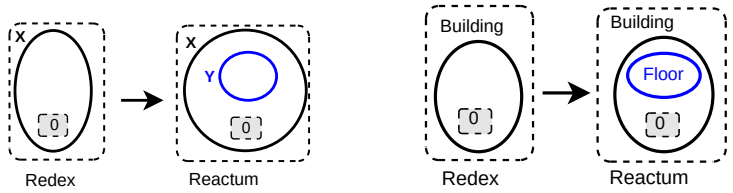
Meta-rule We look upon our meta-rules as generic templates from which reaction rules can be automatically instantiated. Our meta-rules allow to parametrize the node controls and ports for a given reaction rule as the case may be. Our approach supports the instantiation of meta-rules in order to guarantee the construction of correct rules.

In order to construct an SoS architecture, we proposed two meta-rules:

- The nesting meta-rule $nest(Y, X)$ depicted in Figure 2(a) encapsulates reaction rules allowing to add a nested node in a given node. The parameters X (i.e., the control of the nesting node) and Y (i.e., the control of the nested node) are passed to the reaction rule that will be constructed. The site (grey rectangle) in the *Redex* represents all other possible nodes of the X-node which will be unchanged after the application of the rule. When we pass $X = Building$ and $Y = Floor$, to the meta-reaction rule, we get the reaction rule of Figure 2(b).
- The connecting meta-rule $connect(s, nb)$: it allows to connect constituent systems belonging to the same system. The input parameters are the system name (s) and the number of constituent systems (nb). As depicted in Figure 3(a), this meta-rule consists in looking for nb CS-nodes marked with the s-node (marking constituent systems belonging to the system s) and connecting them with the same hyperedge. This meta-rule can be instantiated (for example $connect(s1, 2)$) to give the rule of Figure 3(b), that allows to connect two constituents of the system $s1$.

Furthermore, we proposed five meta-rules for enhancing the SoS architecture with the publish/subscribe communication model:

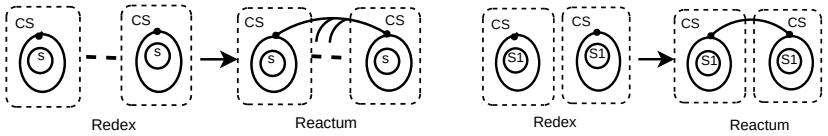
- The $addEPEC(s)$ meta-rule, depicted in Figure 4(a), encapsulates reaction rules allowing to add a pair of EP and EC in a CS that belongs to the system s.
- The $addCM(s)$ meta-rule, depicted in Figure 4(b), encapsulates reaction rules allowing to add a CM in a CS that belongs to the system s.
- The $linkEP(s)$ encapsulates reaction rules allowing to link an EP that belongs to the system s to its corresponding CM. As depicted in Figure 5(a), the *Redex* consists in finding an EP and a CM that belong to the same system (marked by an s-node) and then connecting them via the port ep.
- The $linkEC(s)$ encapsulates reaction rules allowing to link an EC that belongs to the system s to its corresponding CM. As depicted in Figure 5(b), the *Redex* consists in finding an EC and a CM that belong to the same system (marked by an s-node) and then connecting them via the port ec.



(a) The nesting meta-rule $nest(Y, X)$

(b) The $nest(Floor, Building)$ rule

Figure 2: *The nesting meta-rule and examples of its instantiated rules*



(a) The *connecting* meta-rule $connect(s, nb)$

(b) The $connect(s1, 2)$ rule

Figure 3: *The connecting meta-rule and one example of its instantiated rules*

Correctness by construction To efficiently tackle the correctness of architectures, B3MS follows a correct by construction approach. This approach is based on this idea: if the initial architecture conforms to a given architectural style and the changes of this architecture (refinements) preserve the constraints of this architectural style, by induction, the evolving architecture would conform to this architectural style, too.

Since we use a bigraph to describe one scale architecture, and reaction rules to describe the refinement of the architectures, we argue the above idea based on the conformance theorem proven by Chang et al. [Chang et al., 2007]:

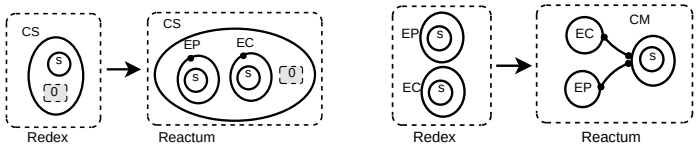
Conformance theorem *The changing bigraphs always preserve the constraints defined by $\sum$-sorted BRS if the initial bigraph and reaction rules do.*

Hence, the refinement process is performed on a correct scale architecture by applying the appropriate rules that comply with the defined constraints, ensuring in this way the correctness of the next scale architectures. These rules are instantiated from our proposed meta-rules. So, we should guarantee that the right rule is applied and that it is correct, too.

To do so, we proposed an algorithm that defines tactics for prescribing the sequence in which the meta-rules should be instantiated and applied. These tactics comply with the hierarchy of nodes defined in the architectural style.

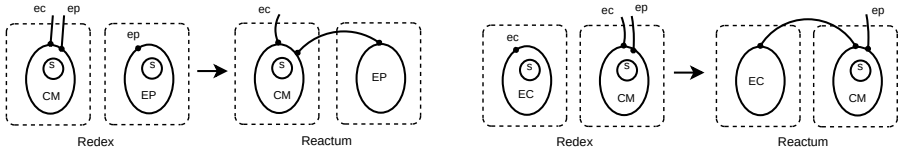
Moreover, we identified conditions on instantiating the meta-rules in order to obtain only correct rules. For this reason, the $nest(Y, X)$ meta-rule is instantiated according to the conditions on the nesting of nodes defined in the architectural style. Hence, we ensure that all instantiated rules comply with the constraints. For example, if the condition is : "*Floor*-node in *Building*-node", then we instantiate the meta-rule with the parameters $Y = Floor$ and $X = Building$.

Regarding the $connect(s, nb)$ meta-rule, it complies with the linking constraints defined in the architectural style. It allows to connect only nodes belonging to the same system. So, it will be instantiated for all defined systems in the SoS. Likewise, the meta-rules for adding and connecting publish/subscribe components are defined in a way that they comply with the constraints defined in the architectural style. Hence, all instantiated rules are corrects, too.



(a) $addEPEC(s)$ meta-rule (b) $addCM(s)$ meta-rule

Figure 4: *The meta-rules for adding publish/subscribe components*



(a) The $linkEP(s)$ meta-rule (b) The $linkEC(s)$ meta-rule

Figure 5: *The meta-rules for linking publish/subscribe components*

ii. Modeling the SoS reconfigurations

We address the dynamic aspect of SoS by modeling reconfigurations. A reconfiguration describes how an architecture can evolve by adding or removing not only components and connections inside a system but also the containing systems themselves considered as larger components in their own. The difficulty of dealing with such reconfigurations can be reduced by focusing on coarse-grained level without going into details of fine-grained levels. Hence, the modeling of SoS reconfigurations can benefit from multi-scale modeling. Moreover, it benefits from the ability of BRS to abstract the constituent components via parameters. This combination of multi-scale modeling and BRS offers an intuitive and a formal way to describe the dynamic aspect of SoS architectures.

While SoS architectures are represented by Bigraphs, we model reconfigurations via reaction rules. Therefore, we define a set of meta-rules which allows a SoS to handle communication requirement changes such as adding, removing or replacing constituent systems [Selberg and Austin, 2008]. These meta-rules are defined as follows:

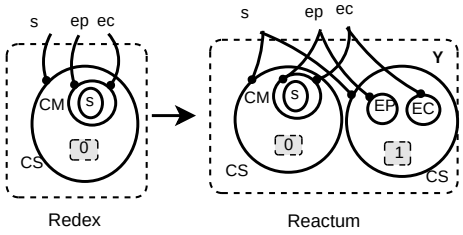


Figure 6: *The addCS reconfiguration rule*

Adding a constituent system We defined the meta-rule *addCS(s, Y)* that allows to add a constituent system having the control Y (i.e., the node type) to a given system (s). This reconfiguration is denoted in Figure 6. The *Redex* consists in finding a *CS*-node that belongs to the system s (containing the CM marked with an s-node). Afterwards, the *Reactum* consists in adding another *CS*-node and connecting it to the same hyperedge as the first *CS*-node. Then, it allows to add EP and EC components in the new *CS* and connecting them to the CM through the port *ep* and *ec*, respectively. The site numbered 1 enables the new *CS*-node to host its owns constituent systems.

Removing a constituent system We defined the meta-rule *removeCS(cs, s)* that allows to remove a constituent system (having the id *cs*) from a given system (s). This reconfiguration

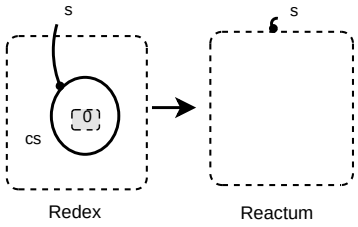


Figure 7: *The removeCS reconfiguration rule*

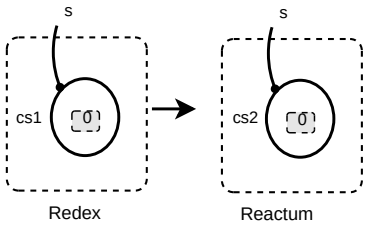


Figure 8: *The replaceCS reconfiguration rule*

(denoted in Figure 7) removes likewise all the constituent systems of *cs* that are abstracted via the site 0. Here, we highlight the advantage of BRS by avoiding the dealt with lower details.

Replacing a constituent system We defined the *replaceCS(cs1, cs2)* meta-rule that allows to replace a constituent system *cs*1 with another one *cs*2 from a given system. This meta-rule, denoted in Figure 8, consists in removing *cs*1. Then, adding *cs*2 and connecting it to the system. The constituents of *cs*1 that ensure the communication will be moved to *cs*2 after applying the reconfiguration.

IV. Case study: Smart Buildings

Let present our proposal by looking at the example of Smart Buildings.

i. Smart Buildings

In Smart Cities, many systems are interconnected such as power plants, solar farms, smart buildings, hospitals, etc. Most of these systems are categorized as SoS [Pérez et al., 2013]. To illustrate B3MS, we focus on Smart Buildings. Such buildings aim at improving the life quality of their occupants and at enhancing their comfort levels with the aim of setting them free of manual control of different tasks, while considering energy conservation whenever possible. For instance, various systems and appliances are turned off in vacant rooms or when they are useless.

As showed in Figure 9, Building Automation Systems are managed and controlled by a City Control Unit. Each system may consist of HVAC systems (i.e., Heating, Ventilation and Air-Conditioning), Automated Control of Lighting (ACL), security or surveillance systems, etc. installed in building floors. In our case study, we take the example of the two first systems. These floors contain rooms and each room can be equipped with heterogeneous devices (sensors like thermometer, presence sensor, light sensor, etc. and actuators like air conditioner, lamp, etc). These devices are connected together and communicate to ensure an intelligent building control. Sensors collect information such as rooms lighting, human presence, temperature, etc. The Building Control Unit receives and analyses these information in order to adaptively configure the devices.

ii. Modeling the static aspect of Smart Buildings

Step 1: Scales identification

In a city, the Smart Buildings are characterized as a hierarchy of systems that are located in different spatial scales like city, buildings, floors, rooms, and so on (Figure 9). While each constituent system must be placed in a specific location, locations in turn may have a hierarchical description. So, the Smart Buildings architecture describes systems, their constituents and their locations. According to this architecture, we define the following scales:

- City scale: this scale contains the Smart Buildings system which is managed by the City Control Unit to control all the buildings in the city. It includes also city

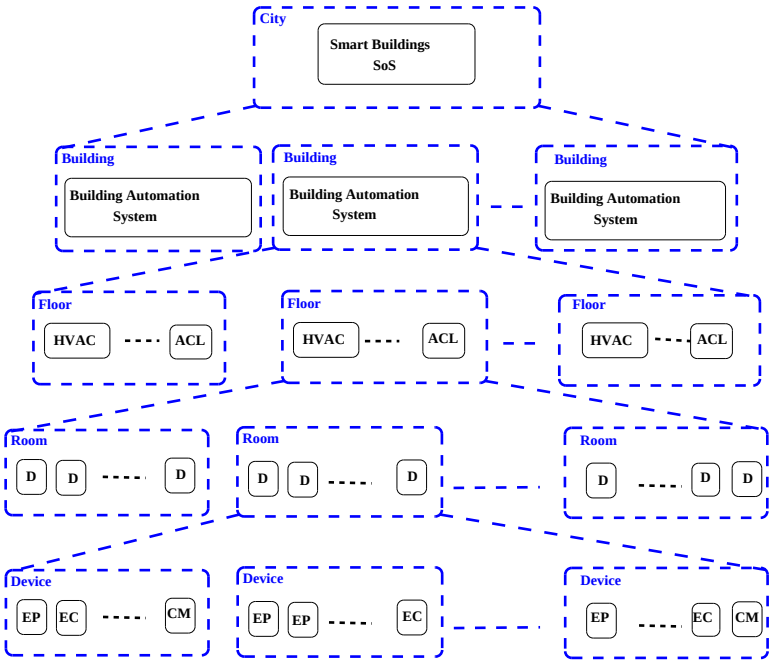


Figure 9: *The hierarchy of the Smart Buildings as a SoS*

resources such as buildings, homes, etc.

- Building scale: it contains the building floors and the Building Automation System which is managed by a Control Unit.
- Floor scale: it contains rooms as well as different systems such as HVAC systems and Automated Control of Lighting in our case study.
- Room scale: it includes equipped devices like sensors and actuators. In fact, the HVAC system is composed of thermometers and air conditioners connected to the Building Control Unit. Whereas, the Automated Control of Lighting is composed of lamps, presence sensors and light sensors connected to the Building Control Unit.
- Device scale: it describes the communication entities deployed in each device and how they are connected.

Step 2: Submodel identification

This step is already defined by our methodology. We identified BRS as a submodel for all scales (See Section i.2).

Step 3: Multi-scale model construction

Following our methodology, the designer should start by identifying the node controls (CCU representing the City Control Unit, BCU representing a Building Control Unit, Building representing a building, Floor representing a floor, Room representing a room, D representing a device, EP representing an event producer, EC representing an event consumer and CM representing a channel manager).

Then, the designer defines the structural constraints that should be respected while instantiating the meta-rules. For smart Buildings, he/she defines the following information and constraints:

- A building node can contain only a Building Control Unit node and floor nodes.
- A floor node can contain only room nodes.
- A room node can contain only device nodes.
- Smart Buildings System is composed of Building Control Units and the City Control Unit.

- HVAC System is composed of devices and their corresponding Building Control Unit.
- Automated Light Control system is composed of devices and their corresponding Building Control Unit.

Based on the publish/subscribe paradigm, we identified likewise a set of structural constraints:

- EPs and ECs communicate only via CM and all EPs and ECs belonging to the same system are connected to the same CM.
- Each system has a CM that is placed in a device belonging to this system.
- Each constituent system contains a pair of EP and EC.

Moreover, the designer should specify some constraints on the number of nodes. For example, each building has two floors. Each floor contains three rooms and is equipped with a HVAC system and an ACL system. Each room in turn is equipped by two devices belonging to the HVAC system and three devices belonging to the ACL system.

Since the architectural style of Smart Buildings describes physical entities including locations, constituent systems and publish/subscribe entities, we distinguish three main kinds of nodes: *Loc* representing locations, *CS* representing constituent systems and *PS* representing publish/subscribe entities.

Based on these requirements, let's present the corresponding sorting that defines this architectural style.

Definition IV.1 (Sorting for Smart Buildings)**.** $\sum_{SC} = (\Omega, \mathcal{K}, \Phi)$ where:
$\Omega = \{Loc, CS, PS\}$
$\mathcal{K}$ = {*Building*:(*Loc*,0),*Floor*:(*Loc*,0), *Room*:(*Loc*,0),*CCU*:(*CS*,4),*BCU*:(*CS*,4), *D*:(*CS*,1), *EP*:(*PS*,1), *EC*:(*PS*,1), *CM*:(*PS*,2)}.
Φ = {*BCU* in *Building*, *Floor* in *Building*, *Room* in *Floor*, *D* in *Room*, *CS* nodes marked with the same node should be connected with the same hyperedge, a pair of EP and EC in a *CS*, one CM in a *CS* that belongs to its system, EP must be connected to the CM of its system through the port *ep*, EC must be connected to the CM of its system through the port *ec*}.

The signature of this sorting defines the controls *Building*, *Floor* and *Room* which are assigned to the sort *Loc* and have 0 ports as well as the controls *CCU*, *BCU* having 4 ports and *D* having 1 port. These latter controls are assigned to the sort *CS*. Also, it defines the controls *EP* and *EC* having 1 port and *CM* having 2 ports. These controls are assigned to the sort *PS*.

Finally, the designer defines the first scale model (i.e., city scale). After that, the construction of the Smart Buildings architecture begins as highlighted in the following:

City scale The first scale (i.e., city scale), given by the designer, includes buildings and the City Control Unit. The first part of Figure 10 shows the corresponding bigraphical model, where *CCU*-node represents the City Control Unit and *Building*-nodes represent buildings. Here, we present only three buildings in order to obtain a simple and understandable model.

Transition from city scale to building scale The city scale model is refined to obtain the next scale model (i.e., building scale). This refinement consists in adding building floors. To do so, we instantiated the nesting meta-rule to have the rule enabling to add a floor in the building $nest(Floor, Building)$ (cf. Figure 2(b)). Therefore, this rule can be applied several times in order to add many floors residing in the same building.

Moreover, we instantiated this meta-rule to have the rule enabling to add a Building Control Unit in the building $nest(BCU, Building)$. Then, we instantiated the connecting meta-rule to obtain $connect(sb, 4)$ rule. This rule allows to connect Building Control Units to the City Control Unit. These control units contain *sb*-nodes in order to mark that they belong to the smart buildings system (marks are not presented in the model).

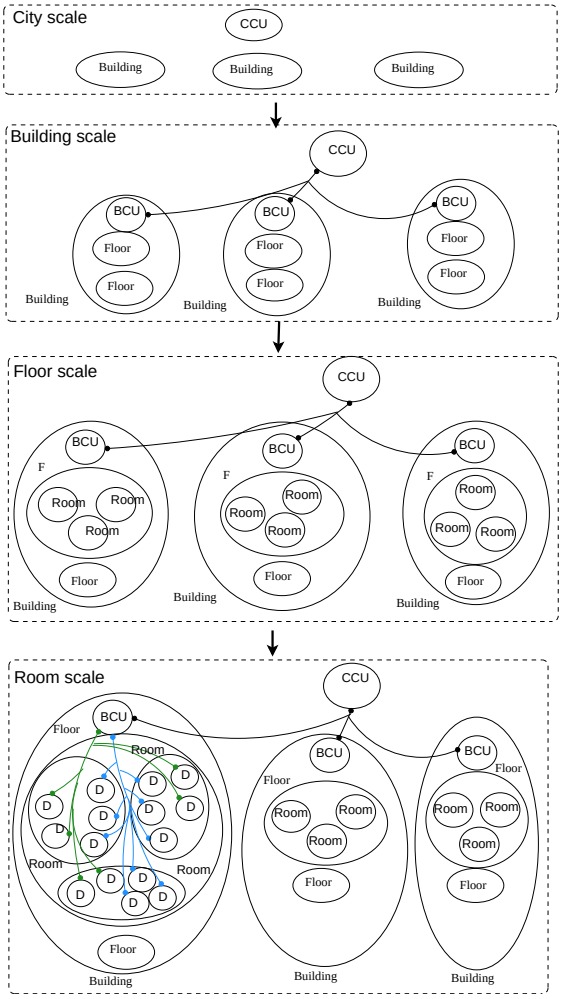


Figure 10: *The different scales of Smart Buildings as a bigraphical description*

Building scale This scale bigraphical model is showed in the second part of Figure 10, where *Floor*-nodes represent floors and *BCU*-nodes represent Building Control Units and edges represent connections between the the City Control Unit and Building Control Units.

Transition from building scale to floor scale The building scale model is refined in turn to obtain the floor scale model. This refinement consists in adding rooms. To do so, we instantiated the nesting meta-rule to have the rule $nest(Room, Floor)$ enabling to add a room in a floor. Furthermore, we applied the instantiated rule as many times as the number of rooms.

Floor scale The third part of Figure 10 shows the correspondent bigraphical model, where *Room*-nodes represent rooms. For sake of clarity, we represent only the rooms of one building floor.

Transition from floor scale to room scale The floor scale model is in turn refined in order to obtain the room scale model. This refinement is performed by applying rules enabling to add devices in rooms and to connect them with the Building Control Unit. To do so, we instantiated the nesting meta-rule to add devices ($nest(D, Room)$). Then, we have instantiated the connecting meta-rule for each system ($connect(hvac, 6)$ and $connect(alc, 9)$). In fact, *hvac*-nodes mark devices belonging to the HVAC system which are connected via the same hyperedge and *alc*-nodes mark devices belonging to the ACL system which are connected via another hyperedge.

Room scale This scale model is depicted in the last part of Figure 10. Likewise, for sake of clarity, we represent only the devices of one building floor. The devices that are connected by a blue hyperedge ensure the light control (represent the ACL system). Whereas the devices that are connected by a green hyperedge ensures the temperature control (represent the HVAC system).

Transition from room scale to device scale The room scale model is in turn refined in order to obtain the device scale model. This refinement is performed by applying rules enabling to add publish/subscribe components in devices and then to connect them. To do so, we instantiated the *addEPEC* and *addCM* meta-rules for each system. Then, we instantiated the *linkEP* and *linkEC* meta-rules for each system.

Device scale This scale model represents the architecture of the smart buildings SoS en-

hanced with the communication entities deployed in each device. EPs are connected to their corresponding CM through the port *ep* and ECs are connected to their CM through the port *ec*.

Hence, we obtain the bigraphical model that describes the architecture of Smart Buildings in a city. This model represents a correct architecture since the first scale architecture and all applied rules are corrects (i.e., conform to the architectural style).

iii. Modeling the Smart Buildings reconfigurations

According to the needs of the residents inside buildings and the resource changes of the equipped devices, many reconfigurations may be appealed. We take the example of some scenarios:

Scenario 1: In order to automate the switching on/off task of the Air conditioner, each room is equipped with a new presence sensor to be connected to the HVAC system. The air conditioner is turned on only when the room is occupied. In this case, we instantiate the meta-rule *addCS(s, Y)* with the parameter $s = hvac$ and $Y = D$. The obtained rule allows to add a device and connect it to the HVAC system.

Scenario 2: When a device breakdowns, it should be replaced by a good one. For example, if a lamp having the identifier $lamp1$ is broken down, it is replaced with a lamp having as identifier $lamp2$. In this case, we instantiate the meta-rule *replaceCS(cs1, cs2)* with the parameter $cs1 = lamp1$ and $cs2 = lamp2$. Then, we apply it on the SoS architecture.

V. Conclusions and Future Work

We proposed, B3MS, a novel methodology for SoS modeling based on the formal technique of Bigraphical Reactive System with an inspiring vision from multi-scale modeling. Multi-scale modeling is addressed in various disciplines such as physics, chemistry, biology and materials science. It allows to describe different granularity of a system at varying degrees of complexity and different description scales. In our work, we adopted this modeling approach for SoS architecture design since there is a need for both coarse-grained levels and fine-grained levels.

Our methodology provides a correct by design approach based on a formal model ensuring the correctness of the SoS architectures. This approach follows a refinement process for the transformation between coarse-grained and fine-grained descriptions. Considering that the first scale given by the designer is correct, this refinement is performed by adding lower scale details. It is implemented by applying corrects rules (i.e., preserve the system constraints). Hence, the obtained scale architectures are in turn corrects.

Moreover, we address the dynamic aspect of SoS by providing model-based rules of basic reconfigurations such us adding, removing and replacing constituent systems. These reconfigurations are defined as meta-rules ensuring the correctness of the evolved SoS architectures. To illustrate and evaluate our approach, we considered a case study called "Smart Buildings".

In future work, we aim at focusing more on the dynamic aspect of SoS by modeling safe reconfigurations. In fact, we should guarantee the correctness of the modeled reconfiguration rules defined by the designer.

References

References

[dan,] Danse - designing for adaptability and evolution in system of systems engineering. http://danse-ip.eu/home/main-project-objectives.html.

[Bryans et al., 2013] Bryans, J., Payne, R., Holt, J., and Perry, S. (2013). Semi-formal and formal interface specification for system of

systems architecture. In *Systems Conference (SysCon)*, pages 612–619.

[Chang et al., 2007] Chang, Z., Mao, X., and Qi, Z. (2007). An Approach based on Bigraphical Reactive Systems to Check Architectural Instance Conforming to its Style. In *The first Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE)*, pages 57–66.

[Clark, 2009] Clark, J. (2009). System of Systems Engineering and Family of Systems Engineering from a standards, V-Model, and Dual-V Model perspective. In *The 3rd Annual IEEE on Systems Conference*, pages 381–387.

[Garlan, 2003] Garlan, D. (2003). Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events. In *Formal Methods for Software Architectures*, volume 2804 of *Lecture Notes in Computer Science*, pages 1–24. Springer Berlin Heidelberg.

[Gassara et al., 2013] Gassara, A., Bouassida Rodriguez, I., and Jmaiel, M. (2013). Towards a Multi-scale Modeling for Architectural Deployment Based on Bigraphs. In *The 7th European Conference on Software Architecture*, ECSA, pages 122–129. Springer-Verlag.

[Gassara et al., 2015] Gassara, A., Bouassida Rodriguez, I., Jmaiel, M., and Drira, K. (2015). A Formal Method for Modeling Deployment Architectures Based on Bigraphs. *SIGAPP Applied Computing Review*, 15(2):8–16.

[Guessi et al., 2015a] Guessi, M., Cavalcante, E., and Oliveira, L. B. R. (2015a). Characterizing Architecture Description Languages for Software-intensive Systems-of-systems. In *The Third International Workshop on Software Engineering for Systems-of-Systems (SESoS)*, pages 12–18. IEEE Press.

[Guessi et al., 2015b] Guessi, M., Neto, V., Bianchi, T., Romero Felizardo, K., Oquendo, F., and Yumi Nakagawa, E. (2015b). A Systematic Literature Review on the Description of Software Architectures for Systems of Systems. In *30th ACM Symposium on Applied Computing*, pages 1–8.

[Ingram et al., 2004] Ingram, G., Cameron, I., and Hangos, K. (2004). Classification and analysis of integrating frameworks in multiscale modelling. *Chemical Engineering Science*, 59(11):2171 – 2187.

[Khlif et al., 2014] Khlif, I., Hadj Kacem, M., Hadj Kacem, A., and Drira, K. (2014). A Multi-scale Modelling Perspective for SoS Architectures. In *The 8th European Conference on Software Architecture Workshops*, ECSAW, pages 30:1–30:5. ACM.

[Madachy, 2009] Madachy, R. (2009). Integrated COCOMO suite tool for education. In *The 24th International Forum on COCOMO and systems/software cost modeling*.

[Maier, 1998] Maier, M. W. (1998). Architecting principles for systems-of-systems. *Systems Engineering*, 1(4):267–284.

[Meier and Cahill, 2005] Meier, R. and Cahill, V. (2005). Taxonomy of Distributed Event-Based Programming Systems. *The Computer Journal*, 48(5):602–626.

[Milner, 2009] Milner, R. (2009). *The Space and Motion of Communicating Agents*. Cambridge University Press.

[Nakagawa et al., 2013] Nakagawa, E. Y., Gonçalves, M., Guessi, M., Oliveira, L. B. R., and Oquendo, F. (2013). The State of the Art and Future Perspectives in Systems of Systems Software Architectures. In *The First International Workshop on Software Engineering for Systems-of-Systems*, SESoS, pages 13–20. ACM.

[Oquendo and Legay, 2015] Oquendo, F. and Legay, A. (2015). Formal Architecture Description of Trustworthy Systems-of-Systems with SosADL. *ERCIM News*, 2015(102).

[Pérez et al., 2013] Pérez, J., Díaz, J., Garbajosa, J., Yagüe, A., Gonzalez, E., and Lopez-Perea, M. (2013). Large-scale Smart Grids As System of Systems. In *The First International Workshop on Software Engineering for Systems-of-Systems*, SESoS, pages 38–42. ACM.

[Petitdemange et al., 2015] Petitdemange, F., Borne, I., and Buisson, J. (2015). Approach Based Patterns for System-of-Systems Reconfiguration. In *The IEEE/ACM 3rd International Workshop on Software Engineering for Systems-of-Systems (SESoS)*, pages 19–22.

[Riddle, 2012] Riddle, S. (2012). Contract-based modelling and analysis technologies for Systems-of-Systems. In *7th International Conference on System of Systems Engineering (SoSE)*, pages 469–470.

[Selberg and Austin, 2008] Selberg, S. A. and Austin, M. A. (2008). Toward an Evolutionary System of Systems Architecture. *INCOSE International Symposium*, 18(1):1065–1078.

[Valerdi and Lane, 2004] Valerdi, R. and Lane, J. A. (2004). Steps Toward Model Unification for Software, Systems Engineering, and Systems of Systems. In *The 19th International Forum on COCOMO and Software Cost Modeling*.

[Woodcock et al., 2012] Woodcock, J., Cavalcanti, A., Fitzgerald, J., Larsen, P., Miyazawa, A., and Perry, S. (2012). Features of CML: A formal modelling language for Systems of Systems. In *The 7th International Conference on System of Systems Engineering (SoSE)*, pages 1–6.

[Zhou et al., 2011] Zhou, B., Dvoryanchikova, A., Lobov, A., and Lastra, J. (2011). Modeling system of systems: A generic method based on system characteristics and interface. In *The 9th IEEE International Conference on Industrial Informatics (INDIN)*, pages 361–368.