A multi-scale modeling approach for software architecture deployment

Amal Gassara, Ismael Bouassida Rodriguez, Mohamed Jmaiel

A multi-scale modeling approach for software architecture deployment

Amal Gassara
ReDCAD Research laboratory,
University of Sfax, Tunisia
amal.gassara@redcad.org

Ismael Bouassida Rodriguez
CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
Univ de Toulouse, LAAS, F-31400 Toulouse, France
bouassida@redcad.org

Mohamed Jmaiel
ReDCAD Research laboratory,
University of Sfax, Tunisia
Research Center for Computer Science, Multimedia and Digital Data Processing of Sfax, B.P. 275, Sakiet Ezzit, 3021 Sfax, Tunisia
mohamed.jmaiel@enis.rnu.tn

ABSTRACT

For large component-based applications, identifying a valid deployment architecture has emerged as a major challenge. Actually, this deployment architecture (i.e, allocation of software components to its hardware hosts) should satisfy various constraints related to the software components and the target environment such as the hierarchical description of components, their connections and the resource constraints. The numerous constraints make hard to construct manually the correct deployment architecture. In this work, we propose a formal method based on a formal language called BRS (Bigraphical Reactive System) in order to guarantee the correctness of the deployment architecture. Furthermore, in order to support its automatic construction, our proposed method follows a multi-scale modeling. In fact, the designer starts by modeling the first scale architecture which is refined automatically by successively adding smaller scale components until obtaining the deployment architecture at the last scale. This refinement is ensured by applying a set of rules. In this paper, we address communicating systems as a study domain.

Keywords

Deployment architecture, Correctness, Multi-scale modeling, Bigraphs, BRS

1. INTRODUCTION

With the continuous change of computer systems, software development is more and more complex and brings new challenges. Software deployment is considered one of challenging tasks. It represents a sequence of related activities for placing a developed application into its target environment and making the application ready for use. For large distributed systems, finding a correct deployment architecture can be a challenging issue. Actually, a correct deployment architecture should satisfies various constraints related to both software components and target environment such as the hierarchical description of components, their connections and the resource constraints. These constraints make hard and might be impossible to find manually the correct architecture. Instead, several correct deployment architectures can be found. So, it is necessary to choose the efficient one to be deployed. The difficulty of this task motivate us to look for a solution to automate the construction of the correct and the efficient deployment architecture.

So, in this paper, we focus on the construction of correct deployment architectures (i.e., that respect structural constraints like the hierarchy of components and their connections). Then, in our ongoing work, the efficient architecture is selected according to resource constraints. In the literature, there are several research activities dealing with software deployment. But most of them are based on informal model and lack a solid mathematic foundation to ensure the correctness of deployment architecture. They have focused on satisfying only the resource constraints. Whereas, in our work, we use BRS (Bigraphical Reactive System) as a formal foundation. Moreover, our approach is based on a multi-scale modeling that helps to automate the construction of correct deployment architectures. Actually, in order to generate deployment architectures, we need to specify the software architecture model that describes the software components and their composition and the execution environment model that describes the target environment architecture on which application will be deployed. In fact, each model is represented as a set of scales, and each scale denotes a set of architectures represented as bigraphs. Following our approach, the designer starts by modeling the first scale architecture which is refined to give one or many architectures for the next scale. Then, these architectures are refined in turn to give the following scale architectures and so on until reaching the last scale. The transition between scales is ensured by applying specific rules defined as bigraphical reaction rules. After constructing the architectures of both software architecture model and execution environment model, we apply the relation between the two models in order to obtain deployment architectures.

The rest of this paper is organized as follows. In section 2, research activities dealing with software deployment are presented and in section 3, we present an overview of bigraphs. In section 4, we explain our bigraphical based approach for the deployment modeling. In section 5, we consider communicating systems as a study field. Then, we introduce in section 6 a case study called "Smart Home" to apply our approach and its simulation with the BPL Tool in section 7. Finally, section 8 concludes this paper and gives some directions for future work.

2. RELATED WORK

Various research studies have proposed methods to address the issues of software deployment. These methods include the use of OMG Deployment and Configuration (D&C) specification [13]. we have identified some frameworks which have been developed on top on this specification like DAnCE [3], Dacar [4] and Deployment Factory [6]. DAnCE is a QoS-enabled Component Deployment and Configuration Engine targeted for DRE systems. This framework deals only with CORBA Component Model. Whereas Deployment Factory is an unified environment for deploying component based applications. It proposes a generic component model which is an extension to the OMG D&C specification. These frameworks do not provide mechanisms for redeployment and dynamic reconfiguration. However, Dacar is a model-based framework for deploying autonomic software distributed systems. It is based on a control loop and Event-Condition-Action (ECA) rules. The main limitation of these research activities is the manual deployment planning. The designer should assign the software components to the hardware ones which is a hard task especially with large scale systems.

Other research activities have proposed architecture-based approaches using ADL (Architecture Description Language) [7, 9] and graphs [5, 15, 14]. Hoareau et al [7] present a support for deploying and executing an application built with hierarchical components. It presents an ADL extension for specifying a context-aware deployment. This deployment is performed in a propagative way and is driven by constraints put on the resources of the target hosts. The framework presented in the work of Malek et al [9] aims at finding the most appropriate deployment architecture for a distributed software system with respect to multiple QoS dimensions. The framework supports formal modeling of the problem that provides a set of algorithms for finding the optimal deployment. Heydarnoori et al [5] propose a graph based deployment planning approach for maximizing the reliability of component-based applications. They demonstrate that this deployment problem corresponds to the multiway cut problem in graph theory. Also, the work of Zhang et al [15] defines a component graph to represent component-based distributed applications and a tree network topology to describe the runtime environment. It defines the resource cost objective function and formulates component deployment optimization problem as mathematical programming problem.

Other research studies like [1, 10] have proposed a dedicated language (Domain Specific Language) for deployment. Dearle and Kirby [1] propose a framework for autonomic management deployment and configuration of component-based distributed applications. An initial deployment goal is specified using Deladas (DEclarative LAnguage for Describing Autonomic Systems). A constraint solver is used to find a configuration that satisfies the goal, and the configuration is deployed automatically. If, during execution, the goal is no longer being met, a full restart of the deployment process is performed. Matougui et al [10] propose the j-ASD middleware that addresses the autonomic deployment of ubiquitous systems. This middleware provides a DSL specifying deployment constraints. This specification is compiled into a constraint satisfaction problem, which is resolved automatically by a constraint solver. The generated deployment plan is dynamically executed by a mobile agent system.

We can note that the research activities [7, 9, 5, 15, 1, 10] deal only with resource constraints during the construction of the deployment architecture. They do not take into account the respect of structural constraints to validate he deployment architecture. Whereas, in our work, we deal with both structural and resource constraints.

3. PRELIMINARIES

3.1 Bigraphs

Bigraphs [12] formalise distributed systems by emphasizing both locality and connectivity. A bigraph consists principally of hyperedges and nodes which can be nested and have ports. Each hyperedge can connect many ports on different nodes. Each node in the bigraph is assigned a control. Controls indicate the node type and the node ports' number through the arity. We can use the notation "X-node", which means a node that has been assigned the control X.

3.2 Bigraphical Reactive System

A BRS (Bigraphical Reactive System) is a set of bigraphs and a set of reaction rules that may be applied to rewrite these bigraphs. Each reaction rule consists of two bigraphs: a *Redex R* and a *Reactum R'*. The application of the rule consists of identifying the image of R in a bigraph and replacing it by the corresponding R'.The graphical representation used above is handy for modeling, but unwieldy for reasoning. Fortunately, bigraphs have an associated term language [2].

4. THE PROPOSED APPROACH

In order to construct a deployment architecture, we need to describe the software architecture, the execution environment and the relation between them. Based on this issue, we propose an approach for deployment modeling of distributed systems which defines:

- **Software architecture model:** This model describes software components, their properties and their architecture (i.e, hierarchy of components and connections between them).
- **Execution environment model:** This model describes the runtime environment including physical nodes, hosts, devices, etc as well as their resource constraints and their architecture.

- **Relation between the software architecture model and the execution environment model:** To obtain a deployment architecture, we should define the relation between the two models to map software components on physical ones.

The key objective of our work is to automate the construction of a correct deployment architecture that respects the defined models. For this, we have proposed a formal method which is based on a formal language to guarantee the correctness of the deployment architecture. This formal language should be able to describe both software and physical components. It should emphasize both hierarchy and connectivity of components. It should also provides information on both static and dynamic aspect of the system since we intend to deal with autonomic systems in future work. We have noticed that BRS is the most appropriate language that supports these requirements. Furthermore, our formal method provides three steps to be followed:

- Step 1: Description In this step, the designer describes the necessary information like software and hardware components, their properties and their resource constraints. Each component is represented with Bigraph as a node type annotated with attributes to indicate properties or available resources. The designer describes also the structural constraints through conditions on the hierarchy and the connectivity of nodes.

- Step 2: Generation In this step, the generation of the deployment architecture is performed automatically following a multiscale modeling approach. In fact, for each model (i.e., environment execution model and software architecture model), a large scale is defined by the designer. Then, it is refined by successively adding smaller scale details until reaching the last scale. Hence, we obtain the set of possible deployment architectures by linking the two models.

The refinement process is performed by applying specific rules. Since we aim to facilitate the modeling task for the designer, we have proposed the concept of meta-rule to describe the transition between scales. Thus, the designer identifies the corresponding meta-rule which will be instantiated automatically according to the specification in order to have the necessary rules for scale transitions. With BRS, a meta-rule is a meta-reaction rule that contains nodes having a variable control (i.e., a variable can represent any control).

- Step 3: Selection In our ongoing work, a deployment architecture is selected from those generated in the previous step according to resource constraints. Hence, we obtain a deployment architecture that respects both structural and resource constraints.

5. MULTI-SCALE MODELING FOR COMMUNICATING SYSTEMS

In our work, we have addressed communicating systems. Since we aim to facilitate the modeling task to the designer, we have defined, for these systems, the scales of each model and we have defined the necessary meta-rules to be applied for the transitions between these scales and for the relation between the execution environment model and the software architecture model.

5.1 The execution environment model

This model is represented by the following scales and transitions (we use the notation "scale i" where i is the scale number):

- Scale i: such $i \in [0, n]$ where n corresponds to the depth of nesting in a bigraph. For $i = 0$, we obtain the first scale.

- Transition from scale i to scale $\mathbf{i+1}$: The transition to the scale $i+1$ is obtained by applying a meta-reaction rule allowing to nest a node.The corresponding algebraic expression of this rule is:
Nest a node: $X.d_0 \rightarrow X.(Y|d_0)$
This rule enables to nest a node. So, the transition between two scales leads to increment by 1 the depth of nesting. Therefore, this meta-rule can be applied several times in order to add many nodes residing in the same node.

- Scale $\mathbf{i+1}$: such $i \in]0, n]$. With i=n, we obtain the scale n that represents all physical entities and their composition (i.e, hierarchy). So, we reach this scale when there is no physical entities to add.

- Transition from scale n to scale $\mathbf{n+1}$: The transition to the scale $n+1$ is characterized by defining the link graph. So, we add hyperedges that represent the communication between different devices of the application. This operation is defined by a closure $/x \circ$ G (i.e., outer names x under a bigraph G is replaced by an edge). Hence, we link nodes belonging the same communication group (i.e., having the same outer name).

- Scale $\mathbf{n+1}$: This is the last scale of the execution environment model. It represents all the physical entities and their communication.

5.2 Relation between execution environment and software architecture models

We propose that the relation between the software architecture model and the execution environment model is a transition from scale $n+1$ of the execution environment model to scale 0 of the software architecture model. The latter includes sender and receiver components. In fact, each communication group is ensured by a set of senders and receivers. We consider that communication is done in pull mode (i.e., response to a request). So, an entity belonging a communication group should contains a pair of sender and receiver.

To ensure this transition, we define the following meta-rule:
Add a sender and a receiver: $Y_x \rightarrow Y_x.(Sr.x|Rc.x)$
We nest in each node having an outername x, a sender (Sr-node) and a receiver (Rc-node), then we nest in both of them an x-node that mark their communication group.

5.3 The software architecture model

For communicating systems, the software architecture model includes the entities that take part in the communication like senders, receivers and communication middleware components. Hence, we have identified for this model the necessary components, three scales and transitions between them by defining corresponding meta-rules.

- Scale 0: represents sender and receiver components.

- Scale 1: provides the middleware components that ensure the communication between the application components. Here, we use the Event-Based Communications (EBC) [11]. EBC is a communication model which provides three types of EBC entities: *event producers* (EP), *event consumers* (EC) and *channel managers* (CM). The EP and EC can be connected to CM, but they can not be directly interconnected. The EP can send data to the CM to which they are connected. The CM returns a copy of the received data to all the EC connected to it.
This scale is obtained by nesting an EP-node in each sender, an EC-node in each receiver and a CM-node for each communication group in a node that belongs to this group.

- Transition from scale 0 to scale 1: This transition is performed by applying a set of meta-reaction rules defined by the algebraic expressions given below:
Add an EP: $Sr.x \to Sr.EP.x$
Add an EC: $Rc.x \to Rc.EC.x$
Add a CM: $/x\ X1_x||...||Xn_x \to /x\ X1_x||...||(Xn_x|CM.x)$
For the third rule (Add a CM), n is the number of nodes belong a communication group. It will be instantiated for each communication group.

- Scale 2: This is the last scale of the software architecture model. It consists at enriching the *link graph* by adding new edges that link EBC components.

- Transition from scale 1 to scale 2: Reaching the scale 2 is obtained by applying a set of meta-reaction rules given below:
Link EP to CM: $EP.x||CM.x \to /y\ EP_y||CM_y.x$
Link EC to CM: $EC.x||CM.x \to /y\ EC_y||CM_y.x$

6. CASE STUDY: SMART HOME

In order to apply our approach, we consider a case study named "Smart Home" denoted in the Figure 1. Each room in a smart home 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 to a home gateway that manages their communication to ensure an intelligent home control like lighting control and temperature control. Sensors record information such as rooms lighting, human presence, temperature, etc. The home gateway receives these information and analyses them in order to configure the devices.

6.1 The execution environment model

For the smart home, the execution environment model represents home, rooms, home gateway and devices. It includes the following scales.

- Scale 0: The designer identifies the node controls (H representing a home,R representing a room, a HG representing a home gateway and D representing a device: sensor or actuator). Then, he models this scale using a bigraph. For the smart Home, this bigraph contains one H-node that represents a Home (cf. Figure 2).

Figure 1: Smart Home

- Transition from scale 0 to scale 1 (adding Rooms and Home Gateway): The transition to the scale 1 is obtained by instantiating the meta-rule for nesting a node. So, the rule is: $H.d_0 \to H.(R|d_0)$.
This rule enables to add a Room (R-node) in a Home. We apply this rule as many times as the number of rooms in the home. This number is given by the designer. Here, we have 3 rooms.
The meta-rule for nesting a node is instantiated again to add a Home Gateway. The rule is: $H.d_0 \to H.(HG|d_0)$
- Scale 1: This scale presents a home, three rooms and a home gateway. Its bigraph is depicted in Figure 2.
- Transition from scale 1 to scale 2 (adding Devices): The transition to the scale2 is obtained by instantiating the meta-rule for nesting a node. So, the rule is:
$R.d_0 \to R.(D|d_0)$.
This rule enables to add a device (D-node) in a room. We apply this rule as many times as the number of devices in the room. This number is given by the designer. Here, we have 5 devices.
- Scale 2: In this scale, we obtain the bigraph specifying the home, the home gateway and the 3 rooms. One of these rooms contains 5 devices. This bigraph is depicted in Figure 2.
- Transition from scale 2 to scale 3 (connecting entities within groups): The transition to scale 3 is obtained by applying the closure operation on the bigraph of the scale 2: $/gt\ gl \circ scale2$
This closure operation enables to link lighting communication group (i.e., link the Home Gateway with the three devices having an outer name gl: presence sensor, light sensor and lamp). It enables also to link temperature communication group (i.e., link the Home Gateway with the two other devices having an outer name gt: thermometer and air conditioner).
- Scale 3: The scale bigraph is defined in the last part of Figure 2.

6.2 Relation between execution environment and software architecture models

The transition from scale n of the execution environment model to scale 0 the software architecture model is obtained by instantiating the meta-rule for adding a sender and a receiver for devices within the temperature communication group and devices within lighting communication group. For sake of shortness, we present the instantiated rules for temperature communication group:
$D_{gt}.d_0 \to D_{gt}.(Sr.gT|Rc.gT|d_0)$
This rule enables to add a sender (Sr-node) and a receiver

Figure 2: Scales of the execution environment model for Smart Home

(Rc-node) in a device (D-node). The gT-node nested in a sender or a receiver denotes the temperature communication group. We instantiate the meta-rule again to add senders and receivers in the home gateway.
So the rule is: $HG_{gt}.d_0 \rightarrow HG_{gt}.(Sr.gT|Rc.gT|d_0)$

6.3 The software architecture model

This model represents senders, receivers and EBC components.

- Scale 0: This scale represents the execution environment model including sender and receiver components.
- Transition from scale 0 to scale 1 (adding EBC components): The transition to the scale 1 is obtained by instantiating the three meta-rules of adding EP, adding EC and adding CM. So, the instantiated rules for the temperature communication group are:
$Sr.gT \rightarrow Sr.EP.gT$
$Rc.gT \rightarrow Rc.Ec.gT$
$/gt\ D_{gt}||D_{gt}||HG_{gt} \rightarrow /gt\ D_{gt}||(D_{gt}|CM.gT)||HG_{gt}$
- Scale 1: At this scale, we have the execution environment model (home, home gateway, rooms and devices) with senders, receivers and EBC components. The bigraph at this scale is like the bigraph at the scale 2 depicted in Figure 3 but without the green hyperedges. In this scale, we can obtain many Bigraphs due to the choice of the channel manager placement (i.e., the CM-node is deployed on one node belongs to the communication group).
- Transition from scale 1 to scale 2 (connecting EBC components): The transition to the scale 2 is obtained by instantiating the two meta-rules of linking an EP to a CM and linking an EC to a CM:
$EP.gT||CM.gT \rightarrow /z\ EP_z||CM_z.gT$
$EC.gT||CM.gT \rightarrow /u\ EC_u||CM_u.gT$

Figure 3: The Bigraph at the scale 2

- Scale 2: The bigraph obtained at this scale is denoted in Figure 3. It depicts deployment infrastructure, senders, receivers and connected EBC components. So, it defines one of the set of deployment architectures.

7. VALIDATION WITH BPL

In order to verify the feasibility of the case study, we model our BRS model using the BPL Tool (Bigraphical Programming Languages) [8]. BPL is a tool for experimenting with bigraphical models. It provides manipulation and simulation of BRS. It relies on an SML (Standard ML) compiler with an interactive mode to provide a command line interface. The language used in the BPL Tool is called BPLL (BPL Language), and it consists of a number of SML constructs which allows to write BPLL directly in SML programs.

For the implementation of our case study, we create a SML file to define the BRS for the execution environment model. Listing 1 presents a portion of this file. In this listing we define the signature of the system denoted in lines 2-5. It is the set of nodes controls. Then, we define the rules denoted in lines 7-13 (i.e., rule for adding a room, adding a home gateway and adding a device). We define also the tactics for prescribing the sequence in which reaction rules should be applied (lines 15-18 of listing 1). Finally, we define the initial system denoted in line 21 of listing 1. It represents the Home.

Listing 1: Execution environment model implementation

```
1  (* Nodes controls *)
2  val H = active0 ("H")
3  val R = active0("R")
4  val D = active ("D" -:1)
5  val HG = active ("HG" -:2)
6  (* Rules for execution environment model *)
7   val add_room="add_room":::
8       H o idp(1) --[0|->0]--|> H o (idp(1) `|` R o <->)
9   val add_HG="add_HG":::  H o (idp(1)) --[0|->0]--|>
10    (-/gt*-/gl) o H o (idp(1) `|` HG[gt,gl] o <->)
11  val add_device_gt="add_device_gt":::
12     H o (idp(1) `|` R o idp(1)) --[0|->0,1|->1]--|>
13     -/gt o H o (idp(1) `|` R o (idp(1) `|` D[gt] o gT))
14  [...]
15  (* Tactics *)
16  val rules = mkrules[add_room,add_HG,...]
17  val tactics_01 = 3 TIMES_DO react_rule "add_room"
18  ++ react_rule "add_HG"
19 [...]
20 (* Initial system : scale 0 *)
21 val scale0 = H o <->
```

After running the simulation, we obtain the expression of each scale in BPLL. Furthermore, to implement the software architecture model, we complete the SML file with the nodes controls (i.e., Sr representing a Sender, Rc representing a Receiver, EC representing an Event Consumer, EP representing an Event Producer and CM representing a Channel Manager), the rules and their tactics (i.e., rules for adding senders and receivers, rules for adding EBC components and rules for connecting EBC components within communication groups). After running the simulation of the software architecture model implementation, we obtain the expression of its scales in BPLL.

8. CONCLUSION AND FUTURE WORK

In this paper, we have focused on one of the challenging tasks of software deployment which consists in the construction of a correct deployment architecture. To tackle this issue, we have proposed a formal method based on bigraphical reactive systems. This formal language allows to guarantee a correct by construction architectures. This method provides three steps. At the first step, the designer describes the necessary information for the execution environment model, the software architecture model and the relation between them. Then, the second step consists in generating automatically all the correct deployment architectures following a multiscale modeling approach. In fact, for each model, a large scale is defined by the designer. Then, it is refined by successively adding smaller scale details. This refinement process is performed by applying specific rules. Finally, the third step is the selection of the efficient deployment architecture according to resource constraints. In our work, we have addressed communicating systems. For these systems, we have identified some information in order to ease the task for the designer in the description step. In fact, we have defined the component types for the software architecture model, the scales of each model and the transition between them and also the relation between the software architecture model and the execution environment model. Finally, in order to illustrate our approach, we have presented a case study called Smart Home and its implementation using BPL Tool. In future work, we aim to focus on the third step of our approach (i.e, selecting the efficient deployment architecture according to resource constraints). Then we intend to deal with autonomic systems by planning redeployment actions. Moreover, we are working at implementing a tool for Bigraph transformations since we have noticed some weaknesses of BPL Tool and we have noticed also that it does not meet our needs.

9. ACKNOWLEDGMENTS

This research is supported by the ITEA2's A2NETS (Autonomic Services in M2M Networks) project[1].

10. REFERENCES

[1] A. M. A. Dearle, G. Kirby. A framework for constraint-based deployment and autonomic management of distributed applications. In *International Conference on Autonomic Computing*, pages 300–301, may 2004.

[2] L. Birkedal, S. Debois, E. Elsborg, T. Hildebrandt, and H. Niss. Bigraphical models of context-aware systems. In L. Aceto and A. Ingólfsdóttir, editors, *Proceedings of the 9th International Conference on Foundations of Software Science and Computation Structure (FoSSaCS'06)*, volume 3921 of *Lecture Notes in Computer Science*, pages 187–201. Springer-Verlag, Mar. 2006.

[3] G. Deng, J. Balasubramanian, W. Otte, D. C. Schmidt, and A. S. Gokhale. Dance: A qos-enabled component deployment and configuration engine. In *Component Deployment*, pages 67–82, November 2005.

[4] J. Dubus and P. Merle. Towards Model-Driven Validation of Autonomic Software Systems in Open Distributed Environments. In *Workshop M-ADAPT, in conjunction with ECOOP 2007*, July 2007.

[5] A. Heydarnoori and F. Mavaddat. Reliable deployment of component-based applications into distributed environments. In *Proceedings of the Third International Conference on Information Technology: New Generations*, ITNG '06, pages 52–57. IEEE Computer Society, April 2006.

[6] P. Hnetynka. A model-driven environment for component deployment. In *Proceedings of the Third ACIS Int'l Conference on Software Engineering Research, Management and Applications*, SERA '05, pages 6–13. IEEE Computer Society, Aug. 2005.

[7] D. Hoareau and Y. Mahéo. Constraint-based deployment of distributed components in a dynamic network. In *Proceedings of the 19th international conference on Architecture of Computing Systems*, pages 450–464, March 2006.

[8] E. Hojsgaard and A. J. Glenstrup. The bpl tool: A tool for experimenting with bigraphical reactive systems. Technical Report TR-2011-145, IT University of Copenhagen, Oct. 2011.

[9] S. Malek, N. Medvidovic, and M. Mikic-Rakic. An extensible framework for improving a distributed software system's deployment architecture. *Software Engineering, IEEE Transactions on*, 38(1):73–100, Jan. 2012.

[10] M. E. Matougui and S. Leriche. A middleware architecture for autonomic software deployment. In *ICSNC '12 : The Seventh International Conference on Systems and Networks Communications*, pages 13–20. XPS, November 2012. 12619 12619.

[11] R. Meier and V. Cahill. Taxonomy of distributed event-based programming systems. In *The Computer Journal*, pages 585–588, July 2002.

[12] R. Milner. Bigraphical reactive systems: basic theory. Technical Report UCAM-CL-TR-523, University of Cambridge, Computer Laboratory, Sept. 2001.

[13] I. Object Management Group. Deployment and configuration of component-based distributed applications specification, version 4.0, Apr. 2006.

[14] I. B. Rodriguez, N. V. Wambeke, K. Drira, C. chassot, and M. Jmaiel. Multi-layer coordinated adaptation based on graph refinement for cooperative activities. *Communications of SIWN*, 4(1):163–167, 2008.

[15] Q. Zhang, D. Qiu, Q. Tian, L. Sun, and X. Xu. Deployment planning of component-based distributed applications using mathematical programming. In *Computational Intelligence and Software Engineering (CiSE 2010)*, pages 1 –4, Dec. 2010.

[1]https://a2nets.erve.vtt.fi/