A UML-based Approach for Multi-scale Software Architectures

Ilhem Khlif[1,2,3], Mohamed Hadj Kacem[1], Ahmed Hadj Kacem[1] and Khalil Drira[2,3]
[1] *University of Sfax, ReDCAD Laboratory Research, Sfax, Tunisia*
[2] *CNRS, LAAS, 7 avenue du Colonel Roche, F-31400 Toulouse, France*
[3] *Univ. de Toulouse, LAAS, F-31400 Toulouse, France*

Keywords: Software Architecture, Multi-scale Description, UML Notation, Refinement, Abstraction, Model Transformation.

Abstract: Multi-level software architecture design is an important issue in software engineering. Several research studies have been done on the modeling of multi-level architectures based on UML. However, they neither included the refinement between the levels nor clarified the relationships between them. In this paper, we propose a multi-scale modeling approach for multi-level software architecture oriented to facilitate adaptability management. The proposed design approach is founded on UML notations and uses component diagrams. The diagrams are submitted to vertical and horizontal transformations for refinement; this is done to reach a fine-grain description that contains necessary details that characterize the architectural style. The intermediate models provide a description with a given abstraction that allow the validation to be conducted significantly while remaining tractable w.r.t. complexity. The validation scope can involve intrinsic properties ensuring the model correctness w.r.t. the UML specification. To achieve this, we propose a set of *model refinement rules*. The rules manage the refinement and abstraction process (vertical and horizontal) as a model transformation from a coarse-grain description to a fine-grain description. Finally, we experimented our approach by modeling an Emergency Response and Crisis Management System (ERCMS) as a case of study.

1 INTRODUCTION

Software architecture design and description is an important activity in software engineering. It is crucial during the earlier steps of the development of large and complex software systems for mastering the costs and the quality of their development.

The design of a software architecture based on a multi-level description is a complex task. Several studies proposed design approaches based on UML to handle this complexity. However, they neither included the refinement between the levels nor clarified the relationships between them. In this work, we explore multi-level software architectures and propose a multi-scale description for them. The multi-scale perspective considers the refinement of a coarse-grain description level to a fine grain description level aiming to minimize software systems complexity. The issue addressed in this approach, is to define the relationships that may exist between different description levels of a model, and also between different horizontal descriptions inside a given scale.

Our objective is to provide solutions for modeling software architectures to facilitate their validation at different description levels. The contribution of this paper is a multi-scale modeling approach for multi-level architecture oriented to facilitate adaptability management. The approach extends the work presented by Khlif et al. (Khlif et al., 2012), (Khlif et al., 2014). In (Khlif et al., 2012), the multi-scale modeling solution only considers a fixed number of scales and describes a progressive process of refinement from a generic model describing a given point of view at a given scale to a specific model describing this point of view at another scale. In (Khlif et al., 2014), we have proposed a multi-scale modelling perspective for Systems of Systems (SoS) architecture description. We have focused on SysML (System modeling language) notations and used block diagrams.

In this work, we extend the approach by considering both the refinement and the abstraction process as a vertical and horizontal model transformation from a coarse-grain description to a fine-grain description, and by adding details on vertical and horizontal scales until reaching a fine-grain description that contains necessary details that characterize the architectural style. The intermediate models provide a description

khlif I., Hadj Kacem M., Hadj Kacem A. and Drira K..
A UML-based Approach for Multi-scale Software Architectures.
DOI: 10.5220/0005380403740381
In *Proceedings of the 17th International Conference on Enterprise Information Systems* (ICEIS-2015), pages 374-381
ISBN: 978-989-758-097-0

with a given abstraction that allow the validation to be conducted significantly while remaining tractable w.r.t. complexity. The validation scope can involve intrinsic properties ensuring the model correctness w.r.t. the UML specification (eg. interface compatibility) To achieve this, we propose a set of *model refinement rules*. The rules manage the refinement and abstraction process (vertical and horizontal) as a model transformation from a coarse-grain description to a fine-grain description. In order to show the viability and usefulness of our solution, we present the modeling of an Emergency Response and Crisis Management System (ERCMS) as a case of study.

The remainder of the paper is organized as follows. In section 2, we present a survey of related work. We describe our approach in section 3. Section 4 illustrates the case study. We conclude and outline some perspectives in section 5.

2 RELATED WORK

Considerable research studies have been presented on the description of software architectures. Levels and views are two major concepts introduced in the software engineering domain in order to enhance the architectural organization of complex systems' requirements. A view is a description of the whole system from the perspective of a related set of concerns. As refinement of the architectural description continues, the views of a software architecture can be organized into distinct architectural levels (or layers). These levels are distinguishable by their degree of abstraction and by the stakeholder concerns they capture. The basic idea of multi-level modeling is to explicitly represent the different abstraction levels to enable a clear encapsulation of problems in different levels to tackle them independently. Low level details are abstracted for higher levels, and therefore complexity is reduced, thus achieving a clear and coherent design. Several works have focused on the use of UML for multi-level modeling.

2.1 Multi-level Description with UML

Mallet et al. (Mallet et al., 2010) proposed a UML profile called "UML Domain Specification". They applied the multi-level paradigm to the MARTE model time.

Anwar et al. (Anwar et al., 2010) described a formalism and methodology to support view-based modeling. They developed a design methodology based on the UML profile, called the VUML (View-based Unified Modeling Language), that enables the modeling of a software system according to the viewpoint of each actor. The authors proposed a framework to generate a set of executable composition-oriented transformations and implemented transformation rules.

Both works (Mallet et al., 2010; Anwar et al., 2010) used UML for modeling multi-level and multi-view architecture. They, however, did not include the refinement between the levels or views nor clarified the relationships that hold between them.

2.2 Architecture Refinement

In (Ferrucci et al., 1992), the authors defined the refinement between levels to provide more details. This guarantees that any property of the higher level is also verified at the lower one.

In (Rafe et al., 2009), Rafe et al. proposed an automated approach to refine models in a specific platform. For each abstraction level, a style should be designed as a graphical diagram and graph rules. In their approach, the model design can transform or refine the PIM into service specific models that are designed by the rules of graph transformation. Their modeling approach was not illustrated through a case study or implemented in a development tool.

Juan et al. (Juan et al., 2010) proposed a formal framework for describing software architectures using Π-ARL, an architecture refinement language that requires the use of a toolset to support complex systems.

Bouassida et al. presented, in (Bouassida et al., 2010), proposed a multi-level modeling approach for collaborative systems. They examined several problems related to these systems and offered a design framework for solving them. The authors presented two procedures for converting models between levels, namely refinement and selection. To validate their work, the authors defined a generic algorithmic framework for multi-level architectural reconfiguration and applied it to the case of group communication and cooperation using formal techniques. This work explored architectural refinement based on graph transformations. The application of these techniques necessesarily requires special training and mathematical expertise.

Accordingly, several works sought to explore other multi-level or multi-layer approaches.

2.3 Multi-level/ Multi-layer Architecture

Lange and Middendorf (Lange and Middendorf, 2010) introduced a multi-level concept for reconfig-

urable architectures with more than two levels. In their work, the authors developed an algorithm that reduces cost and time reconfiguration and proposed a heuristic to solve the optimal number of reconfiguration levels.

In (Baresi and Guinea, 2013), Baresi et al. investigated domain-specific abstractions for a multi-level service based system. They proposed the Multi-layer Collection and Constraint Language(mlCCL) which allows to analyze runtime data in a multi-layered system. To support this language, they implemented a visualization tool by extending their ECoWare framework to event correlation. The authors did not, however, deal with the refinement between levels or layers.

Brosch et al. (Brosch et al., 2011) proposed an approach that integrates multi-level monitoring and adaptation of software architectures, where the authors proposed a meta-model for specifying adaptability characteristics in a software product line. This model is expressed on different levels. The architectural level uses composite components to encapsulate subsystems and enable their replacement. The component level selects component implementations for a given specification. The component implementation level uses component parameters for specific product configurations The resource level uses allocation references for for product line configurations. The authors presented a tool support that allows the architects to design the architecture with UML models, which are automatically transformed to Markov-based prediction models, and evaluated to determine the expected system reliability.

2.4 Discussion

A thorough overview of the literature indicates that several studies have been performed on the modeling of multi-level, multi-layer, multi-view architectures. Views described the system from the perspective of the stakeholders' concerns. As refinement of the architectural description continues, the views of a software architecture can be organized into distinct architectural levels (synonym to layers). We continually update the architecture description refinement to demonstrate that the levels can also be organized into different description **scales** oriented to facilitate the architectural organization of complex systems' requirements. Several studies have been presented for modeling these architectures using a unified and visual notation based on UML. These semi-formal approaches did not, however, include the concept of refinement. Although formal techniques and, more specifically, works based on graph transformations allow architecture refinement, they require certain expertise in mathematics for architects. Moreover, only few studies have provided a clearly defined process that takes the compatibility between different description levels into account, a challenging condition for the multi-level description of dynamic architectures. Our work focuses on modeling software architectures.

Similarly to those works, we consider that a given modeling level can be described by several scales. We choose pure UML notations to describe multi-scale architectures. We need to follow a clear refinement process to transit between scales. The aim of this work is to provide solutions for modeling software architectures so as to facilitate their validation at different description levels. Thus, our approach is adding a refinement element based on "scales" compared to other UML approaches. The multi-scale perspective is using onely the multi-level concept to describe software architectures.

3 PROPOSED APPROACH

We propose a multi-scale modeling approach for multi-level software architectures. The proposed design approach is founded on UML notations and uses component diagrams. The diagrams are submitted to vertical and horizontal transformations for refinement; this is done to reach a fine-grain description that contains necessary details that characterize the architectural style.

We present the multi-scale approach by a two-dimensional array describing vertical and horizontal refinements. Vertical description levels allow the architect to describe the same inherent requirements while providing multiple descriptions having different granularity levels. Under each scale, there are several horizontal refinements. We define a *Vertical description Scale* "Sv_n" as a model that provides additional details of design (in a vertical way) which is related to the higher scale "Sv_{n-1}" and more abstraction which is related to the lower scale "Sv_{n+1}". It can be used, for example, to represent a given description level or a given communication layer in communicating systems. Moreover, a vertical scale can be refined into several *Horizontal description Scales* "Sv_n/Sh_m" providing more details of the UML initial description (in a horizontal way). We consider that n (resp. m) represents the scale number (n,m $\geq$ 0).

We start by modeling the first scale, which is defined by a UML component diagram. This diagram is refined, through model transformation operations, until reaching the last scale that represents the architectural style. A simple description of our pro-

posed approach is presented in (Figure 1). We propose a hybrid approach. The top-down approach is presented by the refinement process which transforms architecture in both a vertical and a horizontal way. The bottom-up approach is described by the abstraction process, which consists of vertical and horizontal transformations. A multi-scale description guarantees the execution of the necessary model transformations rules. These rules manage the refinement/abstraction between scales.

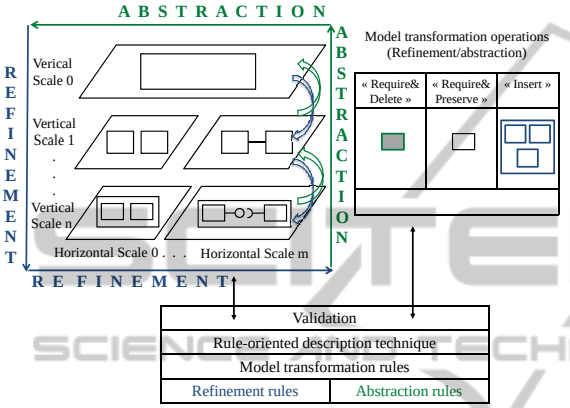

Figure 1: Description of the proposed approach.

3.1 Multi-scale Description

We present the multi-scale approach by a two-dimensional array describing vertical and horizontal scales (Figure 2). Gray classes represent the added details in each refined scale and white classes represent the conserved architectural properties. Vertical scales are the vertical description levels that allow the architect to describe the same inherent requirements while providing multiple descriptions having different granularity levels. Under each vertical description scales there are several horizontal description scales. The first scale Sv_0 begins with specifying the application requirements. It defines the whole application by its name.

Two horizontal refinements called horizontal scales are associated with the first level Sv_1. The first horizontal scale shows all component types that compose the application. The second one describes the links between those components.

Four horizontal refinements are associated with the second level Sv_2. The first scale presents composites for component types, and enumerates all the roles that each component can take. The second one identifies the list of communication ports for each component, and refines those roles. The third one shows the list of interfaces for communication ports. The last one is obtained by successive refinements while adding the list of connections established between components and composites. This scale allows us to define the architectural style.

3.2 Model Refinement Rules

To ensure the correctness of the modeled system, we adopt a rule-oriented description technique. The rules manage the refinement between scales.

3.2.1 Vertical Model Refinement

R_0 **rule:** Sv_0 represents the entire application `Application-Name` as a unique class. The uniqueness is represented by the multiplicity "1". $R_{1.1}$ **rule:** The transition from Sv_0 to Sv_1 provides details on the generic component. Sv_1 includes a class `Component-Type` that specifies more than "2" components in the application. $R_{2.1.1}$ **rule:** The transition from Sv_1 to Sv_2 provides more details on the properties of component types, and data relating to the components. Sv_2 consists of `Component-Type`, eventually composed of `Composite`, and enumerates roles for each component and each composite.

3.2.2 Horizontal Model Refinement

$R_{1.2}$ **rule:** The transition from Sv_1/Sh_0 to Sv_1/Sh_1 means the addition of associations between component types. A class `Association` is between at least "2" types of components, and contains "2" ends of association. A class `Association-End` has a multiplicity defined by a lower limit set to "1" and an upper bound set to "*". $R_{2.1.2}$ **rule:** Sv_2/Sh_0 adds details on the components by specific data through enumerations. An enumeration named *« Role »* is used to enumerate roles that components can take. A component can play a role among this list: ‘‘*"Event-Dispatcher"*, *"Producer"*, *"Consumer"*, *"Producer-Consumer"*, *"Server"*, *"Service"* , *etc.* $R_{2.2}$ **rule:** The transition from Sv_2/Sh_0 to Sv_2/Sh_1 means the addition of communication ports and details on previous enumerations. The class `Port` represents the point of interaction for a component. A component can have one or more ports. One port is associated with a component. An *"Event-Dispatcher"* can have an architecture using a centralized event service or an architecture using a distributed event service. An enumeration called *« Topology »* can be *"Unique-Dispatcher"*, *"Network-Dispatcher-Hierarchical"*, *"Network-Dispatcher-AcyclicPeerToPeer"*, or *"Network-Dispatcher-GeneralPeerToPeer"*. $R_{2.3}$ **rule:** The transition from Sv_2/Sh_1 to Sv_2/Sh_2 allows us to add all provided and required services called

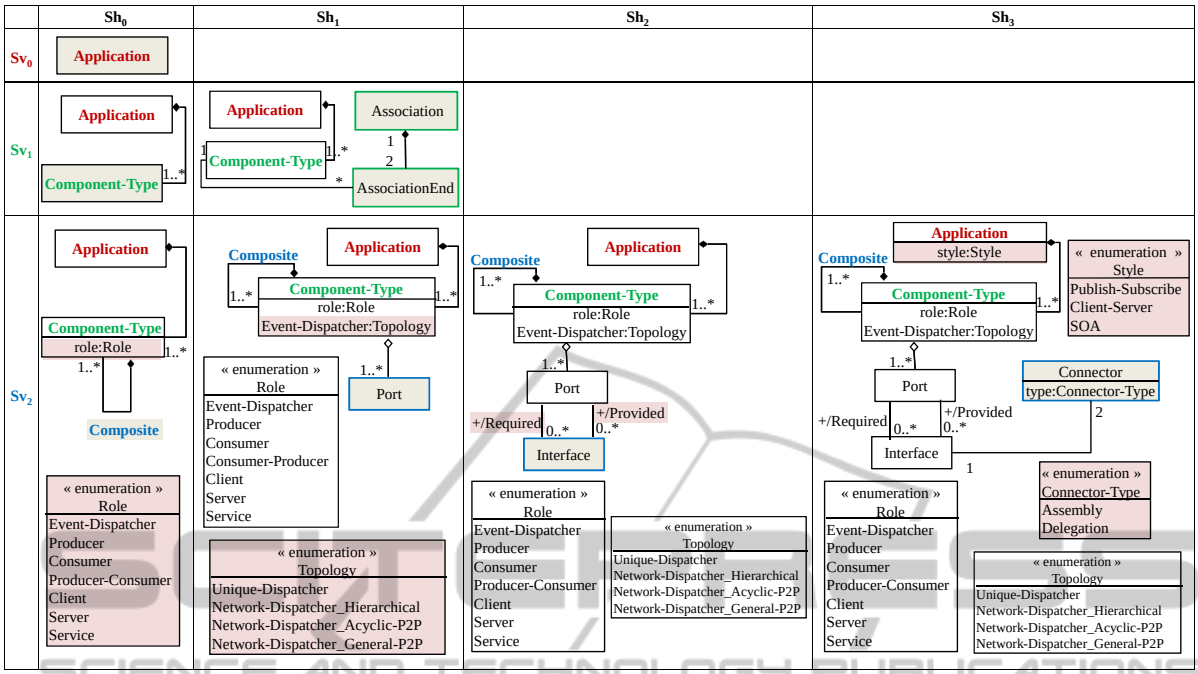

Figure 2: UML notations of a multi-scale architecture description.

interfaces. The class `Interface` represents the interface of a component. An interface can have a type which is either provided '*+=Provided*' or required '*+=Required*'. $R_{3.4}$ **rule:** The transition from Sv_2/Sh_2 to Sv_2/Sh_3 means the addition of connections. The class `Connector` refers to a link that enables communication between two or more components. The enumeration « *Connector-Type* » specifies two types of connectors: *"Delegation"* and *"Assembly"*. For each connection, it is necessary to identify the source and the recipient by respecting the followed topology. Sv_2/Sh_3 allows us to determine the architectural style of the application. An enumeration named « *Style* » makes it possible to enumerate the following styles: *"Publish-Subscribe"*, *"Client-Server"*, *"SOA"* or another architectural style. Finally, we obtain, through vertical and horizontal successive refinements, a meta-model based on a named application with a precise architectural style.

3.3 Architecture Adaptability

Adaptability is the property of being able to modify the topology of an application in terms of components and connections. The adaptability modeling approach was performed by generating model transformation operations. Each transformation corresponds to a possible refinement/abstraction. (Kallel et al., 2012) proposed a UML profile to describe software architectures. Thus, we propose to adopt the notation proposed by (Kallel et al., 2012). The model is composed of five sections: The name of the operation to perform, « Require&Delete » (the part of the system to remove during the operation), « Insert » (the part of the system to create during the operation), « Require&Preserve » (the part not changed during the operation),and the pre-conditions that must be verified so that the operation can be performed. We suggest adopting the notation proposed by the authors and use the reconfiguration operations model to specify refinement/abstraction rules. The adaptability modeling approach was performed by four basic operations, namely adding a component, removing a component, adding a connection, and deleting a connection. We note that a model refinement executes the « Insert » transformation operation to add new components and connections. Moreover, a model abstraction executes

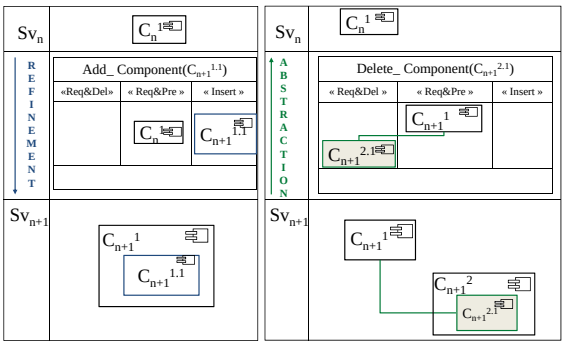

Figure 3: Refinement/Abstraction between vertical scales.

the « Require&Delete » transformation operation to remove components and connections.

Thus, we propose the following rules: If a new component is to be added at a given vertical scale Sv_n, we can estimate at the refined scale Sv_{n+1} these components will be added by inserting new composites that depend on it. If an existing composite needs to be deleted at the vertical scale Sv_{n+2} , we just need to see which components compose it at the higher level Sv_{n+1}, and hence, may be subject to change by deleting it at Sv_{n+1} and changing the component type composition at Sv_n (Figure3). If these components only impact the Sv_{n+2} as independent composite, it means that it can be deleted at this scale, and the system is easily adapted to the given change. Moreover, adding a new connection in Sv_n/Sh_m leads to adding a connection in Sv_n/Sh_{m+1} between the two components. Deleting a connection in Sv_n/Sh_{m+1} causes the removal of the associated connection between its component types in Sv_n/Sh_m (Figure4). During the refinement process, we apply the rules related to adding new components or connections using the « Insert » transformation operation.

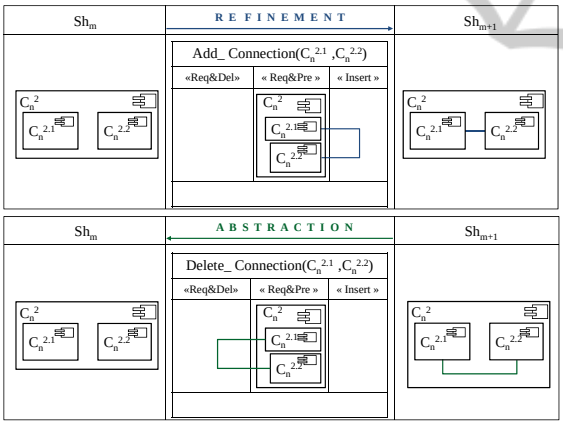

Figure 4: Refinement/Abstraction between horizontal scales.

The first operation named *Add-Component* ($C_{n+1}^{1.1}$) presents, in the « Insert » part, an instance of the Component to add to the system. In this operation we have nothing to remove, so the « Require&Delete » part is empty. In addition to the inclusion of this component, we need to preserve an instance of the component type (C_n^1) in the « Require&Preserve » part.

The second operation named *Add-Connection* ($C_n^{1.1}$, $C_n^{2.2}$) presents, in the « Insert » part, an instance of the the connection between two Components to add to the system. We conserve an instance of the components type $C_{n+1}^{1.1}$ and $C_{n+1}^{2.2}$ in the « Require&Preserve » part.

During the abstraction process, we apply the rules related to deleting components or connections using the « Require&Delete » transformation operation. The operations *Delete-Component* ($C_{n+1}^{2.1}$) and *Delete-Connection* ($C_n^{2.1}$, $C_n^{2.2}$) are shown in the « Require&Delete » part. Thus, the presented approach renders the adding/deleting of such structural elements as refinement/abstraction rules which supports tracing and adaptation.

3.4 Structural Intrinsic Properties

The multi-scale approach aims to verify architectural intrinsic properties. To refine the approach, we choose the refinement of a subset of components that are considered important for the validation of a given property. We explore the multi-scale approach to facilitate traceability in software architecture. We suppose that traceability supports alignment between scales. To ensure traceability in a multi-scale architecture, we propose two approaches : an approach with overlap between scales, and an approach with scale separation.

3.4.1 Approach with Overlap between Scales

This approach defines rules for component identification. If we keep track of a component, the traceability is trivial, and the identification of the component is preserved. We propose this notation for a component name C_n^m where n represents the scale number (n $\geq$ 0), and m represents a cursor on the current component (m $\geq$ 0). It can be decomposed in the next scale. For example, if we have a component C_n^1, then its composite in the next scale will obtain the name $C_{n+1}^{1.1}$.

3.4.2 Approach with Scale Separation

This approach defines rules for decomposing links between components. If a link is divided according to its identifiers, then a trace of the link decomposition is added. A link between two components C_n^1 and C_n^2 in Sv_n can be transformed into a connection assembly in Sv_{n+2} extending (from the source $C_{n+1}^{2.1}$ to the target $C_{n+1}^{1.1}$) or (from the source $C_{n+1}^{1.1}$ to the target $C_{n+1}^{2.1}$). It can also be transformed into two connections of assembly in Sv_{n+1}: One part from the source $C_{n+1}^{2.1}$ to the target $C_{n+1}^{1.1}$, while other part from the source $C_{n+1}^{1.1}$ to the target $C_{n+1}^{2.2}$. It can finally be transformed into a double connection of assembly in Sv_{n+1}, connecting ($C_{n+1}^{2.1}$ and $C_{n+1}^{1.1}$) or ($C_{n+1}^{2.2}$ and $C_{n+1}^{1.1}$). The structural intrinsic properties, previously described, are applied onely during the refinement process.

4 CASE STUDY: ERCMS

To show the viability of our solution, we present the modeling of an Emergency Response and Crisis Management System (ERCMS). The ERCMS involves structured groups of participants who are communicating and cooperating to achieve a common mission (e.g. save human lives, fight against a fire, etc). The ERCMS activities are based on information exchanges between mobile participants collaborating to achieve their mission. We define the following participant roles (Figure 5): a supervisor, coordinators, and investigators. The supervisor manages coordinators, supervises the application, and expects all reports. The coordinator leads a section of investigators and is expected to collect information received from investigators and diffuse them to the supervisor. The investigator acts to help, rescue and repair. Communication relationships between participants evolve throughout the mission.

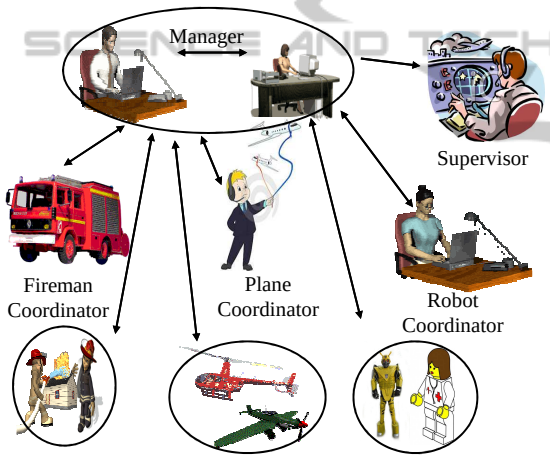

Figure 5: ERCMS case study.

4.1 Illustration via the ERCMS

Based on this case study, we perform the transition from the style level, presented in (Figure 2) through UML meta-models, to the instance level (Figure 6). We applied successive refinements and implemented the previously described model transformation rules. We obtained then the following results: In Sv_0, we apply the R_0 rule to define the application named “ERCMS" .

We obtain the level Sv_1 through the « Insert » transformation operation (Figure 3). In fact, participants in the ERCMS are represented (in Sv_1/Sh_0) by their components, named supervisor, coordinator, investigator, and manager. Those participants communicate with each other via the manager. Those relationships are represented (in Sv_1/Sh_1) as UML associations while applying the “Insert" transformation operation (Figure 4).

Finally, we illustrate instances obtained in Sv_2: Sv_2/Sh_0 presents two composites of type coordinator and investigator, only one manager and only one supervisor. We specify the role of each component of the application. Thus, the supervisor of the mission plays the role of a component consuming events and will be defined as “Consumer". In the same way, we specify the roles of each type. A manager is the “Events-service", an investigator and a coordinator have the role of “Producer-Consumer". Sv_2/Sh_1 illustrates the list of the ports for each component. Indeed, coordinators and investigators communicate with each other symmetrically as peers, adopting a protocol that allows a bidirectional flow of communication. So, we choose the topology *“Network-Dispatcher-AcyclicPeerToPeer"*. Sv_2/Sh_2 assigns to each port an interface of the type provided or required according to the type of service. Sv_2/Sh_3 establishes connections according to this topology and defines the style “Publish-Subscribe".

Finally, we reach a fine-grain description at scale Sv_2/Sh_3. The ERCMS was presented as a whole. The refinement steps allow us to compose the application, to extract all design details which are related to complexity, and to establish clear configurations.

5 CONCLUSION

In this paper, we have presented several studies related to the multi-level, multi-layer, multi-view , refinement architecture concepts. We have introduced a multi-scale modelling approach for multi-level architectures based on UML component diagrams and supporting adaptability management. We presented the approach at the conceptual level and proposed UML notations at the architectural style level. We have also presented a description of the instance level through a case study. We have then presented the rules of refinement/abstraction through model transformation techniques. Finally, we have provided verification rules of intrinsic properties for model traceability. We are currently working on the improvement of the validation scope of our approach based on a notational correction through XML and semantic correction. In our future work, we expect to apply our multi-scale approach to other use cases for modeling complex system architectures (e.g. Systems of Systems).

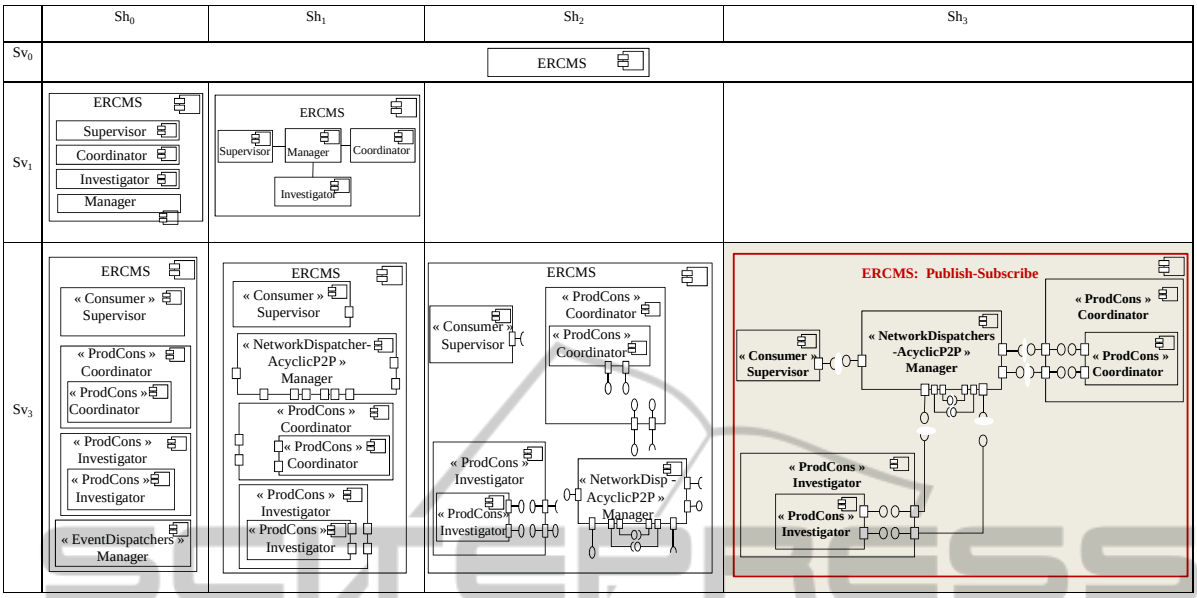

Figure 6: Illustration via the ERCMS.

REFERENCES

Anwar, A., Ebersold, S., Coulette, B., Nassar, M., and Kriouile, A. (2010). A rule-driven approach for composing viewpoint-oriented models. *Journal of Object Technology*, 9(2):89–114.

Baresi, L. and Guinea, S. (2013). Event-based multi-level service monitoring. In *ICWS 2013 IEEE 20th International Conference on Web Services*, pages 83–90.

Bouassida, I., Drira, K., Chassot, C., Guennoun, K., and Jmaiel, M. (2010). A rule-driven approach for architectural self adaptation in collaborative activities using graph grammars. *Int. J. Auton. Comp.*, 1(3):226–245.

Brosch, F., Buhnova, B., Koziolek, H., and Reussner, R. (2011). Reliability prediction for fault-tolerant software architectures. In *QoSA/ISARCS*, pages 75–84.

Ferrucci, F., Nota, G., Pacini, G., Orefice, S., and Tortora, G. (1992). On the refinement of logic specifications. *International Journal of Software Engineering and Knowledge Engineering*, 02(03):433–448.

Juan, Z., Xiaojuan, B., Qiang, L., Jie, C., and di, W. (2010). A component-based method for software architecture refinement. In *Proceedings of the 29th Chinese Control Conference*.

Kallel, S., Hadj Kacem, M., and Jmaiel, M. (2012). Modeling and enforcing invariants of dynamic software architectures. *Software and System Modeling*, 11(1):127–149.

Khlif, I., Hadj Kacem, M., and Drira, K. (2012). An approach to multiscale and multi points of view description for dynamic software architectures. In *CAL 6ème Conférence francophone sur les architectures logicielles*, pages 162–168.

Khlif, I., Hadj Kacem, M., Hadj Kacem, A., and Drira, K. (2014). A multi-scale modelling perspective for SoS architectures. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW14, pages 30:1–30:5.

Lange, S. and Middendorf, M. (2010). Multi-level reconfigurable architectures in the switch model. *Journal of Systems Architecture - Embedded Systems Design*, 56(2-3):103–115.

Mallet, F., Lagarde, F., André, C., Gérard, S., and Terrier, F. (2010). An automated process for implementing multilevel domain models. In van den Brand, editor, *Software Language Engineering*, volume 5969, pages 314–333. Springer.

Rafe, V., Reza, M., Miralvand, Z., Rafeh, R., and Hajiee, M. (2009). Automatic refinement of platform independent models. *IEEE International conference on computer technology and development*, pages 397–411.