Open Archive TOULOUSE Archive Ouverte (OATAO)

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 16840

The contribution was presented at SERA 2016 :
http://www.acisinternational.org/sera2016/

To cite this version : Khlif, Ilhem and Hadj Kacem, Mohamed and Stolf, Patricia and Hadj Kacem, Ahmed *Software architectures: multi-scale refinement.* (2016) In: 14th International Conference on Software Engineering Research, Management and Applications (SERA 2016), 8 June 2016 - 10 June 2016 (Hammamet, Tunisia).

Any correspondence concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

Software Architectures: Multi-Scale Refinement

Ilhem KHLIF[1,2,3,4], Mohamed HADJ KACEM[1], Patricia STOLF[4] and Ahmed HADJ KACEM[1]

[1] University of Sfax, ReDCAD Research Laboratory, Sfax, Tunisia
[2] CNRS, LAAS, 7 avenue du colonel Roche, F-31400 Toulouse, France
[3] Univ de Toulouse, UPS, LAAS, F-31400 Toulouse, France
[4] University of Toulouse, IRIT, France
ikhlif@laas.fr, mohamed.hadjkacem@isimsf.rnu.tn, patricia.stolf@irit.fr, ahmed.hadjkacem@fsegs.rnu.tn

***Abstract*—We propose a multi-scale modeling approach for complex software system architecture description. The multi-scale description may help to obtain meaningful granularities of these systems and to understand and master their complexity. This vision enables an architect designer to express constraints concerning different description levels, oriented to facilitate adaptability management. We define a correct-by-design approach that allows a given abstract architectural description to be refined into architecture models. We follow a progressive refinement process based on model transformations; it begins with a coarse-grain description and ends with a fine-grain description that specifies design details. The adaptability property management is performed through model transformation operations. The model transformation ensures the correctness of UML description, and the correctness of the modeled system. We experimented our approach with a use case that models a smart home system for the monitoring of elderly and disabled persons at home.**

Keywords—Software architecture, multi-scale modeling, refinement, UML notation, model transformation rules, adaptability.

I. INTRODUCTION

Software architecture design is an important research area in software engineering, for mastering the costs and the quality of the development of large and complex software systems. The design of a software architecture based on a multi-level description is still a complex task. This study investigates how to model software architectures to facilitate their validation at different description levels. To solve this problematic, we propose to consider different architecture descriptions with different levels of modeling details: "the scales". Our objective is to provide solutions for modeling software architectures to facilitate their validation at different description levels. We present a multi-scale description at design time for complex software systems. The multi-scale description may help to obtain meaningful granularities of these systems and to understand and master their complexity. This vision enables an architect designer to express constraints concerning different description levels, oriented to facilitate adaptability management. We define a correct-by-design approach that allows a given abstract architectural description to be refined into architecture models. The proposed design approach is founded on UML notations and uses component diagrams. The diagrams are submitted to vertical and horizontal transformations for refinement, for reaching a fine-grain description representing the necessary details that characterize the architectural style. Our approach is organized around a set of vertical and horizontal refinement rules. The adaptability property management is performed through model transformation operations. The model transformation ensures the correctness of UML description, and the correctness of the modeled system. In order to show the viability and usefulness of our solution, we experiment our methodological design to deal with a complex system dedicated to the smart home for monitoring elderly and disabled persons at home. The remainder of the paper is organized as follows. We describe our contribution in section II. Section III illustrates the use case. In section IV, we present a survey of related work. We conclude and outline some perspectives in section V.

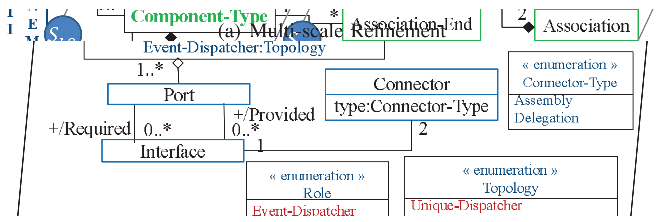
II. CONTRIBUTION

This work refines previous work by the authors on a multi-scale modeling approach for software architectures. In [6], the multi-scale modeling solution has only considered 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. We have proposed a multi-scale modeling perspective for Systems of Systems (SoS) architecture description. We have focused on SysML (System modeling language) notations and used block diagrams. To generalize the approach, this paper considers unfixed number of scales and proposes notations and common generic rules at all scales. We extend the approach by considering the refinement process as a vertical and horizontal model transformation, and by adding more details on scales. Reaching a fine-grained description that contains all necessary details that characterize the architectural style will trigger the stop condition and decide the last scale. We have proposed, in [7], a hybrid approach. The top-down approach was 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. This paper details only the top-down approach to study the multi-scale nature of complex software systems. We develop a specific solution modeling multi-scale software architecture that focuses on reference architectures in particular the Publish-Subscribe style.

A. Multi-scale modeling approach

We illustrate the stepwise refinement (or the top-down modeling) by a graphical representation shown in (Figure 1(a)). We present the multi-scale approach by a two-dimensional array describing vertical and horizontal scales. We define a vertical description scale "S_v" as a model that provides additional details of the design, which pertain to "S_{v+1}" and more levels of abstraction related to "S_{v-1}". A vertical scale can be further refined into several horizontal description scales

"$S_{v.h}$", thus providing more details. We consider that v (resp. h) represents the scale number (v,h $\geq$ 0). Colored `classes` represent the added details in each refined scale. 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 scale, there are several horizontal description scales. The first scale 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. 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. 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.



(a) Multi-scale Refinement

(b) Meta-model

Fig. 1: Multi-scale modeling approach

1) Model refinement rules: We detail the refinement between vertical and horizontal description scales through an algorithm written by the designer to guide the user (Algorithms 1, and 2).

a) Vertical Refinement: We, first, elaborate an initial abstract architecture description from the user requirements. The scale $S_{0.0}$ represents the entire application `Application` as a unique class. We initialise the application by a precise name. This is the beginning of traceability on which application requirements are specified. The first vertical refinement from S_0 to S_1 provides details on the whole system. $S_{1.0}$ includes a class `Component-Type` that specifies more than "1" component in the application. A `Component-Type` represents a subclass of an `Application`. The second vertical refinement is helpful for checking that S_2 consists of `Component-Type`, possibly composed of a sub-class named `Composite`.

b) Horizontal Refinement: At the same vertical scale S_1, an horizontal refinement allows the addition of associations between components type (at the scale $S_{1.1}$) A class `Association` is between at least "2" 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 "*". A stepwise horizontal refinement is needed in the scale S_2. We start by providing more details on data relating to the components. $S_{2.0}$ enumerates roles for each component and each composite. An enumeration named « *Role* » is used to

Algorithm 1: Refinement Step(1/2)

```
Input:  v, h = 0: integer; S: array[N][M] of (component
        diagram) /*$S_{v.h}$: Scale (vertical.horizontal)*/
BEGIN
while There are components to refine
/*Initialize $S_{0.0}$ by a unique component*/
do
    Identify the application name;
    S[v][h]$\leftarrow$Insert-Application(Application);
    /*Refine in a vertical way to obtain $S_{1.0}$*/
    for v from 1 to N-1 do
        for h from 0 to M-1 do
            Identify all component types in the application;
            while There are components to refine do
                S[v][h]$\leftarrow$Insert-Component-
                Type(Component-Type);
            end
            /*Refine in a horizontal way to obtain $S_{1.1}$*/
            Identify all links between all component types;
            S[v][h+1]$\leftarrow$Insert-Association(Component-
            Type,Composite);
        end
    end
end
```

enumerate roles that components can take. A component can play a role among this list: *"Event-Dispatcher", "Producer", "Consumer", "Consumer-Producer", "Server", "Service", etc.*

We are especially interested in refining the increasingly commonly used architectural style for component-based systems: the Publish/Subscribe style. The strength of this event-based interaction style lies in the full decoupling between producers, and consumers. This decoupling is provided by the event-dispatcher. This makes the publish/subscribe style relatively easy to add or remove components in a multi-scale architecture, introduce new events, register new consumers on existing events or modify the dispatchers. To generalize our approach, an added component is refined to play other roles for other styles (SOA, Client-Server, etc). This refinement is eventually guided by stylistic constraints to treat added details. For example, an enumeration called « *Service-Type* » can be defined at this scale to add details on the service related to the SOA style among which can have one of the types *"Discovery", "Registry", "Provider" or "Consumer"*. $S_{2.1}$ means the addition of communication ports and more details on previous enumerations. The class `Port` represents the point of interaction for a component. A `Component-Type` has one or more `Ports` that constitute interaction points with their environment. 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"*.

The scale $S_{2.2}$ allows the addition of interfaces in the scale S_2. The class `Interface` represents the interface of a component. An interface can have a type which is either provided *'+=Provided'* or required *'+=Required'*. Then, in $S_{2.3}$ we establish connections between components. 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

```
Algorithm 2: Refinement Step(2/2)
/*Refine in a vertical way to obtain $S_{2.0}$*/
for v from 2 to N-1 do
    for h from 0 to M-1 do
        Identify all Composites for each Component-Type;
        while There are components to refine do
            S[v][h]←Insert-Composite(Composite);
            Identify the role of component types;
            for Each Component-Type do
                Select Role from {Event-Dispatcher, Producer,
                Consumer, Producer-Consumer, Client, Server, Service};
            end
            /*Refine in a horizontal way to obtain $S_{2.1}$*/
            Identify all required and/or provided ports for all
            components;
            S[v][h+1]←Insert-Port(Component-Type,
            Composite);
            if Component-Type is Event-Dispatcher then
                Identify the topology of the event dispatcher;
                Select Topology from{Unique-Dispatcher ,
                Network-Dispatcher-Hierarchical,
                Network-Dispatcher-AcyclicP2P,
                Network-Dispatcher-GeneralP2P}
            end
            /*Refine in a horizontal way to obtain $S_{2.2}$*/
            Identify all required and/or provided interfaces of
            all ports;
            S[v][h+2]←Insert-Interface(Port);
            /*Refine in a horizontal way to obtain $S_{2.3}$*/
            Swith(Topology) {Identify all
            (Assembly/Delegation) connections between
            component types;
            S[v][h+3]←Insert-Connection(Interface);
            {Identify the style of the application;
            Select Style from{Publish-Subscribe, Client-Server, SOA};
        end
    end
end
```

"Assembly". For each connection, it is necessary to identify the source and the recipient by respecting the followed topology. $S_{2.3}$ 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. Algorithms 1, and 2 summarize the different steps detailed by refinements. Finally, we obtain, through vertical and horizontal successive refinements, a meta-model (Figure 1(b)) based on a named application with a precise architectural style. Thus, the proposed approach is guided by heuristics and adopted a rule-oriented description technique. The rules generate model transformation operations to ensure adaptability constraints. Each transformation corresponds to a possible refinement. The validation scope involves the correctness of UML description, and the correctness of the modeled system.

2) Adaptability management: Evolution of software systems is characterized by inevitable changes of software and increasing software complexity. Adaptability is the system's ability to easily accommodate changes (in terms of components and connections), explicitly in the requirements and early design phases, and maintain it during the entire lifecycle [4]. The adaptability management is performed through model transformation operations. Each transformation corresponds to a possible refinement. In [5], the authors proposed a UML profile to describe software architectures. Thus, we propose to adopt the notation proposed by [5]. The model is composed of five sections: The name of the operation to perform, « RequireAndDelete » (the part of the system to remove during the operation), « Insert » (the part of the system to create during the operation), « RequireAndPreserve » (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 rules. The application of a refinement rule is guided by specific rules written by the designer and through constraints based on interface compatibility to ensure consistency between scales. The adaptability modeling approach was performed by both adding and removing components and connections. The model refinement executes the « Insert » transformation operation to add new components and connections. Moreover, a model abstraction executes the « Require&Delete » transformation operation to remove components and connections [7]. In this paper, we express two basic refinement operations, adding a component, and adding a connection (or an association). We propose to use the model transformation operations to specify refinement rules. We propose this notation for a component-type name C_v^m where v represents the vertical scale number (v $\geq$ 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-type C_v^1, then its composite in the next scale will obtain the name $C_{v+1}^{1.1}$.

a) Adding a composite transformation rule: If there is a new composite to be added at a given vertical scale S_{v+1}, we can estimate at the refined scale S_{v+2} this composite will be added to refine the component-type that depend on it. This transformation operation is used to insert an instance of the composite $C_{v+2}^{1.1}$. The modeling of this operation, with the UML notation, is given in (Figure2). The name of the operation is *Insert-Composite ($C_{v+2}^{1.1}$)*. In the « Insert » part, we present an instance of the Composite to add to the system. In this operation we have nothing to remove, so the « RequireAndDelete » part is empty. In addition to the inclusion of this composite, we need to preserve an instance of the component type (C_{v+1}^1) in the « RequireAndPreserve » part. The initial state is shown on the left (S_{v+1}). We apply this rule to obtain the result illustrated on the right (S_{v+2}).

b) Adding an association transformation rule: Adding a new association in the horizontal scale $S_{v.h}$ leads to adding an association in $S_{v.h+1}$ between the two component-types. The name of the operation is *Insert-Association ($C_{v+1}^{2.1}$, $C_{v+1}^{2.2}$)* as shown in (Figure3). This operation allows you to insert an association instance to bind between two such instances of component-types ($C_{v+1}^{2.1}$) and ($C_{v+1}^{2.2}$). To execute this operation, there must be an instance of such association in the « Insert » part. The insertion of an association requires the existence of ($C_{v+1}^{2.1}$) and ($C_{v+1}^{2.2}$) instances that should be represented in the « RequireAndPreserve » part. The initial state is shown on the left ($S_{v.h}$). We apply the rule to obtain the result illustrated on the right ($S_{v.h+1}$).

III. USE CASE

We are interested in studying the smart home system established for the home monitoring of elderly and disabled persons at home. The elderly and disabled person who wishes to stay in his own home rather than in a health care institution may be able to live a more independent life and may feel more autonomous, and self-sufficient. Several software solutions have been presented to specify architectures for the monitoring of the elderly using wireless sensor technologies and the control of the components in smart home as needed. The authors, in [2], designed a methodological approach dedicated to the intelligent management of the comfort and safety of persons at home. The main issue is to ensure efficient management of the optimized comfort, and the safety of the elderly.

A. Smart Home system description

We specify the essential information architecture and we illustrate, in (Figure 4), the participants in the smart home system. The monitoring center is composed of three systems:the Environment Control and Comfort Management, the Emergency Surveillance Center, and the Medical Surveillance Center. Thus, the players selected who interact with other entities of the system are: The Home Care Actor, who interacts with the monitoring center, by setting medical or emergency conditions. The Equipment, that includes sensors and house devices. The emergency surveillance center deals with critical situations that need an urgent intervention. This center controls emergency and crisis situations using the activity sensors. In order to track the presence and activity of the elderly person, infrared activity sensors can be installed in each room. Activity sensors include fall sensors, presence sensors, video camera and microphone. The sensors send urgent signals to the center which treat immediately the received information. Once the signal is correct and the situation is critical, the center call the SAMU dispatch center to react and help the person. The medical surveillance center monitors physiological sensors. To track the medical information of the elderly person, physiological sensors can be installed in the bed, the chair, and on the body to detect the O2 level, the blood pressure, and the weight. The functions are achieved by the Oximeter, the Pressure Sensor, and the Weight Scale Sensor, classified as physiological sensors. While there are problems, the center requires the medical assistant intervention. The doctor and the nurse can deal with the needed medical care. The comfort management and the environment control system guarantees a comfort life for the users which are the elderly person and his relatives. This center enables communications between users, control the environment sensors (Humidity and Temperature Sensors), and commands the house devices (Convectors, Air conditioners).

B. Smart home modeling

We applied successive refinements and implemented the previously described refinement algorithms with model transformation rules. Based on the case study, we perform the transition from the style level (presented in section II through UML meta-models) to the instance level (Figure 5). We obtained then the following results: In $S_{0.0}$, we apply the rule to define the application named "*SmartHome*" . We obtain the scale S_1 through the *"Insert-Application(Smart Home)*" transformation operation. In fact, participants in the smart home are represented (in $S_{1.0}$) by their components. We apply the *"Insert-Component-Type(HomeCare-Actor)*, *Insert-Component-Type(Equipement)*, *Insert-Component-Type(MonitoringCenter)*" operations and the "RequireAndPreserve(Smart Home)" operation. Those participants communicate with each other via the monitoring center. Those relationships are represented (in $S_{1.1}$) as UML associations while applying the "*Insert-Association(HomeCare-Actor, MonitoringCenter)*, "*Insert-Association(Equipement, MonitoringCenter)*" operations.

We also illustrate instances obtained in S_2: In $S_{2.0}$, we apply successive model transformation operations to add the following composites: *MedicalAssistant, EmergencyService, User, Physiological-Sensor, Activity-Sensor, Environemenrt-Sensor, House-Device, MedicalSurveillanceCenter, EmergencySurveillanceSystem, and EnvironementControlAndComfort-Management*. Then, we specify the role of each component of the application. The *MonitoringCenter* plays the role of an "*EventDispatcher*". The *HomeCare-Actor* and *Equipement* play roles of "*Producer-Consumer*" in the application. $S_{2.1}$ allows to add the list of the ports for each component. We briefly describe the list of required/provided services of the *HomeCare-Actor* component. The *MedicalAssistant* receives information about the patient's situation from the *Medical-SurveillanceCenter*, he manages the patient's medical care (provides) and return a report after the care. The *EmergencyService* receives information about a critical situation *EmergencySurveillanceCenter*, reacts to save the patient (provides), and return a report after the intervention. The *User* receives not only emergency and medical services but also comfort services like online communication or house device command provided by the *EnvironementControl And ComfortManagement* component. We define the propagation of events between participants, which are mediated through a network of dispatchers called Monitoring Center. We choose the "*acyclic-P2P*" topology. In fact, equipments and actors communicate with each other symmetrically as peers, adopting a protocol that allows a bidirectional flow of communication. $S_{2.2}$ assigns to each port an interface of the type provided or required according to the type of service. Finally, we indicate at the scale $S_{2.3}$ connections established according to the used topology and we define the "*Publish-Subscribe*" style as shown (in Figure 6(a)).

While there are still components to refine in the smart home application, we apply the algorithms instructions and we move to the scale S_3 to add necessary design details. Applying the model refinement to the smart home application shows that this is a complex software system. We focused on mastering the complexity description details through including the third scale. This scale has not only included new composites but also has detailed the information flow between them. Each added composite (e.g. the doctor) is important for the application design. We illustrate, (in Figure 6(b)), the last horizontal scale $S_{3.3}$ which contains more details on added composites (*Doctor, SAMU dispatch Center, Video Camera, etc*), their ports, interfaces, and connections. Finally, we reach a fine-grain description at scale $S_{3.3}$. The refinement steps allow us to compose the application, to extract all design details which are related to complexity, and to establish clear configurations.

IV. Related Work

Considerable research studies have been presented on the description of software architectures. Several works have focused on the use of UML for multi-level modeling. Brosch et al. [3] 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 authors presented a tool support that allows the architects to design the architecture with UML models. Anwar et al. [1] described a view-based modeling methodology. They developed the VUML (View-based Unified Modeling Language) that enables the modeling of a software system. Other studies have focused on the architecture refinement concept. Oquendo et al. [8] described Π-ARL, an architecture refinement language based on the rewriting logic. The core of Π-ARL is a set of architecture refinement primitives that supports transformation of architecture descriptions. The authors formally modeled the stepwise refinement of software architectures. Rafe et al. [9] 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 is designed by the rules of graph transformation.

Several studies have been performed on the modeling of multi-level architectures 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 the 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. In this work, we have considered that a given modeling level can be described by several scales. We needed a clear refinement process to transit between scales. This work aimed to provide solutions for modeling software architectures so as to facilitate their validation at different description levels.

V. Conclusion and perspectives

In this paper, we have presented a multi-scale approach for software architectures at the conceptual level and proposed UML notations at the architectural style level. We have then presented the rules of refinement/abstraction through model transformation techniques. We have also presented a description of the instance level through a case study. Finally, we have investigated the state of the art on how multi-level software architecture modeling has been addressed. We are currently working on the improvement of the validation scope of our approach based on a notational correction through XML and semantic correction. We are developing a support tool for the multi-scale approach. In our future work, we expect to apply our multi-scale approach to other case studies for modeling complex system architectures (e.g. Systems of Systems). Additional work is also needed to check architectural properties related to both structural and behavioral descriptions. We exect to provide structured representations of components behavior using other UML diagrams (e.g. sequence diagram).

Acknowledgment

The authors would like to thank University of Toulouse who permitted this research opportunity within the IDEX “chaire d’attractivite" delivered to Pr. Gene COOPERMAN.

References

[1] A. Anwar, T. Dkaki, S. Ebersold, B. Coulette, and M. Nassar. A formal approach to model composition applied to VUML. In *16th IEEE International Conference on Engineering of Complex Computer Systems, ICECCS 2011, Las Vegas, Nevada, USA, 27-29 April 2011*, pages 188–197, 2011.

[2] S. Bonhomme, E. Campo, D. Esteve, and J. Guennec. Methodology and tools for the design and verification of a smart management system for home comfort. In *Intelligent Systems, 2008. IS '08. 4th International IEEE Conference*, volume 3, pages 24–2–24–7, Sept 2008.

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

[4] I. Crnkovic, S. Larsson, and M. Chaudron. Component-based development process and component lifecycle. In *Software Engineering Advances, International Conference on*, pages 44–44, Oct 2006.

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

[6] I. Khlif, M. Hadj Kacem, A. Hadj Kacem, and K. Drira. 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, 2014.

[7] I. Khlif, M. Hadj Kacem, A. Hadj Kacem, and K. Drira. A UML-based approach for multi-scale software architectures. In *17th International Conference on Enterprise Information Systems (ICEIS)*, page (To appear), 2015.

[8] F. Oquendo. π-ARL: An architecture refinement language for formally modelling the stepwise refinement of software architectures. *SIGSOFT Softw. Eng. Notes*, 29(5):1–20, Sept. 2004.

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

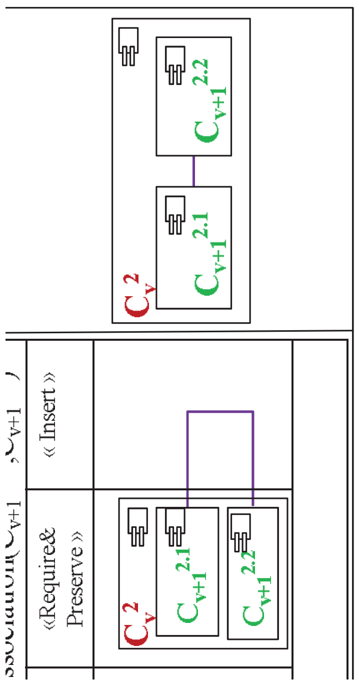


Fig. 3: Horizontal Refinement

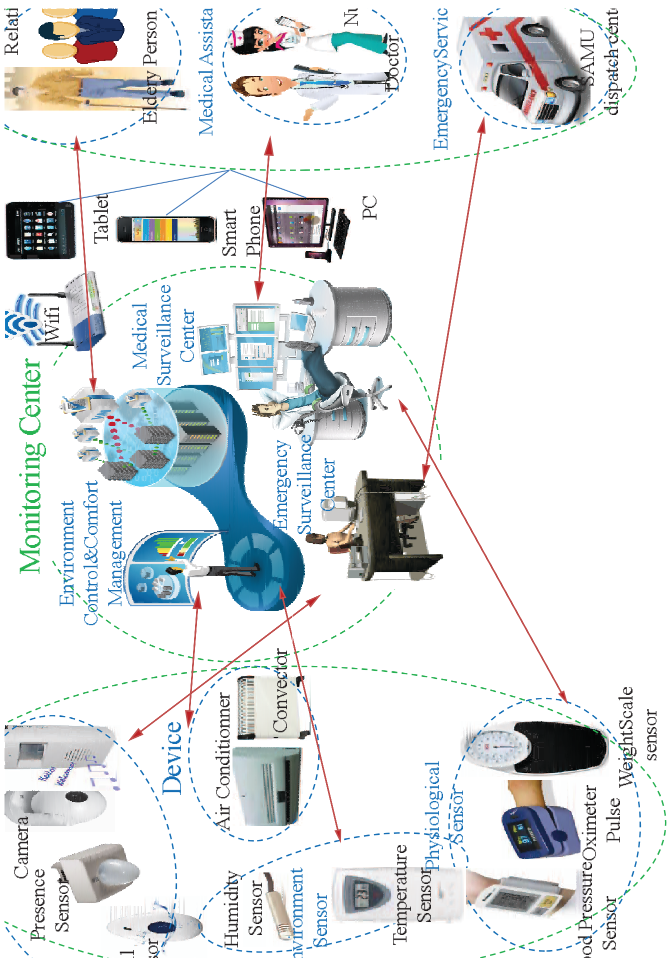
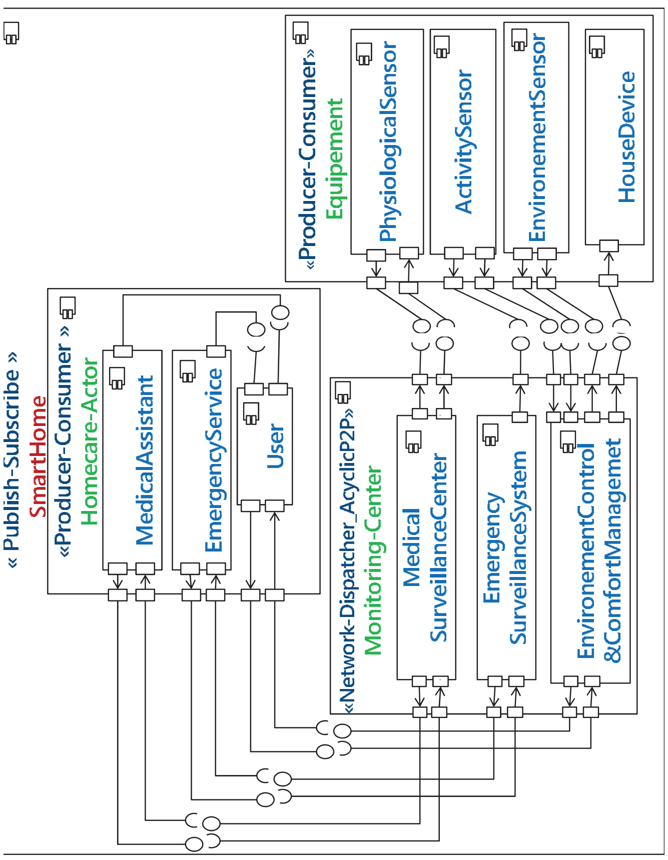
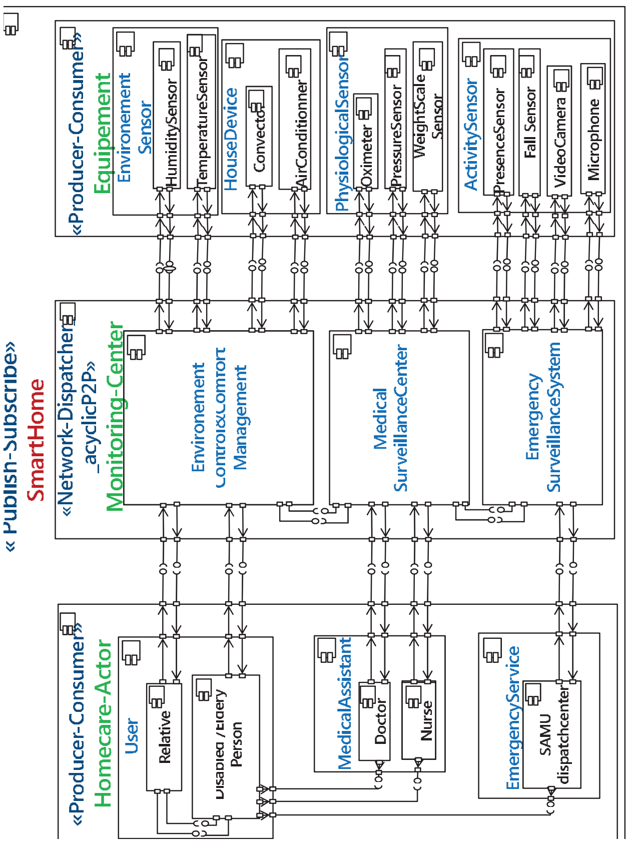


Fig. 4: Smart Home system



(a) Smart Home model at the scale $S_{2.3}$



(b) Smart Home model at the last scale $S_{3.3}$

Fig. 6: Smart Home model (2/2)

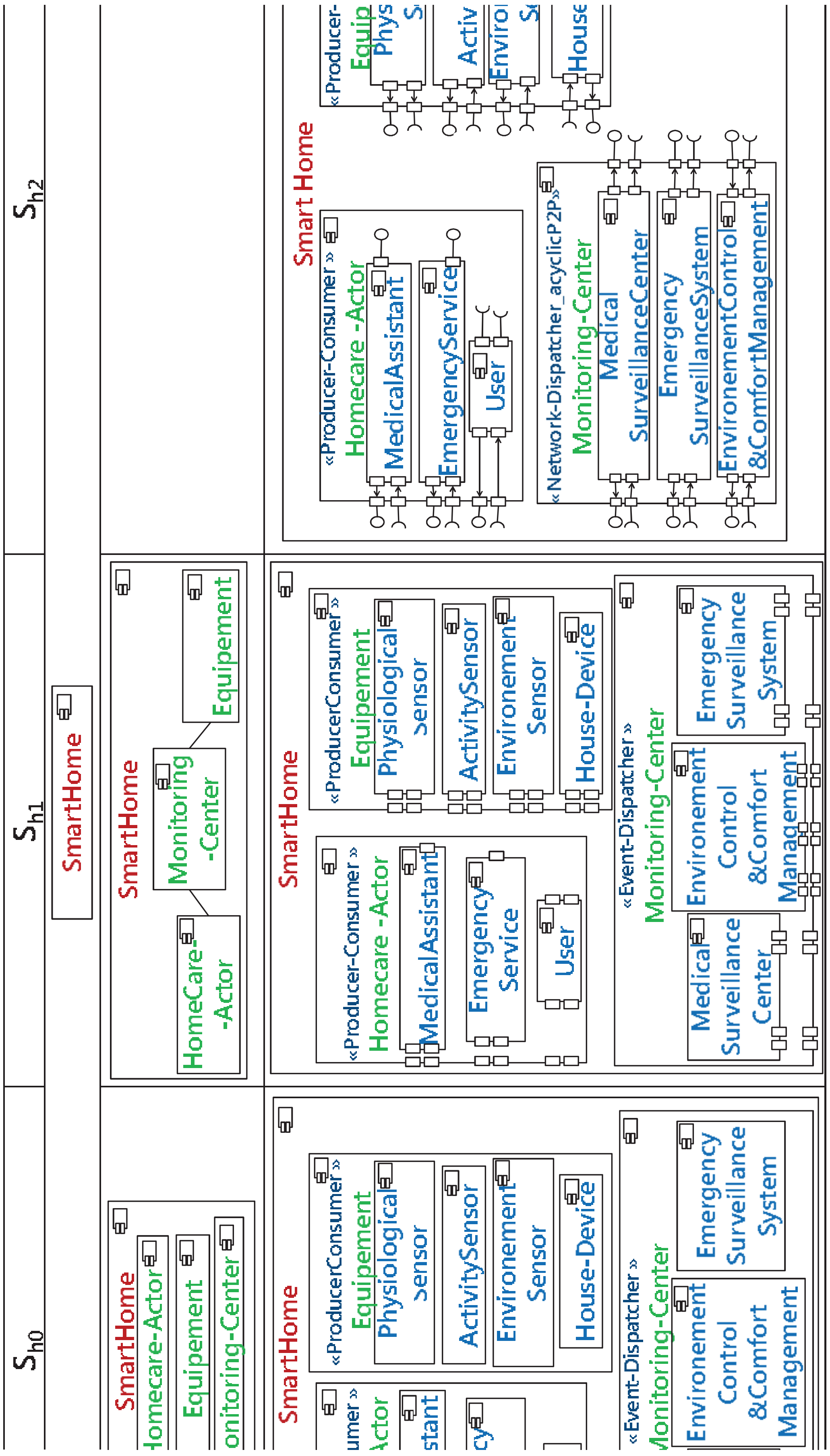


Fig. 5: Smart Home Model (1/2)