Modeling and Formal Specification Of Multi-scale Software Architectures

Ilhem Khlif[1,2,3], Mohamed Hadj Kacem[1], Khalil Drira[2,3] 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, LAAS, F-31400 Toulouse, France
ikhlif@laas.fr, mohamed.hadjkacem@isimsf.rnu.tn, khalil.drira@laas.fr, ahmed.hadjkacem@fsegs.rnu.tn

***Abstract*—Modeling correct complex systems architecture is a challenging research direction that can be mastered by providing modeling abstractions. For this purpose, we provide an iterative modeling solution for a multi-scale description of software architectures. We define a step-wise iterative process starting from a coarse-grained description, and leading to a fine-grained description. The refinement process involves both system-independent structural features ensuring the model correctness, and specific features related to the expected behavior of the modeled domain. We provide a visual notation extending the graphical UML (Uniform Modeling Language) notations to represent structural as well as behavioral features of software architectures. The proposed approach mainly consists of two steps. In the first step, the architecture is modeled graphically according to the UML notations. In the second step, the obtained graphical models are formally specified using the Event-B method. We implement the resulting models describing structural and behavioral properties using the Rodin platform and prove their correctness. We apply our approach for a methodological design of a smart home scenario for the homecare monitoring of disabled and elderly persons.**

Keywords–Software; Architecture; multi-scale; iterative; modeling; UML; formal; specification; structural; behavioral; refinement; Event-B.

I. INTRODUCTION

Software architecture design has become the key factor for the success 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 is a complex task. On the one hand, we have to describe the system with enough details for understanding without ambiguity and implementing in conformance with architects requirements and users expectations. On the other hand, we have to master the complexity induced by the increasing model details both at the human and automated processing levels. Some high level properties can be expressed on informal descriptions with a high level of abstractions and checked on simple formal descriptions. Some other properties need more detailed descriptions to be expressed and deep specifications to be elaborated. Description details may be application-independent and mainly structural such as component decomposition, or system-specific and mainly behavioral, such as message ordering in interaction protocols. An iterative modeling process that helps architects to elaborate complex but yet tractable and appropriate architectural models and specifications can be implemented by successive refinements. Different properties of correctness and traceability have to be maintained between the models and the specifications at the different levels of iterations. Providing Rules for formalizing and conducting such a process is our objective, which we implemented in visual modeling notations and formally specified in a formal description technique. For this purpose, we propose to consider different architecture descriptions with different levels of modeling details called **"the scales"**. We define a step-wise iterative process starting from a coarse-grained description and leading to a fine-grained description. The proposed approach mainly consists of two steps. In the first step, multi-scale architectures are modeled graphically using UML notations. In the second step, the obtained models are formalized with the Event-B method, and validated by its supporting Rodin platform [11]. In order to illustrate our solution, we experiment our approach with a case study dedicated to the smart home system for the homecare monitoring of elderly and disabled persons. The remainder of the paper is organized as follows. We describe the UML modeling approach in Section II. In Section III, we present the generated Event-b specifications. Section IV presents the case study. In Section V, we present a survey of related work. We conclude and outline some perspectives in Section VI.

II. ITERATIVE MODELING

At the level of abstraction, a software architecture is represented as a collection of interconnected components, and it is at this level that the structural and behavioral properties of software systems are addressed. We define multi-scale modeling as an incremental process where we constantly refine software systems descriptions. We propose to illustrate UML notations for describing software architectures at different description levels. In the first iteration, an abstract model is defined. At each iteration, design modifications are made and new details are added. We consider both structural and behavioral descriptions. In model-driven engineering, traceability links are established from the application requirements. The traceability links specify which parts of the design contribute to the satisfaction of each requirement [9].

A. Structural modeling

We propose structural modeling for describing software architectures using a visual notation based on UML. To describe the structure of a multi-scale architecture, we model the first scale by a given coarse-grained description using a UML component diagram. This model is refined until reaching a fine-grained description representing the necessary modeling details. We define a vertical description scale "$S_{v+1.h}$" as a model that provides additional details of the design, that pertain to "$S_{v.h}$" and more abstraction related to "$S_{v+2.h}$". A vertical scale can be further refined into several horizontal description scales ("$S_{v.h}$", "$S_{v.h+1}$",...) thus providing more details. We consider that v, resp. h, represents the vertical and

 ISBN: 978-1-61208-498-5

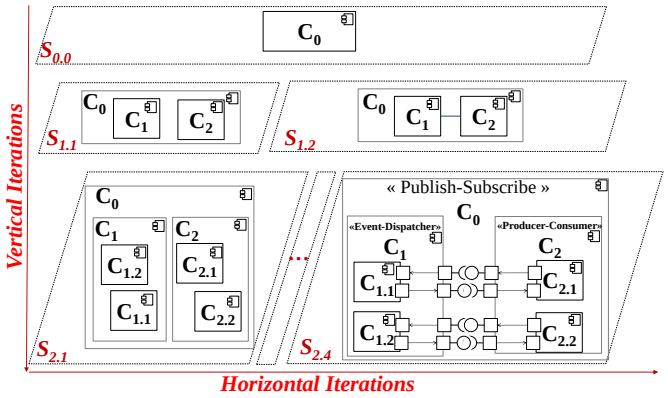

Figure 1. Structural modeling

horizontal iterations (v,h $\geq$ 0). We, first, elaborate an initial abstract architecture description from the user requirements. At the first scale $S_{0.0}$, application requirements are specified (a unique component C_0 is identified). This is the beginning of the traceability. A first vertical iteration from $S_{0.0}$ to $S_{1.1}$ is required in order to provide details on the application, and refine it with several components. In Figure 1, two components named C_1 and C_2 are added. At the same scale, an horizontal iteration is needed to specify the interactions between components. We represent a link between C_1 and C_2 in the scale $S_{1.2}$. A second vertical iteration is helpful for refining components with new sub-components, and checking that at the scale $S_{2.1}$, the components identification is preserved, as we keep traceability of a component from one scale to another. This notation is used for identifying a component: C_m where m represents a cursor on the current component (m $\geq$ 0). It can be decomposed in the next scale. The component C_1, is refined with two sub-components identified as $C_{1.1}$, $C_{1.2}$, etc. The component C_2 is refined with two sub-components ($C_{2.1}$ and $C_{2.2}$). Several horizontal iterations are needed in the second vertical scale to show more specific details (related to the UML description). An horizontal iteration called $S_{2.2}$ adds details on data relating to the components: roles are associated with components such as *"Event-Dispatcher"*, *"Producer"*, *"Consumer"*, *"Producer-Consumer"*, *"Client"*, *"Service"*, *etc.* The scale $S_{2.2}$ is inserting communication ports and more details; the scale $S_{2.3}$ allows the addition of component interfaces. Finally, we obtain the model $S_{2.4}$ where connections are established between components to define the architectural style of the application. In the illustrated example, we are limited on three vertical iterations to show the necessary details. However, the iterative process continues while there are still components to refine. The number of iterations depends on the application requirements. Each new iteration does not only include new sub-components but also adds necessary design details on the information flow between components. In the scale $S_{2.4}$, we propose to refine the interaction (link) between the two components C_1 and C_2 illustrated at the scale $S_{1.2}$ with respect to the following traceability constraints: if the component C_1 performs the role of an "Event-Dispatcher" and the component C_2 is a "Producer-Consumer", the link between C_1 and C_2 in $S_{2.1}$ will be decomposed into a double assembly connection in the scale $S_{2.4}$ connecting ($C_{1.1}$ and $C_{2.1}$). We preserve the model traceability from one scale to another by decomposing links, at the abstract scale, and refining them, at the next scale, to show possible connections established between components. Traceability is a desired characteristic for software management. However, it is not always possible to trace every design (or architectural) component back to requirements. To ensure this property, we check during the iterative process that the interface compatibility is preserved in the multi-scale architecture: First, we verify through added details on component roles that each required interface is associated with a producer component and each provided interface is associated with a consumer component. The main issue is to ensure the well-typed and the well-connected in UML component diagram. For this purpose, we have implemented a tool supporting our approach in visual modeling notation as an Eclipse plug-in to providethe designerwith an editor for UML modeling architecture. Using this editor, we make sure that refined models are correct by design. Second, we check the interface compatibility through constraints on different scales using the OCL (Object Constraint Language) interactive console associated with the Eclipse.

B. *Behavioral modeling*

To specify behavioral features, we use UML sequence diagram that provides a graphical notation to describe dynamic aspects of software architectures [7]. The application is ini-

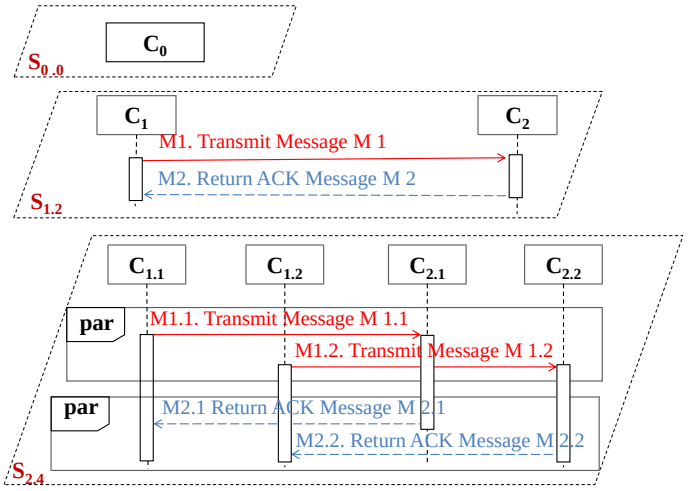

Figure 2. Behavioral modeling

tialized (at the first scale), and after successive iterations, the sets of components and interactions among them are identified in a way that supports the required behavior of the abstract application level. We describe the specified behavior of an application using the UML sequence diagram in Figure 2. The sequence diagram is helpful for describing the message ordering in interaction protocols during the iterative modeling. In the first scale, the whole application is presented as a black box to illustrate the System Sequence Diagram (SSD) named "C_0". The main issue here is to secure the message transmission and how elements cooperate to ensure correct information propagation. Several events may refine an abstract event: The single message (M_1) between actors in the scale (S_1) is refined with a set of messages ($M_{1.1}$, $M_{1.2}$, $M_{2.1}$, and

$M_{2.2}$) in the scale (S_2), or the content of translated messages depends on earlier received message. The sequence diagram, represented in Figure 2, specifies the behavioral features of the publish-subscribe architecture. When the Producer-Consumer component C_1 sends a message (M_1) to the Event dispatcher component C_2 at the scale S_1, the Event-dispatcher tracks this message and, it replies to the Event-dispatcher by sending an acknowledgement message (M_2). At the next scale S_2, the two messages will be refined into a parallel sequence of messages while keeping track of the type of message sent or received in the abstract scale.

Our approach is based on a multi-scale modeling that helps to automate the construction of correct design architectures. So, we need to specify the software architecture model that describes the software components and their composition. In fact, each model is represented as a set of scales, and each scale denotes a set of architectures. 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 using the Event-B specifications. After constructing the architectures of software architecture model, we apply the relation between the two models in order to obtain model-based architectures with different description levels.

III. EVENT-B FORMAL SPECIFICATION

The aim of formal modeling is to achieve a precise specification of the intended structures and behaviors in the design [1]. The advantage of such specifications is to determine whether a modeled structure can successfully satisfy a set of given properties derived from the user requirements. We consider here specifying a multi-scale architecture using the refinement-based formal method: the Event-B [11]. We use the Event-B method and its event based definition to formalize UML models. Our approach facilitates layering and mapping the informal requirements to traceable formal models. An Event-B model is made of two types of components: contexts and machines [2]. The obtained UML models are mapped to Event-B specifications: the component diagram constitutes the static part of the architecture, it is specified with the Event-B method in the Context part. The sequence diagram constitutes the dynamic part of the architecture, it is specified with the Event-B method in the Machine part. A context describes the static part of a model, and a machine describes the dynamic behavior of a model. Each context has a name and other clauses like "Extends", "Constants", "Sets" to declare a new data type and "Axioms" that denotes the type of the constants and the various predicates which the constants obey. Machines and contexts can be inter-related: a machine can be refined by another, can see one or several contexts, while a context can be extended by another [8]. A multi-scale software architecture is described with structural features and behavioral features. Structural features are specified with one or several contexts and behavioral features are specified with one or several machines.

A. Structural specifications

In the component diagram we specify components that constitute the architecture, their types and their connections. This diagram constitutes the static part of the defined architecture. It is specified in the Context part. In the first scale S_0, the graphical model is transformed into an Event-B specification called Context0. In the Context0, we specify the whole application with a Component as constants. The component, that composes the architecture at scale $S_{0.0}$, is named C_0. This is specified by using a partition in the AXIOMS clause (C0_partition).

CONTEXT
$Context0$
SETS
$Component$
CONSTANTS
$C0$
AXIOMS
C0_partition : $partition(Component, \{C0\})$
END

In the next scales, we use the refinement techniques to gradually add details until obtaining the final scale specification. A new context named Context1 extends the Context0 and specifies new components in the application. We define two components C_1 and C_2 as constants and the established link between them. Formally, links are specified with an Event-B relation between two components (Link_partition).

CONTEXT
$Context1$
EXTENDS
$Context0$
CONSTANTS
$C1, C2, Link$
AXIOMS
C1_partition : $partition(Component, C0, \{C1\}, \{C2\})$
Link_partition : $Link \in C1 \leftrightarrow C2$
END

A Context2 is extending the previous Context1, and is adding sub-components of each component. We specify the role of each component (producers, consumers and event-dispatcher) as constants. Connectors are specified with an Event-B relation between two components. The set of Connectors is specified formally with two partitions (Ct1_part, Ct2_part).

CONTEXT
$Context2$
EXTENDS
$Context1$
SETS
$Role$
CONSTANTS
$C1.1, C1.2, C2.1, C2.2, Prod,$
$Cons, EventDis, Prod1, Prodn,$
$Cons1, Consn, ED1, EDn, Connector1, Connector2$
AXIOMS
C2_partition :
$partition(Component, \{C1.1\}, \{C1.2\}, \{C2.1\}, \{C2.2\})$
Ct1_part : $Connector1 \in C1.1 \leftrightarrow C2.1$
Ct2_part : $Connector2 \in C1.2 \leftrightarrow C2.2$
Role_part :
$partition(Role, \{Prod\}, \{Cons\}, \{EventDis\})$
$Cons = \{C1, .., Cn\} \wedge C1 \neq C2 \wedge ..\wedge \neq Cn$
$EventDis = \{ED1, .., EDn\} \wedge ED1 \neq ED2 \wedge ..\wedge \neq EDn$
$Prod = \{P1, .., Pn\} \wedge P1 \neq P2 \wedge ..\wedge \neq Pn$
END

B. Behavioral specifications

The Event B machine is used formally, to find structural errors and to verify the semantic of the UML model. To specify behavioral features, we specify the abstract description scale with a machine at a high level of abstraction. Then, we add all

necessary details to the first machine by using the refinement process. In the first machine, we only specify the modeled application by extending Context0.

MACHINE
$Machine0$
SEES
$Context0$
VARIABLES
$C0$
INVARIANTS
inv : $C0 \in BOOL$
EVENTS
INITIALISATION
BeginAct
$act : C0 := TRUE$
EndAct
END

Machine1 is a refinement of the Machine0, using the context Context1 and adding communication between the components C_1 and C_2. The behavior is described as follows: the component C_1 sends a Message to the component C_2. When the component C_2 becomes available, it receives the Message, processes it and sends the Acknowledgement Message. When the component C_1 becomes available, it receives the ACK-Message. The invariants (Send_Message, Receive_Ack) specifies what is the sent message, who is the sender and the receiver. The Machine1 has a state defined by means of a number of variables and invariants. Some of variables can be general as the variable Send, which denotes the sent message and the variable Receive, which denotes the received message. The variable Send is defined with the invariant (Send_Msg) which specify that Send is a relation between two components so that the sender, the receiver and the message are known.

MACHINE
$Machine1$
REFINES
$Machine0$
SEES
$Context1$
VARIABLES
$Send, Receive$
INVARIANTS
Send_Message : $Send \in BOOL$
Receive_ACK : $Receive \in BOOL$
EVENTS
INITIALISATION
BeginAct
$act1 : Send := FALSE$
$act2 : Receive := FALSE$
EndAct
EVT
$init1 : Send \in C1 \rightarrow C2$
$init2 : Receive \in C1 \rightarrow C2$
$init3 : Transmit := C1 \rightarrow True, C2 \rightarrow False$
END

We follow the same method to specify a second machine named Machine2 which refines Machine1, using the context Context2 and adding communication between the sub-components $C_{1.1}$, $C_{1.2}$, $C_{2.1}$ and $C_{2.2}$. The invariants (SendMsg1.1 , SendMsg1.2 , ReceiveAck2.1 , ReceiveAck2.2) are specified in the INVARIANTS clause to check that each sub-component can't send a message or receive an acknowledgment only if it is authorised.

MACHINE
$Machine2$
REFINES
$Machine1$
SEES
$Context2$
VARIABLES
$SendMsg1.1, SendMsg1.2, ReceiveAck2.1, ReceiveAck2.2$
INVARIANTS
Send_Message : $SendMsg1.1, SendMsg1.2 \in BOOL$
Receive_ACK : $ReceiveAck2.1, ReceiveAck2.2 \in BOOL$
EVENTS
INITIALISATION
BeginAct
$a1 : SendMsg1.1 := FALSE$
$a2 : SendMsg1.2 := FALSE$
$a3 : ReceiveAck2.1 := FALSE$
$a4 : ReceiveAck2.2 := FALSE$
EndAct
EVT
$init1 : SendMsg1.1 \in C1.1 \rightarrow C2.1$
$init2 : SendMsg1.2 \in C1.2 \rightarrow C2.2$
$init3 : ReceiveACK2.1 \in C1.1 \rightarrow C2.1$
$init4 : ReceiveACK2.2 \in C1.2 \rightarrow C2.2$
$init5 : transmit1 := C1.1 \rightarrow True, C2.1 \rightarrow False$
$init6 : transmit2 := C1.2 \rightarrow True, C2.2 \rightarrow False$
END

The Event-B machine is used formally, to find structural errors and to verify the semantic of the UML model. Besides, behavioral properties are checked like liveness and reachability. The reachability means that the components are able to capture all exchanged messages. We formulate those properties as predicates (INVARIANTS, AXIOMS). We check that each component only sends a message if it is authorised. This is controlled by the invariants (Send-Msg, Receive-ACK). Reaching the last scale description by using refinement techniques, we guarantee that refined models are not contradictory and we ensure that they are correct by design. The multi-scale modeling helps to automate the construction of correct design architectures. The aim is to derive those UML models by applying correctness preserving transformations, i.e. refinements, that conform to the constraints defined by the application and by the adopted architecture styles. The refinement techniques proposed by this method allow to represent architectures at different abstraction levels and are implemented using the Rodin platform.

IV. Application to the Smart home

This section focuses on modeling the smart home system for the homecare monitoring of elderly and disabled persons. The main issue is to ensure efficient management of the optimized comfort, and the safety of the elderly and disabled person at home [5]. We illustrate, in Figure 3, the constituent elements of the smart home application. The monitoring center is composed of three systems: the Environment Control and Comfort Management, the Emergency Surveillance Center, and the Medical Surveillance Center. The Home Care Actor interacts with the monitoring center, by setting medical or emergency conditions; the Equipment includes sensors and house devices; the emergency surveillance center controls critical situations using the activity sensors. Activity sensors include fall sensors, presence sensors, video camera and microphone. The medical surveillance center monitors physiological sensors. While there are problems, the center requires the medical assistant intervention (the doctor, the nurse). The comfort management and the environment control system guarantees a comfort life for the users. This center enables communications between users, control the environment sensors (Humidity

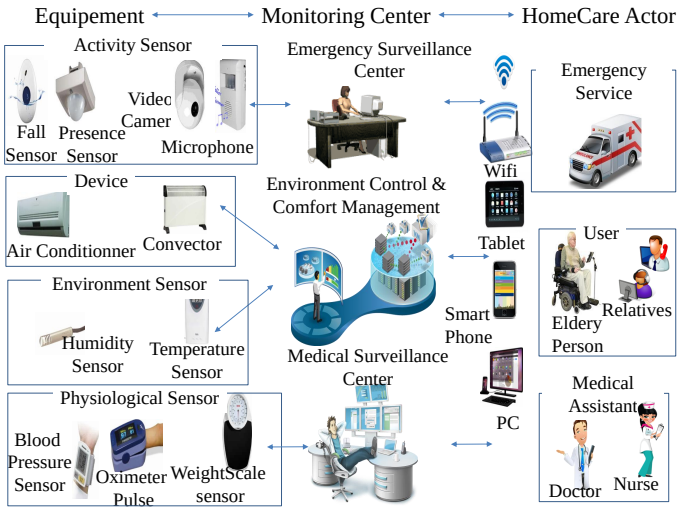

Figure 3. Smart Home application

and Temperature Sensors), and commands the house devices (Convectors, Air conditioners).

A. Smart Home Model

We experiment our approach by applying successive iterations to the smart home application. We obtained then the following results: In $S_{0.0}$, we define the application named "*SmartHome*". The constituent systems of the smart home are described (in $S_{1.1}$): *HomeCare-Actor*, *Equipment*, and *MonitoringCenter)*. Those systems communicate with each other via the monitoring center. Those relationships are represented (in $S_{1.2}$) as links. In Figure 4, We illustrate the iterative process applied to the smart home system. In the next scale, the three components are refined and specified with an associated role as shown in Figure 4. The *MonitoringCenter* plays the role of an "*EventDispatcher*". The *HomeCare-Actor* and *Equipment* play the role of "*Producer-Consumer*" in the application. 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 *MedicalSurveillanceCenter*, he manages the patient's medical care (provides) and returns a report after the care. The *EmergencyService* receives information about a critical situation *EmergencySurveillanceCenter*, reacts to save the patient (provides), and returns 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. During the iteration process, we apply the link decomposing rule with respect to the component role: if C_1 plays the role of an "Event-dispatcher" and C_2 acts as a "Producer-Consumer", the link in the scale $S_{1.2}$ between C_2 which is related to "HomeCareActor" or "Equipment" and C_1 in the scale $S_{1.2}$ will be decomposed into a double assembly connection in the scale $S_{2.4}$ between $C_{1.1}$ which is related to "MonitoringCenter" and $C_{2.1}$ which is related to 'HomeCareActor" or "Equipment". While there are still components to refine in the smart home, we move to the third scale to add more design details. We focused on mastering the system complexity description details through including the third scale. This scale has not only included new sub-components but also detailed the information flow between them. Each added sub-component (e.g. the doctor) is important for the design process. It influences the abstract level where smart home requirements are specified. We illustrate the last horizontal scale $S_{3.4}$ adding new sub-components (*Doctor, Emergency Service, Video Camera, etc*), and their connections.

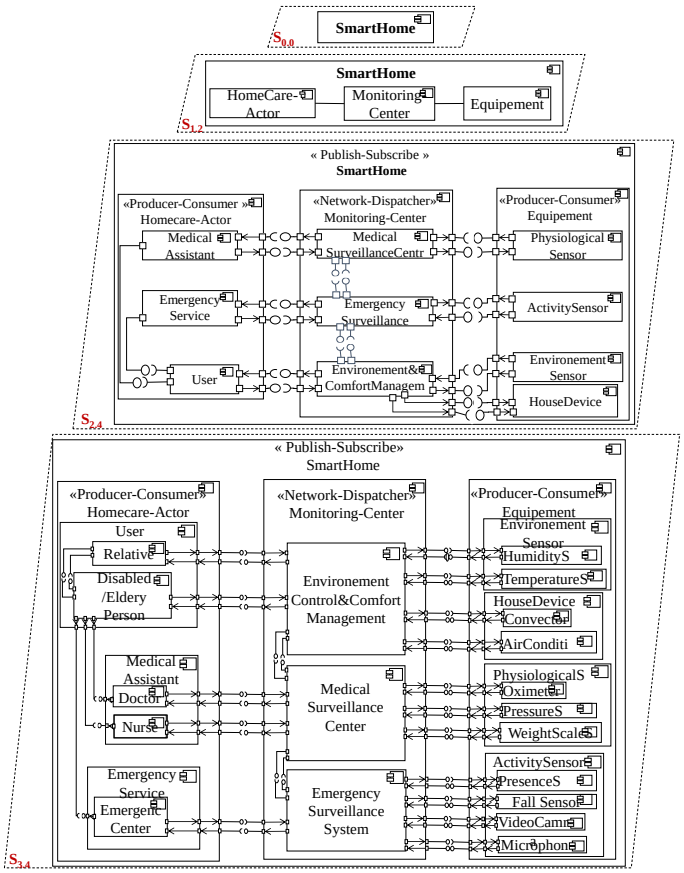

Figure 4. The Smart Home model

B. Smart Home system-specific properties

In Figure 5, we present one of three fragments of the UML sequence diagram to demonstrate the behavior of constituent elements. The sequence diagram shows the instances participating in the interaction having two dimensions: the vertical dimension represents time; the horizontal dimension represents different objects which is related to the behavior of the smart home components. We illustrate the first scale $S_{0.0}$ using the SSD named "Smart Home" to show the whole system (as a black box). A vertical refinement called $S_{1.2}$ allows to describe the objects *HomeCare-Actor*, *Equipment*, and *MonitoringCenter)* and the exchanged messages in the diagram "Sd Monitoring". An object of class *Equipment* starts the behavior by sending an alert message to an object of class *MonitoringCenter)*. This object responds by an acknowledgment message to the equipment and sets the Sleep mode. The monitoring center sends the information to the object *HomeCare-Actor* that will respond immediately and send return message describing the situation after the care.

 ISBN: 978-1-61208-498-5

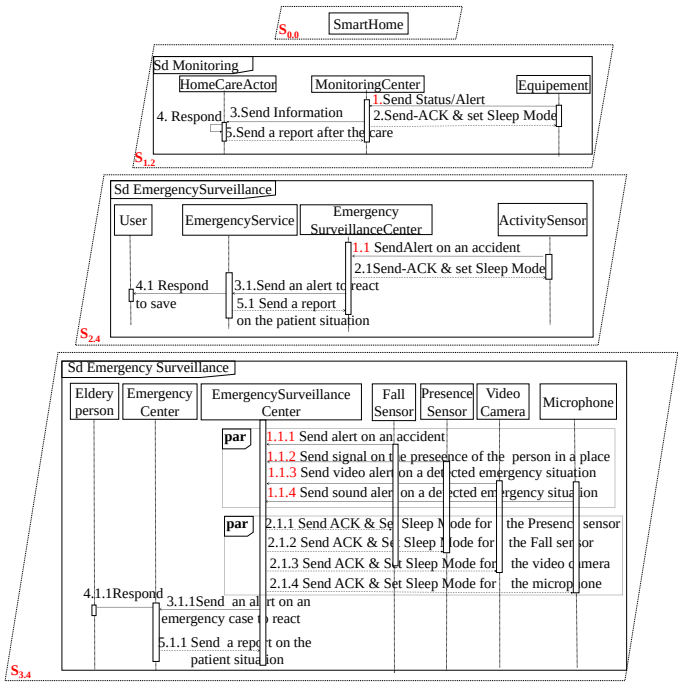

Figure 5. Fragment of the UML Sequence Diagram

C. Event-B specifications

We apply the Event-B refinement techniques to check the correctness of the multi-scale architecture applied to the Smart Home. We illustrate the Context2 that is extending the previous Context1, and is adding all sub-components in the smart home. They are specified with three partitions: equipment-partition, Monitoring-partition and Actor-partition. We specify in the Context2 the components type role (producer-consumers and event-dispatcher) as constants. There are many connections between components. The Connectors are specified with constants in the CONSTANTS clause. The set of Connectors is composed of all Connectors. This is specified formally with a partition (Connector-partition).

CONTEXT
$Context2$
EXTENDS
$Context1$
CONSTANTS
$ActSensor, Device, EnvSensor, PhysSensor,$
$EmerSurvCenter, EnvControl, MedSurvCenter,$
$User, MedAssistant, EmerService, Connector1, ..$
AXIOMS
Eq_partition :
$partition(Component, \{ActSensor\}, \{Device\},$
$\{EnvSensor\}, \{PhysSensor\})$
Eq_partition :
$partition(Component, \{EmerSurvCenter$
$\{EnvControl\}, \{MedSurCenter\})$
$Connector = Connector1, .., Connector15\top$
END

To specify behavioral features, we have two steps. First, we specify the first machine at a high level of abstraction. Second, we add all necessary details by using the refinement technique. We illustrate an example of machines called Machine1 that is refining the Machine0, adding communication between the Smart Home components. The behavior is described as follows: the Monitoring-Center sends a Message to Equipment and then remains released from resources. When the component Equipment becomes available, it receives the Message, process it and sends the Acknowledgement Message. When Monitoring-Center becomes available, it receives the ACK-Message, process it and then becomes deactivated. The invariants (Send_Message, Receive_Ack) specifies what is the sent message, who is the sender and the receiver (The same description for the message from the Monitoring-Center to the HomeCare-Actor Component).

MACHINE
$Machine1$
REFINES
$Machine0$
SEES
$Context1$
VARIABLES
$Send, Receive$
INVARIANTS
Send_Message : $Send \in BOOL$
Receive_ACK : $Receive \in BOOL$
EVENTS
INITIALISATION
EVT
$i1 : Send \in MonitoringCenter \rightarrow Equipment$
$i2 : Receive \in MonitoringCenter \rightarrow Equipment$
$i3 : transmit := MonitoringCenter \rightarrow True,$
$Equipment \rightarrow False$
$i4 : transmit := MonitoringCenter \rightarrow True,$
$HomeCareActor \rightarrow False$
END

During the refinement process, we check the correct transmission of messages between actors and we prove the correctness property using the Event-B specifications. We demonstrate that there is no conflict problem between messages sent and received in parallel sequence which is not possible and correct with UML notations. Dispatchers cooperate together to route information from the producer-consumers to the subscribed event-dispatcher (Monitoring-Center). This interaction is governed by a principle of information dissemination requiring that produced information have to reach all subscribed consumers. This is to check the correct message transmission between dispatchers and producer-consumers.

The Event-b specifications allow to guarantee a correct by construction architectures. This formal method provides three steps. At the first step, the designer describes the necessary information for the software architecture model and the relation between them. Then, the second step consists in generating automatically all the correct design architectures following a multi-scale modeling approach. In fact, for each model, a 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 architecture according to resource constraints.

V. Related Work

Considerable research studies have been proposed on the description of software architectures. Multi-level modeling approaches [10] have been proposed to represent the different abstraction levels. Baresi et al. [3] presented a UML based approach and proposed formal verification and validation of embedded systems. The approach is implemented using the "CorrettoUML": a formal verification tool for UML models. Other research studies have been proposed for the specification

 ISBN: 978-1-61208-498-5

of software systems using formal methods. Model verification activity [12] is performed to ensure the correctness of model. Formal verification means that any errors found in the design of the system should be corrected. Ben Younes et al. [4] proposed a meta-model transformation between UML Activity Diagram and Event B models. A formal framework is defined to ensure the correctness of the proposed transformations, and the event B method is used for the formal verification of applications. Bryans et al. [6] presented a model-based approach to assist in the integration of new or modified constituent systems into a System of Systems. The authors defined two levels for system composition, the high-level structural view that considers the connections within the system, and the low-level behavioral view that deals with the behavior of contractual specifications. They treated an industrial case study for modeling Audio/Video system.

We can note that the research activities [3], [4], [6] deal only with structural features during the design of the architecture. They do not take into account the respect of behavioral featuress to validate the architecture. Whereas, in our work, we deal with both structural and behavioral features.

We analyze that 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 software architectures. Model-based methods have addressed significant challenges in software Engineering. Semi-formal models are used in the architectural description of complex software systems. This representation has advantages, mainly with regard to comprehension, and can help to clarify areas of incompleteness and ambiguity in specifications.

In this study, we have considered that a given modeling level can be described by both vertical and horizontal scales. Our work will help the architect to design a correct and elaborated solutions for modeling multiple different levels of description of the same modeling level through the scales. Thus, we applied our model-based approach for describing multi-scale architecture , defining both the structure and the behaviour of the complex system and interactions between them. Event-B as a formal method support an interactive and an automatic theorem proving so that the resulted specification after the transformation process can be proved automatically. With the notion of refinement, we can to perform successive refinement to the Event-B model in order to specify different description scales.

VI. Conclusion

In this paper, we have presented a multi-scale modeling and specification approach for software architectures. We have proposed UML notations to represent the structure and the behavior for modeling different description scales, and second formally specified the models with the Event-B method. The formalisation phase allows to formally specify both structural and behavioural features of these architectures at a high level of abstraction using Event-B method. We implemented the elaborated specifications under the Rodin platform. We have also presented the application of our approach to the smart home scenario. Finally, we have presented some research studies discussing multi-level modeling for software architectures using semi-formal and formal methods. Currently, we are working on the improvement of the formal verification of architectural properties, and the model transformation from UML to Event-B. In our future work, we expect to apply the multi-scale approach to other use-cases for modeling complex systems architectures (e.g. System of Systems (SoS)) and implement a tool supporting the approach.

References

[1] Compatibility and inheritance in software architectures. *Science of Computer Programming*, 41(2):105 – 138, 2001.

[2] J.-R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, New York, NY, USA, 1st edition, 2010.

[3] L. Baresi, G. Blohm, D. S. Kolovos, N. Matragkas, A. Motta, R. F. Paige, A. Radjenovic, and M. Rossi. Formal verification and validation of embedded systems: The UML-based mades approach. *Softw. Syst. Model.*, 14(1):343–363, Feb. 2015.

[4] A. Ben Younes, Y. Hlaoui, and L. Jemni Ben Ayed. A meta-model transformation from uml activity diagrams to event-b models. In *Computer Software and Applications Conference Workshops (COMPSACW), 2014 IEEE 38th International*, pages 740–745, July 2014.

[5] 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.

[6] J. Bryans, J. Fitzgerald, R. Payne, A. Miyazawa, and K. Kristensen. Sysml contracts for systems of systems. In *System of Systems Engineering (SOSE), 2014 9th International Conference on*, pages 73–78, June 2014.

[7] S. Cîmpan, F. Leymonerie, and F. Oquendo. *Software Architecture: 2nd European Workshop, EWSA 2005, Pisa, Italy, June 13-14, 2005. Proceedings*, chapter Handling Dynamic Behaviour in Software Architectures, pages 77–93. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[8] T. S. Hoang, H. Kuruma, D. Basin, and J.-R. Abrial. *Integrated Formal Methods: 7th International Conference, IFM 2009, Düsseldorf, Germany, February 16-19, 2009. Proceedings*, chapter Developing Topology Discovery in Event-B, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[9] I. Omoronyia, G. Sindre, S. Biffl, and T. Stålhane. *Relating Software Requirements and Architectures*, chapter Understanding Architectural Elements from Requirements Traceability Networks, pages 61–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[10] P. Petrov, U. Buy, and R. Nord. The need for a multilevel context-aware software architecture analysis and design method with enterprise and system architecture concerns as first class entities. In *Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on*, pages 147–156, June 2011.

[11] W. Su, J. Abrial, and H. Zhu. Formalizing hybrid systems with event-b and the rodin platform. *Sci. Comput. Program.*, 94:164–202, 2014.

[12] B. Uchevler and K. Svarstad. Assertion based verification using psl-like properties in haskell. In *Design and Diagnostics of Electronic Circuits Systems (DDECS), 2013 IEEE 16th International Symposium on*, pages 254–257, April 2013.

 ISBN: 978-1-61208-498-5