Han DS, Yang QL, Xing JC *et al.* EasyModel: A refinement-based modeling and verification approach for self-adaptive software. JOURNAL OF COMPUTER SCIENCE AND TECHNOLOGY 35(5): 1016–1046 Sept. 2020. DOI 10.1007/s11390-020-0499-x

# EasyModel: A Refinement-Based Modeling and Verification Approach for Self-Adaptive Software

De-Shuai Han[1], Qi-Liang Yang[2,*], Jian-Chun Xing[2], and Guang-Lian Ma[1,3]

[1] *College of Combat Support, Rocket Force University of Engineering, Xi'an 710025, China*
[2] *College of Defense Engineering, Army Engineering University of PLA, Nanjing 210007, China*
[3] *Teaching and Research Support Center, Rocket Force University of Engineering, Xi'an 710025, China*

E-mail: handeshuai@126.com; {yql, xjc}@893.com.cn; maguanglian@126.com

Received April 6, 2020; revised July 25, 2020.

**Abstract** Self-adaptive software (SAS) is gaining popularity as it can reconfigure itself in response to the dynamic changes in the operational context or itself. However, early modeling and formal analysis of SAS systems becomes increasingly difficult, as the system scale and complexity is rapidly increasing. To tackle the modeling difficulty of SAS systems, we present a refinement-based modeling and verification approach called EasyModel. EasyModel integrates the intuitive Unified Modeling Language (UML) model with the stepwise refinement Event-B model. Concretely, EasyModel: 1) creates a UML profile called AdaptML that provides an explicit description of SAS characteristics, 2) proposes a refinement modeling mechanism for SAS systems that can deal with system modeling complexity, 3) offers a model transformation approach and bridges the gap between the design model and the formal model of SAS systems, and 4) provides an efficient way to verify and guarantee the correct behaviour of SAS systems. To validate EasyModel, we present an example application and a subject-based experiment. The results demonstrate that EasyModel can effectively reduce the modeling and formal verification difficulty of SAS systems, and can incorporate the intuitive merit of UML and the correct-by-construction merit of Event-B.

**Keywords** self-adaptive software, formal modeling, Event-B, refinement, correct-by-construction

## 1 Introduction

Modern software systems, such as large-scale web service systems and cyber-physical systems, are facing problems of increasing size, incremental complexity and unpredictable environment changes. While addressing the above challenges, it becomes necessary to develop so-called self-adaptive software (SAS) systems. In fact, software self-adaptation has become a hot research issue [1–3] in the software engineering community. Software self-adaptation endows a software system with the capability to to reconfigure itself in response to the dynamic changes in the running context.

The early design, modeling and formal analysis of SAS systems is essential to improve development efficiency and to ensure system reliability. As a standardized object oriented modeling language, Unified Modeling Language (UML) has been widely used to depict SAS systems. The self-adaptive software community has proposed ACML (Adapt Case Modeling Language) [4] to specify SAS requirements, the FAME approach [5] to depict self-adaptation attributes, and the design patterns [6] to model real-time characteristics. The above UML-based approaches provide an explicit description of structure characteristics of SAS systems, but they lack formal semantics to support the behaviour analysis of SAS systems. To tackle the formal verification problem of SAS systems, the research community has proposed automata-based approaches [7, 8], Petri Net based approaches [9, 10], Bayesian network based approaches [11] and so on. These approaches pro-

Regular Paper
Special Section on Software Systems 2020
The work was supported by the National Key Research and Development Program of China under Grant No. 2017YFC0704100.
*Corresponding Author
©Institute of Computing Technology, Chinese Academy of Sciences 2020

vide effective modeling solutions to small-scale SAS systems. However, they are limited in dealing with system complexity. Practices [12, 13] show that the above approaches suffer from problems like state-space explosion, when the system scale and complexity increase. Another limitation of existing approaches is the gap between the design model (e.g., UML) and formal model (e.g., Petri Net) of SAS systems, because of a lack of automatic model transformation mechanism. And this gap increases the modeling difficulty of SAS systems.

As SAS is gaining growing attention in complex information systems, such as large-scale web service systems, we need a novel approach to design SAS systems, which can: 1) provide an explicit description of self-adaptation characteristics, 2) decompose system complexity and alleviate modeling difficulty, and 3) offer an efficient formal analysis way and provide guarantees for the correct behaviour.

To address the formulated problems, we present a refinement-based modeling and verification approach called EasyModel (Ease Modeling Difficulty). EasyModel integrates the intuitive UML model with the stepwise refinement Event-B model [14], as UML and Event-B can complement with each other in depicting complex systems. UML is intuitive and easy to use, and is suitable for describing self-adaptation requirements and SAS structures; while Event-B performs well in specifying and verifying complex software behaviours, with its stepwise refinement and correct-by-construction advantages. Besides, both UML and Event-B provide notions to model both static and dynamic software characteristics. The above complementarities and similarity all contribute to integrating the UML model with the Event-B model.

To the best of our knowledge, the EasyModel approach is the first attempt to integrate the intuitive UML model with the stepwise refinement Event-B model in modeling and verifying SAS systems. Concretely, EasyModel makes the following contributions.

1) It preserves the intuitive merit of our initial work [5, 15], and presents an explicit description of SAS characteristics by creating a UML profile called AdaptML.

2) It proposes a stepwise refinement modeling mechanism for SAS systems that deals with system modeling complexity by gradually enriching system details from the abstract model to the concrete model.

3) It offers a model transformation approach from the extended UML model to the Event-B model, which bridges the gap between the design model and the formal model of SAS systems, and alleviates modeling difficulty of SAS systems.

4) It presents an efficient formal verification method by integrating Proof Obligation [14] discharging with model checking, which provides guarantees for the correct behaviour of SAS systems.

We have evaluated the EasyModel approach with an example application and a subject-based experiment, and the results reveal that EasyModel can reduce modeling and formal verification difficulty of SAS systems, and can incorporate the intuitive merit of the UML model and the correct-by-construction merit of the Event-B model.

The rest structure of this paper is as follows. Section 2 introduces corresponding concepts and an adaptation scenario. Section 3 provides requirements for modeling SAS systems and the conceptual framework of the proposed approach. In Section 4, details of the approach are illustrated. Section 5 presents an example application of the EasyModel approach. We evaluate the approach in Section 6 with a subject-based experiment. Section 7 presents related work. Finally, Section 8 concludes the paper with a discussion on future work.

## 2 Background

In this section, we first present a brief introduction to UML extending mechanism and the Event-B model, and then provide an overview on the adaptation scenario of ZNN.com①.

### 2.1 UML and Its Extending Mechanism

UML is a standard object-oriented modeling language offered by the Object Management Group [16]. It provides various diagram views for designing and modeling various aspects of software systems, e.g., the class diagram for structure modeling and the activity diagram for behaviour description. In addition, UML provides composite structures (e.g., composite class) to support large-scare system modeling. With the advantages of being intuitive and easy to use, UML has been widely used for software modeling and designing.

However, UML lacks direct semantic and syntactic support for SAS systems. On the one hand, UML provides no concept of SAS systems (e.g., Monitor), and developers cannot capture semantic information

①http://znn.com/, July 2020.

from the elements' names. In order to support domain-specific modeling, UML is designed to be open and extensible, and it provides the extending mechanism of UML profile, including stereotypes, tagged values, and constraints. UML profile [16] provides a straightforward mechanism to adapt an existing meta-model with constructs that are specific to a particular domain. As a consequence, standard UML modeling tools, such as Papyrus [17], are able to work with profile-based models. On the other hand, UML is a semi-formal modeling language, and cannot be analyzed and verified. To make up for this problem, UML models can be formalized by formal models, such as the timed automata model and the Event-B model.

### 2.2 Event-B Model

Event-B is a system-level modeling and analysis model proposed by Abrial [14]. Main components and features of Event-B are as follows.

*Context and Machine.* Event-B employs set theory as modeling notations and is composed of two basic constructs, context and machine. A context depicts the static structure of a system using carrier sets, constants, axioms and theorems, and a machine describes the dynamic structure of a system using variables, invariants, theorems, variants and events, as shown in Fig.1. A machine may see one or more contexts, and this will allow it to use all the elements defined in the contexts. Events are generally expressed as:

any *variable* where *guard* then *action*.

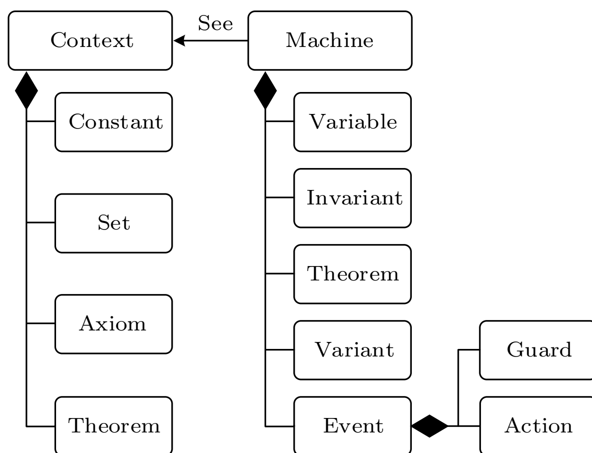



Fig.1. Meta-model of Event-B.

*Refinement.* This mechanism consists in adding details gradually while preserving the original properties of the system. Refinement allows Event-B to model any large and complex systems. The consistency between different refinement levels is guaranteed with a particular invariant called gluing invariant.

*Correct-by-Construction.* In Event-B, each model will be analyzed and proved, guaranteeing that it is correct relative to a set of proof obligations (POs). Therefore, when the last model is finished, we are able to say that this model is correct-by-construction.

Event-B is supported by several tools, currently in the form of a platform called Rodin [18]. Rodin supports lots of plug-ins, such as ProB for animation/model checking [19] and Atelier B provers for POs discharging [20]. Currently, Event-B has been actively used within several European Union projects [21, 22] and cyber-physical systems [23]. However, SAS owns its domain-specific characteristics, and new refinement patterns are needed to model and depict the self-adaptation logic.

### 2.3 Adaptation Scenario: ZNN.com

The self-adaptation scenario of ZNN.com system is selected from Cheng's doctoral dissertation [24]. ZNN.com is a complex web-based client-server system, and it serves new contents to customers through a cluster of application servers. Its architecture is shown in Fig.2.

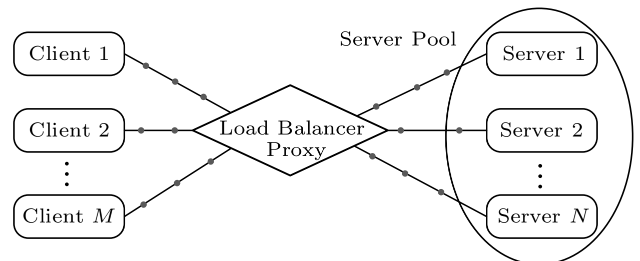



Fig.2. Architecture of ZNN.com [24].

The system of ZNN.com utilizes a balancer to balance requests across the pool of replicated servers. Traditionally, the size of the server pool can be manually adjusted according to the amount of client processes. And the servers can serve contents in either the text mode or the multimedia mode to the clients. However, the amount of client requests changes dynamically during system operation, and ZNN.com demands the ability of self-adaptation to deal with dynamic changes.

However, new problems arose when researchers used the SAS paradigm to reconstruct the ZNN.com system. Firstly, it is important to explicitly specify the self-adaptation logic early at the design phase. But

traditional modeling languages, such as UML, lack direct semantic and syntactic support for self-adaptive systems, and the self-adaptation entities (e.g., LoadMonitor) cannot be explicitly depicted. Secondly, an early formal verification is necessary to provide guarantees for software behaviours. However, it is difficult for software engineers to establish the formal model of ZNN.com system just from the above nature language based requirements, and there is a gap between the design model and the formal model of SAS systems. Therefore, we choose ZNN.com as an adaptation example to verify the effectiveness of EasyModel in modeling and formally verifying SAS systems.

## 3 Overview of the EasyModel Approach

In this section, we firstly analyze the key requirements for modeling self-adaptive systems, and then present the conceptual framework of the EasyModel approach.

### 3.1 Requirements for Modeling SAS Systems

Owning to the high system complexity and domain-specific characteristics, the SAS systems are difficult to design, model and analyze. A systematic modeling approach for SAS systems is needed, and key requirements for modeling SAS are analyzed and summarized as follows.

Firstly, the SAS systems are composed of the self-adaptation logic and the application logic, and the self-adaptation logic is composed of interactive self-adaptation entities (e.g., sensor). As a sequence, self-adaptation entities, interactive relationships (e.g., trigger), and self-adaptation attributes (e.g., monitoredVariable) play crucial roles in SAS systems, and need to be explicitly depicted (R1). Secondly, as the self-adaptation loops intertwine the self-adaptation logic with the application logic, they should be treated as first-class elements in SAS systems. Thus, the self-adaptation loops should also be explicitly depicted early in the design phase (R2). Thirdly, according to the autonomic computing architecture[25], the SAS systems are realized based on the Monitor-Analyze-Plan- Execute activities. Therefore, the self-adaptation activities need to be explicitly depicted (R3). Fourthly, as SAS systems are large, complex and heterogeneous, the modeling approach should provide mechanism to tackle system complexity (R4), and alleviate modeling difficulty. Finally, as system scale increases, the self-adaptation logic is error-prone, and the modeling approach should provide mechanism to guarantee the correctness of the self-adaptation logic (R5).

In a word, the modeling approach for SAS systems should take into account self-adaptation characteristics as well as system complexity.

### 3.2 Conceptual Framework of EasyModel

According to the modeling requirements of R1, R2 and R3 in Subsection 3.1, there is an urgent need to provide an explicit description of SAS characteristics. Therefore, UML is chosen and extended to model SAS systems, as it provides multiple visual diagram views, and is easy to use for most software engineers. Meanwhile, in order to make up for the semi-formal deficiency of UML and to fulfill the requirements of R4 and R5, the extended UML needs to be integrated with the stepwise refinement Event-B model. The conceptual framework of the EasyModel approach is shown in Fig.3. Concretely, it is composed of the following steps.

1) *Visual Modeling of SAS by Extending UML*. In this step, a UML profile called AdaptML is created by extending UML. According to the modeling requirement of R1, a structure view called adapt class diagram is created, and it can present an explicit description of self-adaptation entities, relationships and self-adaptation attributes. Similarly, to meet the modeling requirement of R2 and R3, a behaviour view called adapt activity diagram is established, and it can provide an explicit description of self-adaptation activities and self-adaptation loops. Besides, a supporting tool for AdaptML is developed to improve the modeling efficiency.

2) *Model Transformation from AdaptML to Event-B*. By analyzing corresponding relationships between AdaptML and Event-B model, a set of mapping rules and a supporting tool are created, supporting automatic transformation from the adapt class diagram to context of Event-B and from the adapt activity diagram to the machine of Event-B, respectively. This model transformation can bridge the gap between the design model and the formal model of SAS systems, and alleviate modeling difficulty.

3) *Formal Modeling of SAS Systems by Defining New Event-B Refinement Patterns*. In order to depict complex self-adaptation activities, e.g., sequential node, branch node, and self-adaptation loop, a set of new Event-B refinement patterns have been proposed and proved. The stepwise refinement patterns can build SAS systems by gradually enriching implementation details, decomposing system complexity of SAS systems.

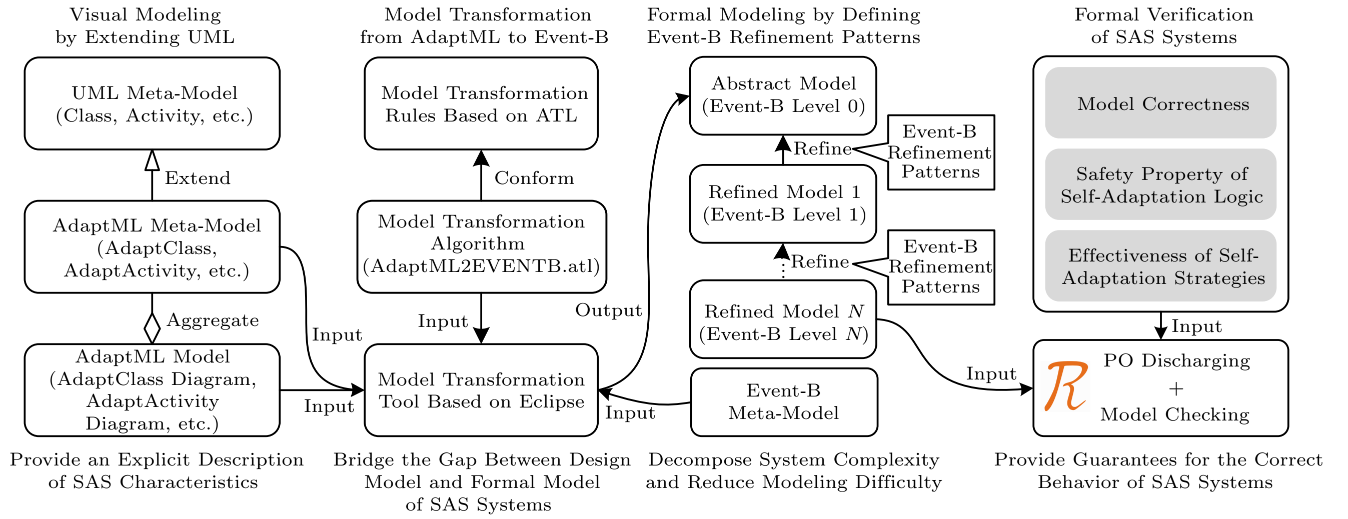



Fig.3. Conceptual framework of the EasyModel approach.

4) *Formal Analysis and Verification of SAS Systems Based on Event-B Model*. Model correctness and self-adaptation properties (e.g., effectiveness of self-adaptation strategies) can be verified by integrating the Proof Obligation discharging technique (supported by the Atelier B Provers [20] plug-in of Rodin) with the model checking technique (supported by the ProB [19] plug-in of Rodin). And this integrated verification method can improve verification efficiency.

## 4 Implementation of the EasyModel Approach

In this section, the implementation details of the EasyModel approach are presented, by defining meta-models, creating refinement patterns, establishing model transformation rules, and constructing supporting tools.

### 4.1 Visual Modeling of SAS by Extending UML

In order to support an explicit description of self-adaptation logic, we create the structure view of adapt class diagram and the behaviour view of adapt activity diagram by extending UML class diagram and activity diagram, respectively.

#### 4.1.1 *Structure View: Adapt Class Diagram*

The adapt class diagram is created to explicitly specify structure characteristics of SAS systems, e.g., the self-adaptation entities, interactive relationships, and self-attributes. And it is created by extending class diagram and by incorporating the self-adaptation concepts. Fig.4 depicts the meta-model of the adapt class diagram. The formal definition of the adapt class diagram is as follows.

**Definition 1** (Adapt Class Diagram). *An adapt class diagram can be defined as a tuple*:

$$ACD = (C_{\rm A}, R_{\rm A}, A_{\rm A}, M_{\rm A}, S).$$

• $C_{\rm A}$ *is a set of adapt classes. For each adapt class* $c \in C_{\rm A}$, *c is created based on the class of UML, and is symbolically represented by* <<*stereotype*>>. $C_{\rm A}$ $= \{$*Monitor, Analyzer, Planner, Executor, Knowledge, Application-Logic*$\}$. *They are all composite classes (symbolized by* $\infty$*), and can be decomposed into a group of sub-classes. For example, Monitor can be decomposed into sensors while Executor can be decomposed into effectors.*

• $R_{\rm A}$ *is a set of adapt-relationships.* $R_{\rm A}$ *is derived by extending the relationship of UML.* $R_{\rm A} \subseteq C_{\rm A} \times C_{\rm A}$ *is a binary relation, and relates all the adapt classes.* $R_{\rm A} = \{$*probe, trigger, invoke, adjust*$\}$, *and Table 1 lists detailed description of all the adapt-relationships.*

• $A_{\rm A}$ *is a set of adapt-attributes, belonging to the adapt classes.* $A_{\rm A}$ *is created by extending the attribute of UML.* $A_{\rm A} = A \cup A_{\rm T}$, *in which A and* $A_{\rm T}$ *represent two different kinds of attributes. A is a set of general attributes, while* $A_{\rm T}$ *is a set of tagged values. Tagged values are represented as* $[Tag] = [Value]$, *which depicts self-adaptation attributes of each adapt class. For example, Kind = Environment implies that the monitored object is the software running environment.*

• $M_{\rm A}$ *is a set of adapt-methods, which is mainly used to depict adaptation algorithms.* $M_{\rm A}$ *is created by extending the method of UML model.*

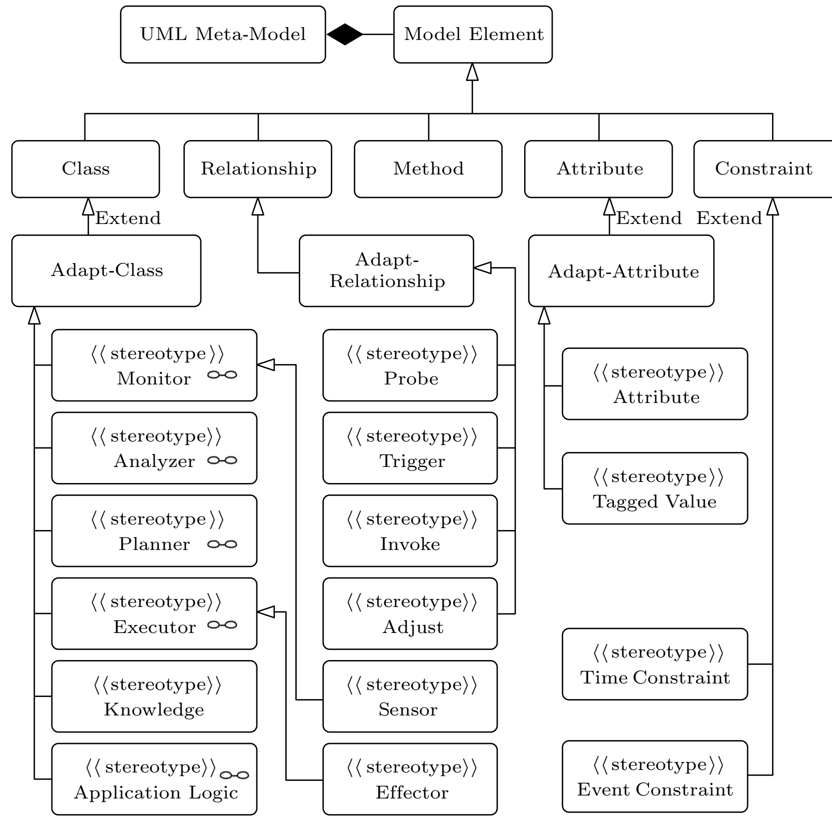



Fig.4. Meta-model of adapt class diagram.

**Table 1**. Detailed Description of the Adapt-Relationships

| Type | Notation | Description |
|---|---|---|
| Probe | $A \to_{\mathrm{M}} B$ | $B$ periodically transmits real-time data to $A$; $A \in$ Monitor, $B \in$ Application Logic. |
| Trigger | $A[c]; B$ | If condition $c$ in $A$ is true, operation in $B$ triggers; $A, B \in$ {Monitor, Analyzer, Planner, Executor}. |
| Invoke | $A \to_{\mathrm{I}} B$ | $B$ stores all data to be invoked by $A$; $A \in$ {Monitor, Analyzer, Planner, Executor}; $B \in$ Knowledge. |
| Adjust | $A \to_{\mathrm{A}} B$ | $A$ is an Executor unit that employs all decisions to adjust the attributes, behaviours, or structures of $B$. |

• *$S$ is a set of constraints, which are used to refine model elements or attributes by imposing conditions or restrictions on specific elements or variables.*

As can be seen from the above definitions, developers can capture semantic information from the elements' names (e.g., Monitor) in the adapt class diagram, and the self-adaptation structures are depicted explicitly (R1). Besides, the composite structure of adapt classes (e.g., Monitor) provides a step-by-step way to model SAS systems, which reduces modeling difficulty of large-scale systems (R4). The definition of adapt class diagram has referred to ideas from the fuzzy class diagram in our previous work [5]. The former is created for general SAS systems, while the latter is customized just for fuzzy self-adaptive software systems. The fuzzy class diagram is composed of a set of fuzzy control based modeling elements, e.g., <<FuAdapter>>, <<Fuzzification>>, <<Defuzzification>>, but it lacks modeling elements to depict general SAS systems. Besides, the fuzzy class cannot be decomposed again, and lacks mechanism to specify SAS systems at different abstract levels.

### 4.1.2 Behaviour View: Adapt Activity Diagram

The adapt activity diagram is defined to explicitly depict behaviours of SAS systems, e.g., the self-adaptation activities. It is created by extending UML activity diagram and by incorporating characteristics of SAS systems, and its definition is as follows.

**Definition 2** (Adapt Activity Diagram). *An adapt activity diagram is defined as a tuple*:

$$AAD = (A_{\mathrm{A}}, T_{\mathrm{A}}, Ctrl, P_{\mathrm{A}}, O_{\mathrm{A}}).$$

• $A_{\rm A} = A_{\rm action} \cup A_{\rm activity} \cup a_{\rm I} \cup a_{\rm F}$. *Concretely,* $A_{\rm action}$ *is a set of adapt-actions, which is created by extending the UML action.* $A_{\rm action}$ *is the basic activity unit and cannot be decomposed.* $A_{\rm activity}$ *is a set of adapt-activities, which is created by extending UML activity.* $A_{\rm activity} = \{Monitor, Analyze, Plan, Execute\}$, *which can be decomposed into Adapt-Actions or sub-activities*; $a_{\rm I}$ *is the initial activity while* $a_{\rm F}$ *is the final activity.*

• $T_{\rm A}$ *is a finite set of transitions from one activity to another, and* $T_{\rm A} = \{t_1, t_2, \ldots, t_n\}$; *each transition is attached with one condition* $G(t)$.

• *Ctrl is a finite set of control nodes,* $Ctrl \subseteq (A_{\rm A} \times T_{\rm A}) \cup (T_{\rm A} \times A_{\rm A})$. $Ctrl = \{Sequential, Branch, Cooperation, Self\text{-}adaptation\ loop\}$. *Sequential represents the sequentially executed activities, such as Monitor and Analyze in Fig.5(a); Branch inherits from the Decision node and represents the conditional activities, such as act 1 and act 2 in Fig.5(b); Cooperation inherits from the Fork/Join node and represents the coordinating activities, such as sense 1 and sense 2 in Fig.5(c); Self-adaptation loop represents the circular relations, such as activities in Fig.5(d).*

• $P_{\rm A}$ *is a set containing all the partitions, which identify a collection of actions belonging to the same objects.*

• $O_{\rm A}$ *is a set of adapt-objects, which are instances of adapt classes* $C_{\rm A}$.

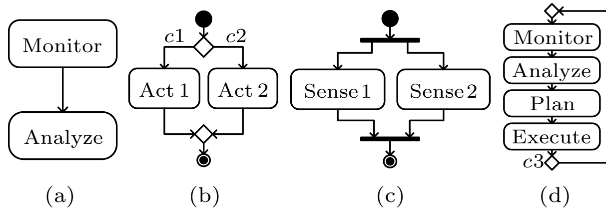



Fig.5. Example of sequential node refinement. (a) Sequential. (b) Branch. (c) Cooperation. (d) Self-adaptation loop.

According to the above definition, it can be seen that the adapt activities provide an explicit description of self-adaptation behaviours and the self-adaptation loops (R2, R3). Besides, the adapt activities can be decomposed into sub-activities or adapt-actions, and provides a step-by-step mechanism to gradually depict SAS behaviours, reducing modeling difficult (R4).

#### 4.1.3 *Supporting Tool: AdaptML Profile*

For the convenience of application, we have created a plug-in called AdaptML profile for the above two diagrams. AdaptML profile is created as a plug-in of Papyrus [17]. Fig.6 illustrates the graphical user interface (GUI) of an adapt class diagram. The modeling infrastructures, such as adapt classes and adapt attributes, are predefined in this tool. And software engineers who are familiar with UML can quickly create an adapt class diagram by applying the predefined stereotypes. This supporting tool of AdaptML profile can reduce modeling difficulty of SAS systems, and improve modeling efficiency. Besides, it owns the following distinctive characteristics.

1) It contributes to efficient use and management, as modeling elements have all been predefined in the toolbar of Papyrus.

2) It owns good compatibility with the standard UML model, facilitating understanding and using for general software engineers.

### 4.2 Model Transformation Rules from AdaptML to Event-B

The AdaptML model provides an intuitive description of SAS systems, but it cannot be formally analyzed and verified. In addition, it is difficult to establish the formal model of SAS systems for general software engineers. In order to bridge the gap between the design model and the formal model of SAS systems, we study the correspondences between the meta-model of AdaptML and the meta-model of Event-B, and propose a set of mapping rules from AdaptML to Event-B, as shown in Fig.7.

According to the mapping rules in Fig.7, the adapt class diagram can be transformed into the context of Event-B; while the adapt activity diagram can be transformed into the machine of Event-B. The composite structures (e.g., composite classes and composite activities) can be transformed to extending mechanism of context; while the partition of adapt activity diagram can be transformed to the refinement mechanism of Event-B. Detailed implementation follows below.

#### 4.2.1 *Transforming Adapt Class Diagram into Event-B Model*

The adapt class diagram is created to describe static characteristics of SAS systems. Thus, it can be formalized using the context of Event-B, and the adapt-attributes and adapt-methods can be formalized using variables and events of Event-B, respectively. Besides, the composite structures of adapt class can be mapped into the extending mechanism of the context, as shown in Fig.7.

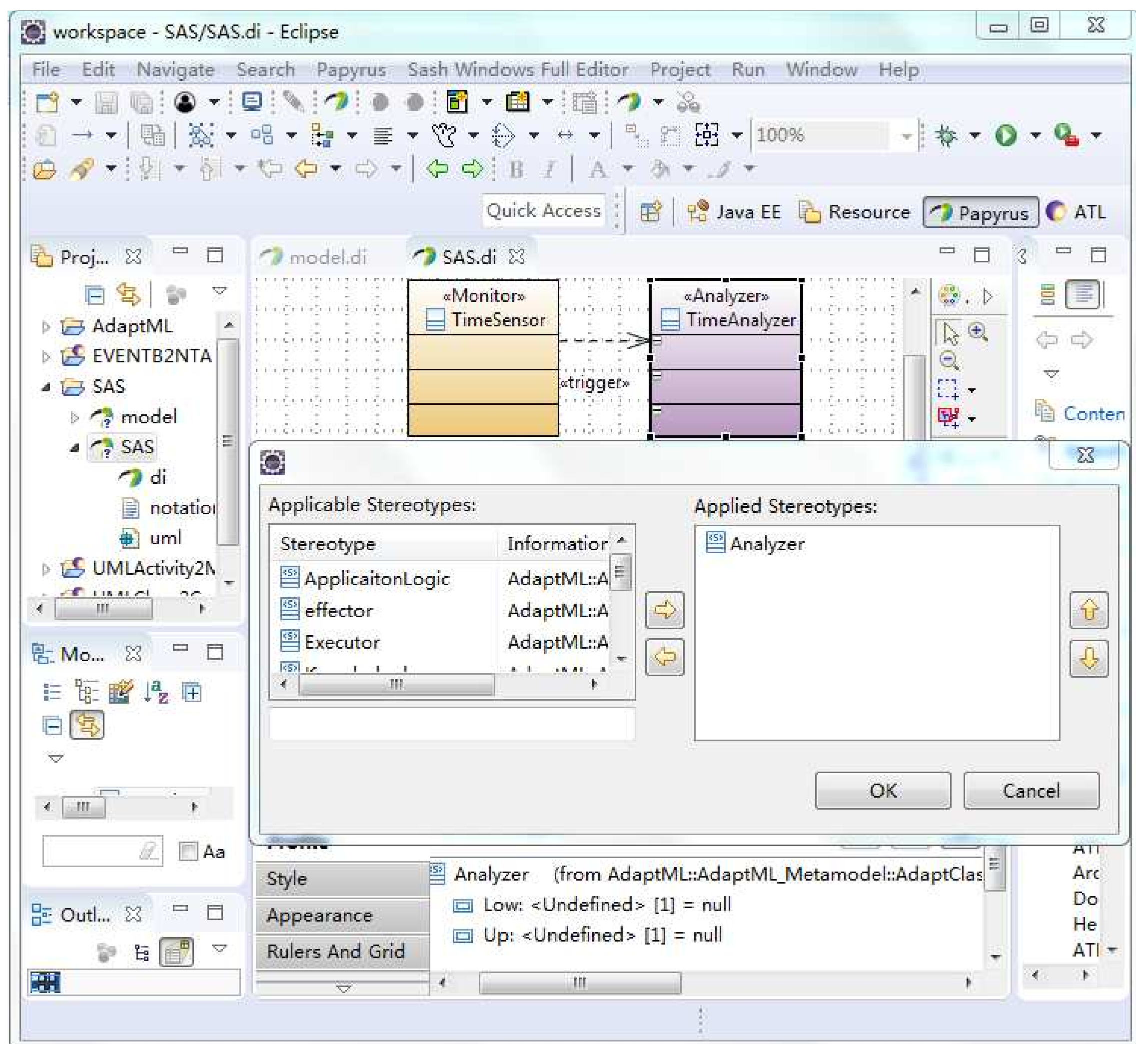



Fig.6. GUI of AdaptML profile (adapt class diagram).

*Transforming the General Adapt Class into Event-B*. Taking part of the adapt class diagram for an example, we illustrate the transformation process from the general adapt class to Event-B.

As shown in Fig.8, the sub-class <<TimeSensor>> and <<Monitor>> of adapt class can be mapped to constant and set of Event-B, respectively. Meanwhile, the generalization relationship between sub-class and super-class can be mapped as the "belong to" (i.e., $\in$) relationship between constant and set, represented as partition (Set, {constant}). Besides, the adapt-relationship, such as trigger, can be defined in constant, and formalized using the relation (represented as $\leftrightarrow$) of axioms. The source and the target of the relationship can be formalized as dom(source) and ran(target), respectively.

Besides, the adapt-attribute, such as *RespTime*, can be mapped as variable of a machine, and depicted using invariant of a machine. The adapt-method, such as *ProbTime*, can be mapped as event of a machine, as shown in Fig.8.

*Transforming Composite Adapt Class into Event-B*. In order to support complex system modeling, the adapt class diagram has defined the composite adapt class, which can be further decomposed into concrete adapt classes. Similarly, the Event-B model provides the extend mechanism to refine context. Thus, we have mapped the composite adapt class into the Extending mechanism of Event-B, as shown in Fig.9. The abstract adapt class of Executor can be decomposed into two sub-classes of effector 1 and effector 2. Correspondingly, the abstract context which defines Executor can be extended to the concrete context which defines effector 1 and effector 2.

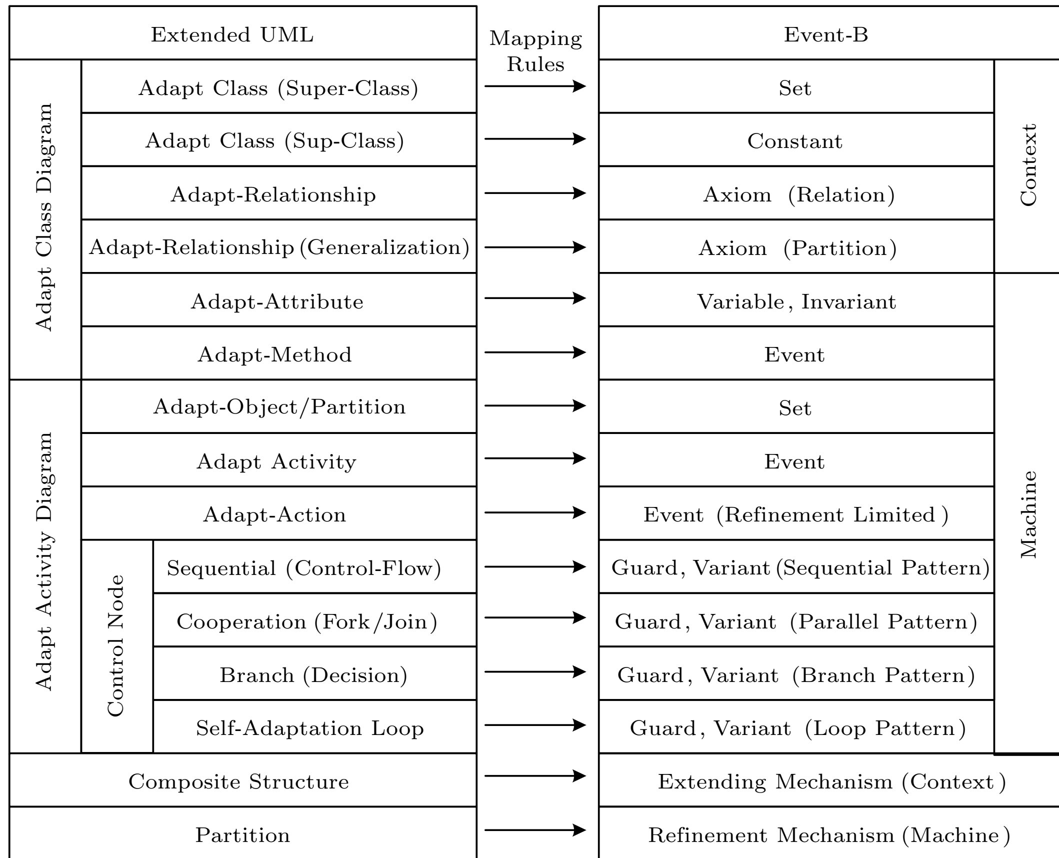



Fig.7. Mapping rules from extended UML to Event-B.

#### 4.2.2 *Transforming Adapt Activity Diagram into Event-B Model*

The adapt activity diagram and machine of Event-B are both in nature state-transition models. Considering this similarity, we have proposed mapping rules from adapt activity diagram to machine of Event-B model.

*Transforming a Basic Adapt Activity into Event-B.* As shown in Fig.7, each activity or action of the adapt activity diagram can be formalized by one event (represented as WHEN *guard* THEN *action* END) of Event-B. The event transformed from activity can be further refined. However, the event transformed from action cannot be refined anymore, as it is the basic atomic unit. Fig.10 presents an example of transforming a basic adapt activity into one event of Event-B model.

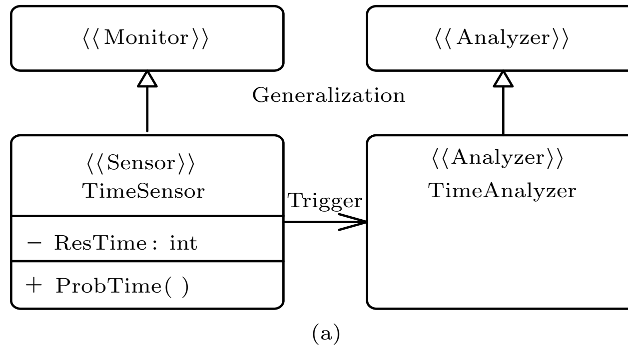
*Transforming Partition into Event-B.* In the adapt activity diagram, the partition divides an abstract adapt activity diagram into several sub-activities, as shown in Fig.11(a). Similarly, one abstract event of Event-B can be refined into several concrete events, as shown in Fig.11(b). Therefore, we transform partition of adapt activity diagram into the refinement mechanism of Event-B. As shown in Fig.11, the abstract adapt-activities of Monitor and Analyze are mapped to abstract events of Monitor and Analyze. Similarly, the concrete activities of act 1, act 2, act 3, and act 4 are mapped to corresponding concrete events of the Event-B model.

The composite structures and partition of AdaptML model, and the extending and refinement mechanism of Event-B model both contribute to dealing with the system complexity of self-adaptive logic and reducing modeling difficulty of SAS systems (R4).

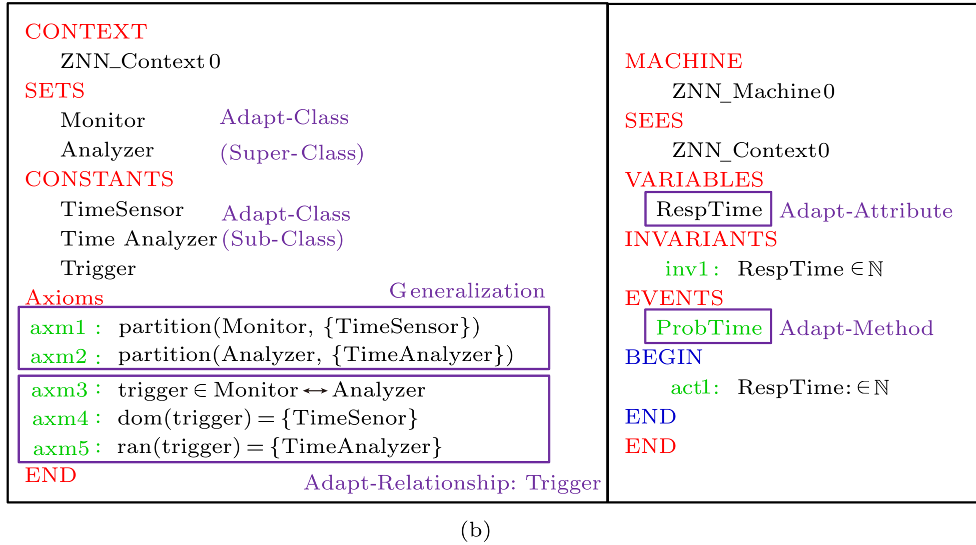
### 4.3 Model Transformation Tool: AdaptML2EventB

In order to support the automatic transformation from the AdaptML model to the Event-B model, we create a model transformation tool called AdaptML2EventB[2].

[2]https://pan.baidu.com/s/1OjjqTDVGz9mSyCEdNIyUbg, July 2020.

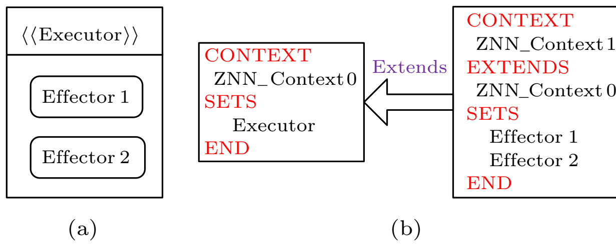



(a)



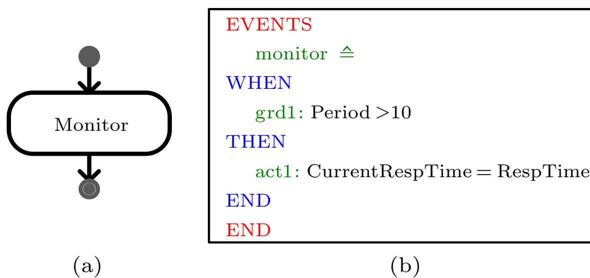

(b)

Fig. 8. Example: transforming the general adapt class diagram into Event-B model. (a) Fragments of adapt class diagram. (b) Corresponding fragments of Event-B model.




(a) (b)

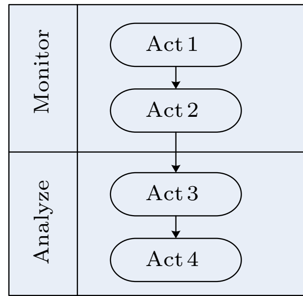
Fig. 9. Example: transforming composite adapt class into extending mechanism of Event-B model. (a) Composite class. (b) Extending mechanism of context.



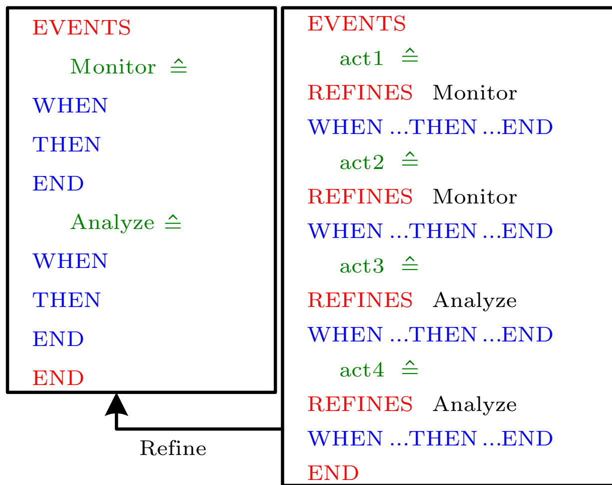

(a) (b)

Fig. 10. Example: transforming a basic adapt activity unit into an event of Event-B model. (a) Basic activity unit. (b) Event.

AdaptML2EventB is created based on the ATL (ATLAS Transformation Language) architecture [26], as shown in Fig.12. The ATL architecture is composed of three layers: the meta-meta-model layer, the meta-model layer and the model layer. In the created AdaptML2EventB, the meta-meta-model layer is defined according to the ecore meta-model. The meta-model layer consists of the source meta-model (i.e., the UML meta-model) and the target meta-model (i.e., the Event-B meta-model), which both conform to the ecore meta-model. Specifically, the UML meta-model consists of the adapt-class meta-model (i.e., the meta-model defined in Fig.4) and the adapt-activity meta-model (defined in Definition 2). The Event-B meta-model is composed of the context meta-model and the machine meta-model (defined in Fig.1). The ATL model transformation rules from AdaptML to Event-B are created based on the above meta-models and the mapping rules in Fig.7. Notably, the AdaptML model to be transformed from Papyrus needs preprocessing, and the generated Event-B model needs further




(a)




(b)

Fig.11. Example: transforming partition into refinement mechanism. (a) Partition. (b) Refinement mechanism.

post-processing before loading into Rodin. The architecture and the implementation principle of the AdaptML2EventB tool are shown in Fig.12.

The core component of the transformation tool is the ATL transformation rules, which are created based on the mapping rules defined in Fig.7. Fig.13 provides the fragment of the ATL model transformation rules from the adapt class to the context model. The transformation rules from the adapt activity to the machine model are similar.

The debug configuration interface of this tool is shown in Fig.14. First of all, users need to configure the model transformation program by selecting the ATL module (e.g., the ATL model transformation rules in Fig.13), the meta-models (e.g., AdaptClass.ecore and Context.ecore), the input path of the source models (e.g., sampleUMLClass.xmi), and the output path of the target modes (e.g., sampleContext.xmi). Then, model transformation can be implemented with just one click on the Debug button. Model correctness of the transformed Event-B model is guaranteed by checking proof obligations, which will be illustrated in Subsection 4.5.

Taking a simple adapt class diagram for example, we illustrate the application of the AdaptML2EventB tool, as shown in Fig.15.

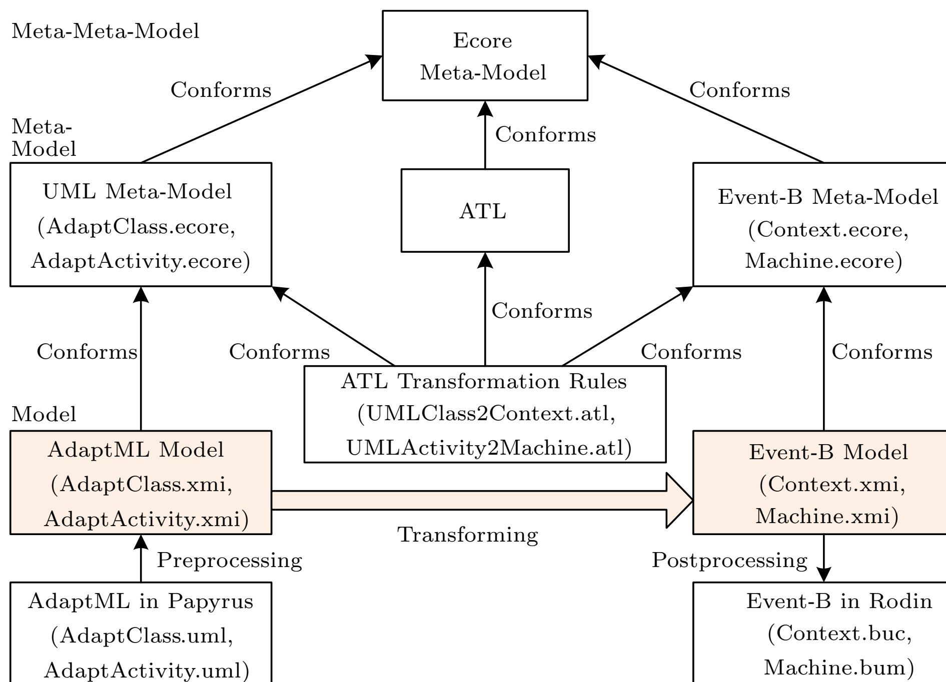



Fig.12. Architecture of the AdaptML2EventB tool.

```
UMLClass2Context.atl
1 -- @path Context=/UMLClass2Context/Context.ecore
2 -- @path UMLClass=/UMLClass2Context/UMLClass.ecore
3
4 module UMLClass2Context;
5 create OUT : Context from IN : UMLClass;
6 helper context UMLClass!AdaptClass def: exit(): Boolean=
7     if not self.name.oclIsUndefined()
8     then true
9     else true
10    endif;
11 helper context UMLClass!SubClass def: exit(): Boolean=
12    if not self.name.oclIsUndefined()
13    then true
14    else true
15    endif;
16 helper context UMLClass!Generalization def: exit(): Boolean=
17    if not self.name.oclIsUndefined()
18    then true
19    else true
20    endif;
21 helper context UMLClass!AdaptRelationship def: exit(): Boolean=
22    if not self.name.oclIsUndefined()
23    then true
24    else true
25    endif;
26 rule AdaptClass2carrierSet {
27    from s: UMLClass!AdaptClass(s.exit())
28    to
29    t:  Context!carrierSet (
30         name <- ' ',
31         identifier <- s.name)
32    }
33 rule SubClass2Constant {
40
41 rule Generalization2Axiom {
42    from s: UMLClass!Generalization(s.exit())
43    to
44
45    t:  Context!axiom (
46         name <- ' ',
47         label <-'axiom_'+ s.name,
48         predicate <- 'partition(' + s.target + ', {' + s.source + '})')
49 }
50 rule AdaptRelationship2Axiom {
51    from s: UMLClass!AdaptRelationship(s.exit())
52    to
53
```

Fig.13. Fragment of the ATL transformation rules.

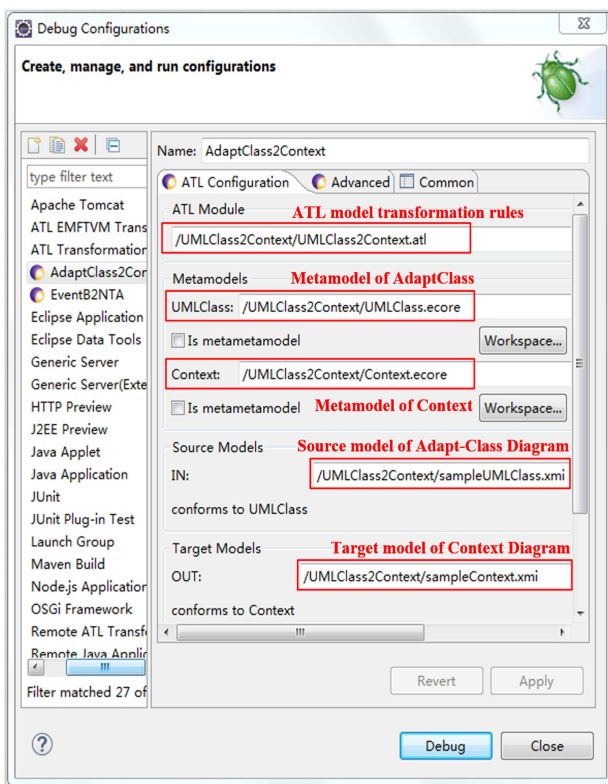
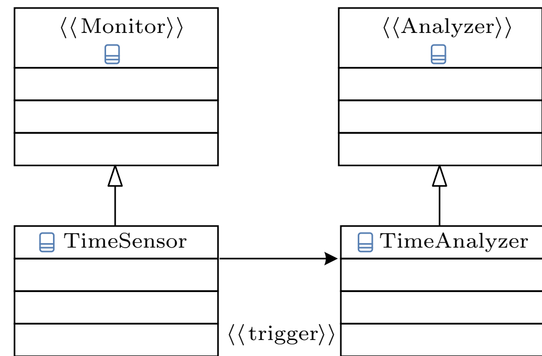



Fig.14. GUI of the model transformation tool.




(a)

```
sampleContext
CONTEXT
    sampleContext
SETS
    Monitor
    Analyzer
CONSTANTS
    TimeSensor
    TimeAnalyzer
    trigger
AXIOMS
    generalization1:   partition(Monitor, {TimeSensor}) not theorem
    generalization2:   partition(Analyzer, {TimeAnalyzer}) theorem
    trigger1:   trigger ∈ Monitor↔Analyzer not theorem
    trigger2:   dom(trigger)={TimeSensor} not theorem
    trigger3:   ran(trigger)={TimeAnalyzer} not theorem
```

(b)

Fig.15. Transforming adapt class diagram into context. (a) Part of an adapt class diagram. (b) Generated context model.

The diagram to be transformed is shown in Fig.15(a), and it is composed of two abstract classes (i.e., Monitor and Analyzer), two sub-classes (i.e., TimeSensor and TimeAnalyzer), and adapt-relationships (i.e., Trigger and Generalization). Applying the model transformation tool, this simple adapt class diagram is firstly preprocessed and saved as .xmi file. Then, a context file with the .xmi format can be generated by debugging and running the transformation tool. Finally, the context file can be loaded into the Event-B tool of Rodin after post-processing operation, and the context model in Fig.15(b) is generated.

Currently, the above model transformation tool of AdaptML2EventB cannot be fully automatic, and still needs human interaction with pre- and post-processing operations.

## 4.4 Event-B Refinement Patterns for SAS Systems

As shown in Subsection 4.3, the Event-B model provides a formal description of SAS systems. However, it lacks mechanism to depict complex self-adaptation activities, such as cooperation, branch, or self-adaptation loops. For this end, we have defined a set of Event-B refinement patterns for SAS systems. These refinement

patterns can be reused, and alleviate modeling difficulty.

### 4.4.1 Sequential Refinement Pattern

In order to explicitly depict sequential activities in SAS systems, we have defined the sequential refinement patterns. As shown in Fig.16(a), the abstract activity of abs can be decomposed into two sub-activities of con1 and con2, which execute sequentially. According to the mapping rules defined in Fig.7, this decomposition process can be formalized with corresponding events in Event-B, as shown in Fig.16(b). The events of con1 and con2 have refined the event abs sequentially, and this process is represented as (con1 $\rightarrow$ con2) refines abs.

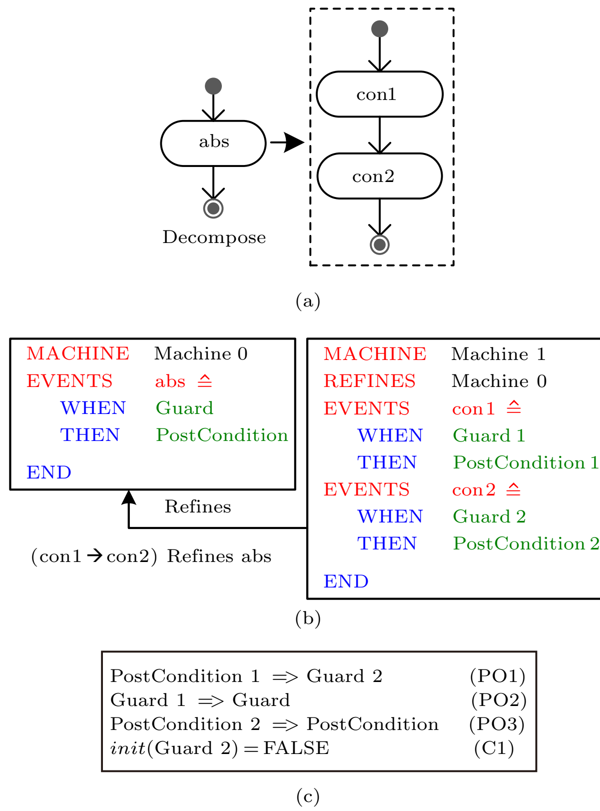



Fig.16. Refinement pattern for sequential activities. (a) Decomposed into sequential activities. (b) Event-B representation of sequential activities. (c) POs for refinement of sequential activities.

The refinement process should satisfy the POs and constraint in Fig.16(c), and the detailed derivation and proof processes are attached in Appendix A.1. Among them, PO1 indicates that actions of event con1 are sufficient to satisfy guards of event con2, that is, the former event can trigger the latter one; PO2 ensures that the concrete guard is stronger than the abstract one; PO3 ensures that the concrete events transform the concrete variables in a way which does not contradict the abstract event; C1 indicates that guard 2 should be set to be FALSE in the initialization event, as event con2 can only be triggered by event con1.

Taking the self-adaptation activity of Analyze for example, Analyze can be decomposed into two subprocesses, Receive and Compare, which are executed sequentially, as shown in Fig.17. The refinement from Analyze to Receive and Compare should always satisfy PO1, PO2, and PO3. Besides, the initial state of guard $triggerC = 1$ is FALSE, that is $triggerC = 0$.

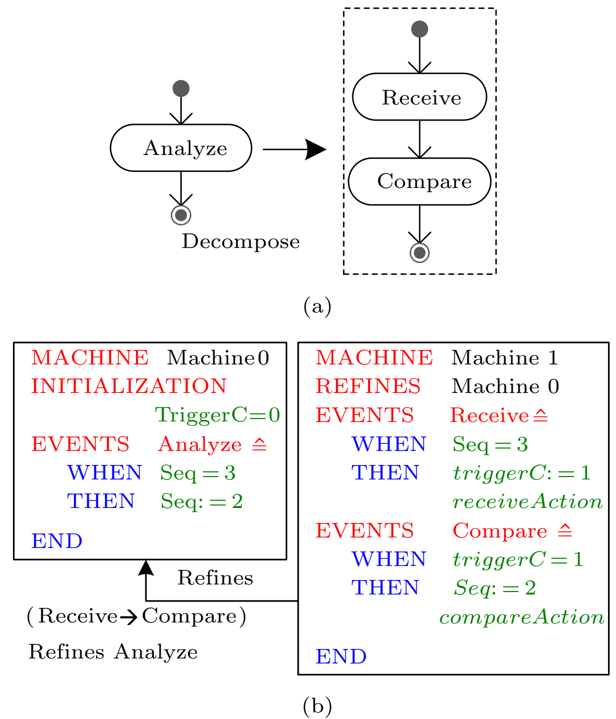



Fig.17. Example of sequential refinement. (a) Decomposing Analyze into Receive and Compare. (b) Event-B representation of Analyze, Receive and Compare.

### 4.4.2 Branch Refinement Pattern

The branch relationship is also quite common in SAS systems. For example, the plan process may consist of several alternative self-adaptation strategies, and only one is triggered at a time according to the guard constraints. In general, the branch refinement pattern is shown in Fig.18. The event abs of abstract machine can be refined into two events of concrete machine, con1 and con2, which are triggered alternatively, according to the guard of condition. For this end, POs of PO1–PO5 are derived and proved (refer to Appendix A.2).

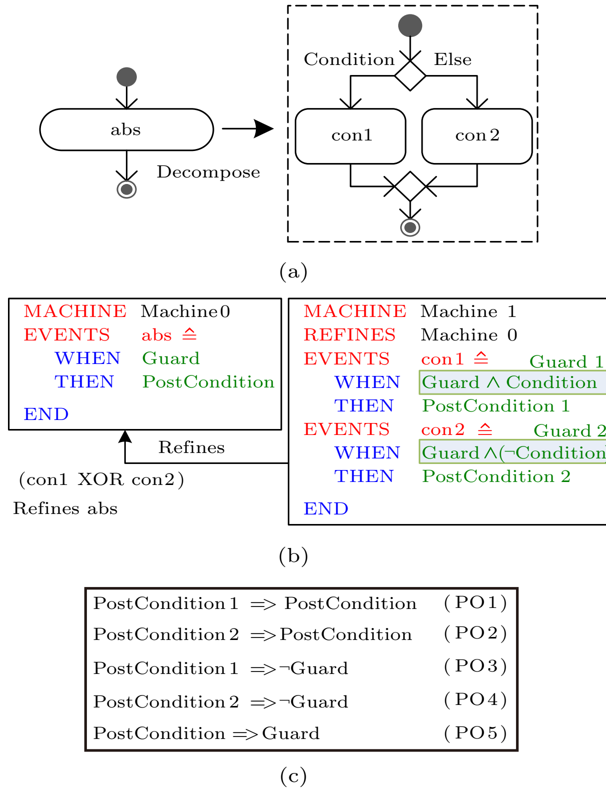



Fig. 18. Refinement pattern for branch activities. (a) Decomposed into branch activities. (b) Event-B representation of branch activities. (c) POs for refinement of branch activities.

PO1 and PO2 indicate that any one of the concrete actions is sufficient to satisfy guards of the abstract event. In addition, the two events can be triggered exclusively, and only once, according to PO3 and PO4 in Fig.18(c). Similarly, the abstract event abs can be triggered only once, according to PO5.

Taking the self-adaptation activity of plan for example, it can be refined into plan1 and plan2 according to different constraints, as shown in Fig.19. When the condition of $Utili > M$ satisfies, event plan1 would be triggered; otherwise, event plan2 would be triggered. The refinement processes always satisfy POs in Fig.18(c).

### 4.4.3 Cooperation Refinement Pattern

The cooperation relationship is another important relationship in SAS systems. For example, in the monitor process, it needs several kinds of sensors to collaboratively collect information from the running environment and the user requirements. And the satisfaction of the abstract monitor process needs the conjunction of both the sub-processes monitor 1 and monitor 2, but the execution order of the sub-processes is not restricted. To meet these requirements, the Event-B refinement pattern for cooperation activities is established, and represented as (con1 || con2) refines abs, as shown in Fig.20. The refined concrete events can be executed following the order of (con1; con2) or the order of (con1; con2), but their manipulated variables must be different.

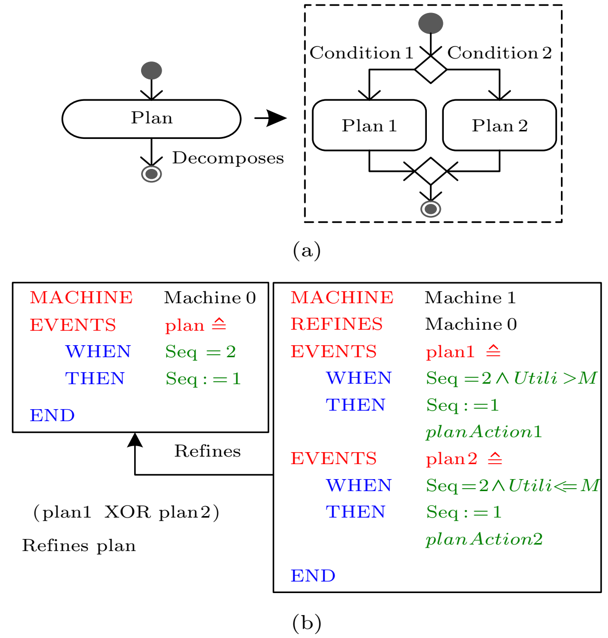



Fig. 19. Example of branch refinement. (a) Decomposing plan into plan 1 and plan 2. (b) Event-B representation of plan, plan 1 and plan 2.

PO1, PO2 and PO3 are derived, as shown in Fig.20(b), and the derivation processes are illustrated in Appendix A.3. PO1 and PO2 ensure that the concrete guards can satisfy the abstract one; and PO3 ensures that the conjunction execution of concrete events can satisfy the abstract event.

Taking the self-adaptation activity of monitor for example, as shown in Fig.21, the monitor activity is composed of the response-time monitor activity (i.e., monitorT) and the CPU utilization monitor activity (i.e., monitorU), which run collaboratively, as shown in Fig.21(a). This collaborative self-adaptation activity can be formalized with Event-B refinement pattern, as shown in Fig.21(b). Analysis indicates that the guards and actions of monitorT and monitorU all obey POs in Fig.20(c), and the refinement process is correct. Notably, the two concrete events manipulate different variables, i.e., *CurUtil* and *CurTime*. However, if they manipulate the same variable, the system would generate different results. For example, for $x = 2$, event monitorT runs $x := x + 1$, and event monitorU runs

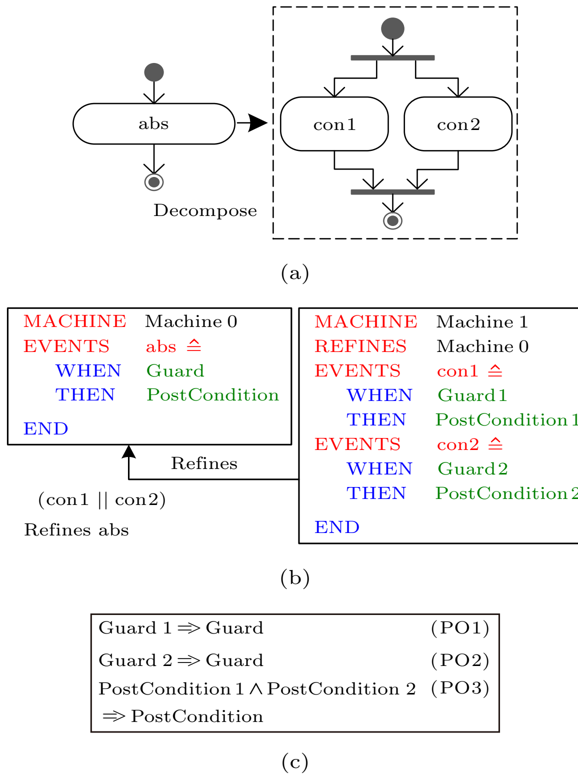



Fig.20. Refinement pattern for cooperation activities. (a) Decomposed into cooperation activities. (b) Event-B representation of cooperation activities. (c) POs for refinement of cooperation activities.

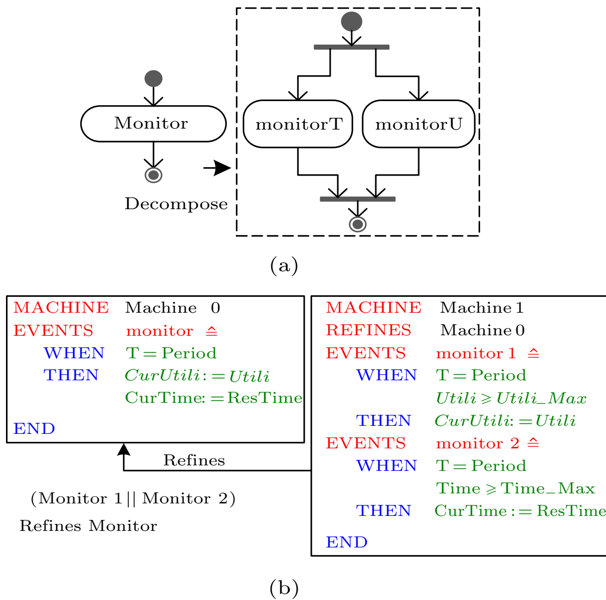



Fig.21. Example of cooperation refinement. (a) Decomposing monitor into monitorT and monitorU. (b) Event-B description of monitor, monitorT and monitorU.

$x := x \times 2$, if the execution order is (monitorT; monitorU), the result is $x = 6$; and if the execution order is (monitorU; monitorT), the result is $x = 5$. Therefore, the cooperation events can only manipulate the disjoint set of variables.

### 4.4.4 Refinement Pattern for Self-Adaptation Loop

The self-adaptation loop is treated as a first-class element in SAS systems. It can be decomposed into four processes, i.e., Monitor, Analyze, Plan and Execute, as shown in Fig.22(a). In order to formally describe the self-adaptation loop, we provide its Event-B

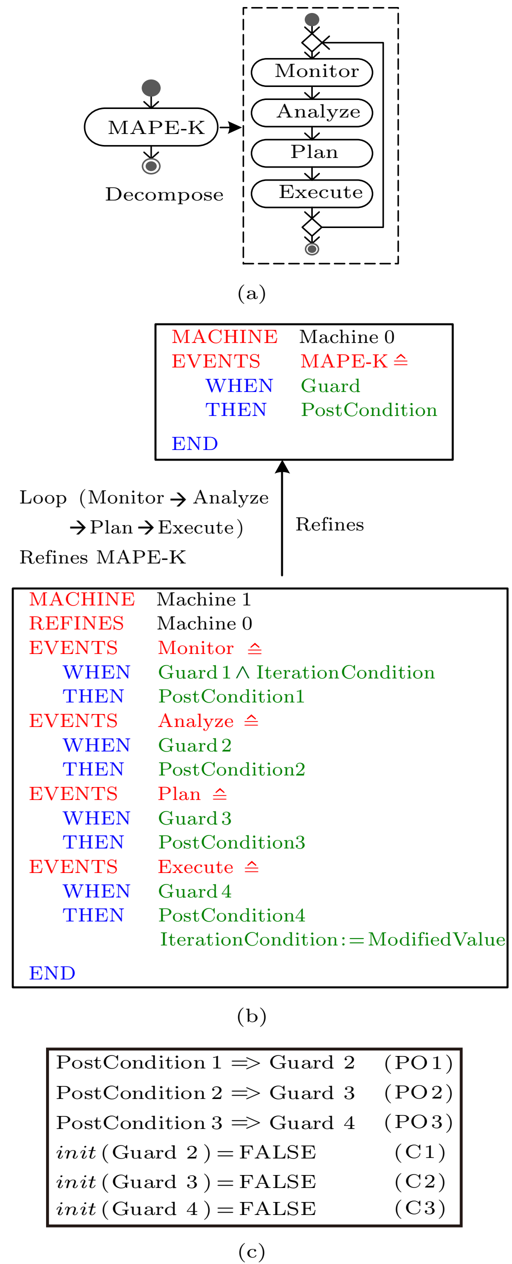



Fig.22. Refinement pattern for self-adaptation loop. (a) Decomposed into a self-adaptation loop. (b) Event-B representation of self-adaptation loop. (c) POs for refinement of self-adaptation loop.

model (cf. Fig.22(b)) and its corresponding refinement pattern (cf. Fig.22(c)). The loop is controlled by the IterationCondition of the guard. According to Fig.22(b), PO1, PO2 and PO3 all describe the sequential characteristics among the concrete events; and C1, C2 and C3 are the constraints restricted on the initial values, which guarantee that the events of analyze, plan, and execute can only be triggered by their corresponding preceding events.

As can be seen in Subsection 4.1 and Subsection 4.2, the AdaptML model provides an explicit description of SAS characteristics; while Event-B provides a correct-by-construction formalization for SAS systems. The model transformation can combine advantages of both models.

### 4.5 Formal Verification of SAS Systems Based on Event-B

In Subsection 4.4, we have established the Event-B model for SAS systems. In order to further analyze and verify the established model, we extract a set of properties in this subsection. The Event-B supporting tool of Rodin [18] provides two means for formal analysis: PO discharging and model checking. These two methods can complement with each other in practice. The extracted properties are as follows.

#### 4.5.1 *Model Correctness*

As shown in Table 2, model correctness of the SAS systems can be analyzed based on the Event-B model from three dimensions, i.e., variable definition, event transition, and model refinement.

**Table 2**. Model Correctness Analysis of SAS Systems

| Dimension | Description | Analysis Method |
|---|---|---|
| Variable definition | To analyze whether the defined variables are correct and practical | Invariant establishment |
| Event transition | To analyze whether the invariants are preserved during event transitions | PO discharging |
| Model refinement | To analyze whether the refined models are consistent with the abstract models | PO discharging |

As for variable definition, we verify if the defined variables conform to the syntax definition of Event-B, and verify if they conform to their physical meanings. Taking the adaptation scenario of ZNN.com for example, we would define the variable of CPU utilization as an integer of *Utili* in [0, 100]. In order to guarantee these two requirements, two invariants are established,

inv1: $Utili \in \mathbb{N}$ inv2: $Utili \leqslant 100$

As for event transition, we verify if the invariants are preserved during the transition processes, which is checked by verifying POs of invariants preservation [14]. For example, the variable of *Utili* would increase, i.e., $Utili := Utili + effect$, as user requests arise. In order to guarantee that the invariant of $inv2$ is still preserved, the following PO (called invariants preservation) needs to be checked,

$$A(s,c), I(s,c,v), G(s,c,x,v), Q\left(s,c,x,v,v'\right) \vdash I\left(s,c,v'\right),$$

where $A$ represents axiom, $I$ represents invariants, $G$ represents guards of events, and $Q$ represents the pre- and post-predicates. This PO requires that the invariants $I$ can still be satisfied after event transitions. In the above example, it requires that $inv1$ and $inv2$ can still be satisfied.

As for model refinement, we verify if the refined models are consistent with the abstract ones, which would be checked by verifying guard strengthening POs, simulation POs, gluing invariants [14], and the POs defined in Subsection 4.4. We would illustrate this point in the following example application part.

#### 4.5.2 *Safety Property of Self-Adaptation Logic*

Safety property means that nothing bad should happen during the program operation. As for SAS systems, the bad thing is that the system gets stuck in the deadlock state, and cannot respond to dynamic changes. In order to avoid this situation, an early analysis of deadlock freeness in the design phase is of vital importance for SAS systems.

On the one hand, the deadlock freeness property can be verified with model checking technique, which is supported by the ProB plug-in [19] of Rodin. The model checking method is carried out by pressing a single key and is easy to use. However, the model checking method suffers from the problem of state-space explosion, when system scale increases.

On the other hand, deadlock freeness can be verified by discharging proof obligations, i.e., proving that at least one event is enabled,

$$P(v) \Rightarrow \bigvee_{i} (\exists x \times G_i(x,v)),$$

where $P(v)$ represents all events, and $G_i(x,\ v)$ represents guards of each event.

#### 4.5.3 Effectiveness of Self-Adaptation Strategies

The effectiveness of self-adaptation strategies indicates that when the dynamic changes are out of range, corresponding self-adaptation strategies can be triggered and executed. This property can be checked with the LTL model checking technique of the Event-B model, as shown below,

$$G\{Changes \geqslant |error|\} \Rightarrow F\{Self_Adaptation_Strategy = 1\},$$

which means that when the context *Changes* are beyond the scope of $|error|$, the strategy of $Self_Adaptation_Strategy$ would be set to 1 and activated.

Taking the adaptation scenario of ZNN.com for example, when the CPU utilization is beyond the upper limit ($Utili \geqslant HighLevel$), the corresponding self-adaptation strategy of putting more servers into operation ($triggerAction = 1$) would be activated. This property is formalized as,

$$G\{Utili \geqslant HighLevel\} \Rightarrow F\{triggerAction = 1\},$$

which can be checked with the LTL model checking technique in Rodin.

## 5 Example Application

As discussed in Section 2, the ZNN.com example is a typical adaptation scenario, and its self-adaptation logic is complex and difficult to depict. This example can basically meet requirements for demonstrating and evaluating the effectiveness of the EasyModel approach. Therefore, we chose this example to qualitatively evaluate the effectiveness of the EasyModel approach.

This example application was carried out following three steps. Firstly, we established the structure model and the behaviour model of ZNN.com based on AdaptML and its supporting tool. Then, we transformed the AdaptML model of ZNN.com into the Event-B model based on the model transformation tool of AdaptML2EventB, and described complex behaviours of SAS systems based on the proposed Event-B refinement patterns. Finally, we formally analyzed corresponding properties of the ZNN.com based on the Event-B model.

### 5.1 Visual Modeling of ZNN.com

In order to provide an explicit description of the self-adaptation characteristics, we established the structure model and the behaviour model for ZNN.com with the adapt class diagram and the adapt activity diagram.

As shown in Fig.23, we established the adapt class diagram of ZNN.com from the abstract level to the concrete level, realizing an explicit description of the structure features and self-adaptation attributes of ZNN.com. First of all, we created the abstract model of Adapt Class Diagram_Level 0, consisting of the compo-

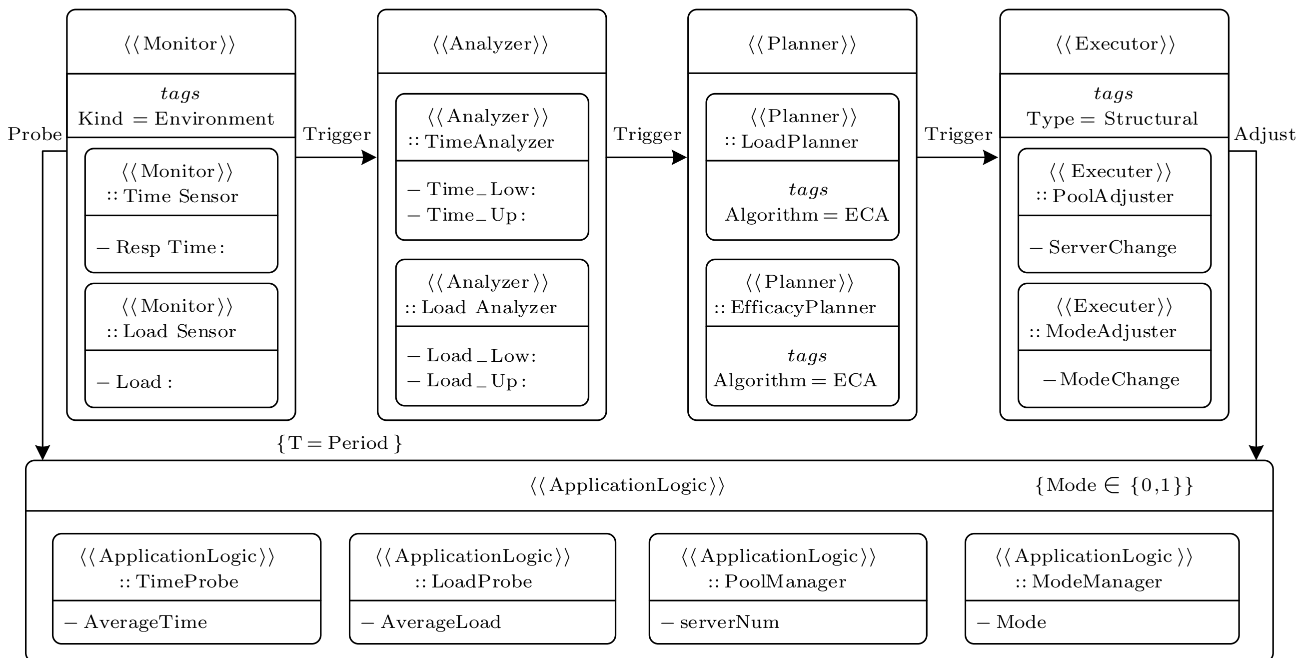



Fig.23. ZNN.com: Adapt Class Diagram_Level 2.

site adapt classes of <<Monitor>>, <<Analyzer>>, <<Planner>>, <<Executor>>, and <<Application-Logic>>. The detailed business logic of ZNN.com was treated as a black box hidden in the <<Application-Logic>>. This abstract model provides an overview of the fundamental elements and interactions of the self-adaptive ZNN.com system. Then, we decomposed the above composite adapt class diagram and created the model of Adapt Class Diagram_Level 1. This model specifies structures of ZNN.com in more detail. For example, the abstract adapt class of <<Monitor>> was refined into two sub-classes of TimeSensor and LoadSensor. Finally, we further refined the above model by adding self-adaptation attributes (e.g., RespTime), tagged values (e.g., *Type* = *Structural*) and constraints, creating a more concrete model of Adapt Class Diagram_Level 2, as shown in Fig.23. The tagged values depicted corresponding self-adaptation features. For example, the tagged value of $Type = Structural$ means that the self-adaptive process was implemented by adjusting system structures. The constraints are attached with self-adaptation attributes. For example, *Mode* $\in \{0,\ 1\}$ limits the range of variable *Mode*. From the above modeling processes, we can conclude that the adapt class diagram provides an explicit description of self-adaptation structures and self-adaptation attributes, and can fulfill the modeling requirements of R1.

Similarly, we established the behaviour model for ZNN.com on basis of the adapt activity diagram, as shown in Fig.24. Firstly, we established an abstract model called Adapt Activity Diagram_Level 0, provid-

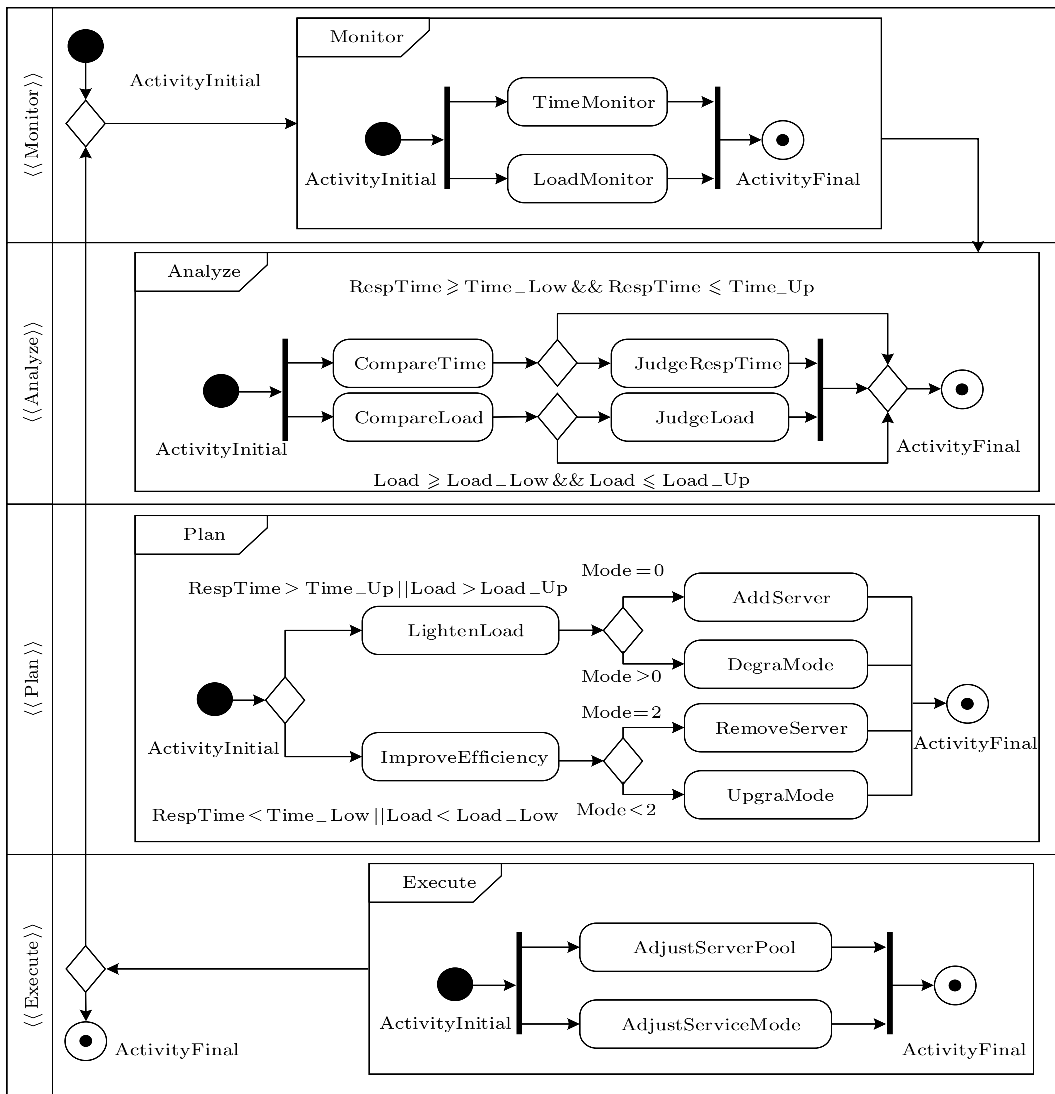



Fig.24. ZNN.com: Adapt Activity Diagram_Level 2.

ing an explicit description of self-adaptation processes and the self-adaptation loop of Monitor-Analyze-Plan-Execute. Secondly, we decomposed each abstract self-adaptation activity into sub-activities, and created a more concrete behaviour model called Adapt Activity Diagram_Level 1. Taking the abstract activity of Monitor for example, it was further decomposed into two concrete activities called TimeMonitor and LoadMonitor, realizing the response-time monitoring of servers and the load-level monitoring of servers respectively. Finally, more implementation details, such as triggering conditions, were enriched to the above adapt activity model, and created the concrete model of Adapt Activity Diagram_Level 2, i.e., Fig.24. From the above modeling processes, it can be seen that the adapt activity diagram provides an explicit description of self-adaptation behaviours and the self-adaptation loops. Therefore, the adapt activity diagram can fulfill the modeling requirements of R2 and R3. In conclusion, the above example application indicates that the adapt class diagram and the adapt activity diagram can fulfill the modeling requirements of R1, R2 and R3, and can basically satisfy the visual modeling requirements of SAS systems. Besides, the adapt class diagram and the adapt activity diagram were established following a step-by-step process, and they were created by gradually enriching implementation details of ZNN.com. The above step-by-step modeling mechanism is in line with the gradual process of human thinking and understanding, and can effectively handle system complexity (R4) of SAS systems and alleviate modeling difficulty.

### 5.2 Formal Modeling of ZNN.com

The formal Event-B model of ZNN.com was transformed from the adapt class diagram and the adapt activity diagram in Subsection 5.1.

As shown in Fig.25, the Event-B model of ZNN_Context1 was transformed from the model of Adapt Class Diagram_Level 1. It formally describes the main components of SAS systems. The model of ZNN_Context 2 was transformed from the model of Adapt Class Diagram_Level 2, and it extends the ZNN_Context1 model by enriching self-adaptation attributes, such as Time_Up (the upper bound of variable *RespTime*). In addition, a constant called RandomTime was added to the ZNN_Context2 model, and this constant was used to define simulation values of variable *RespTime*.

The model of ZNN_Machine1 was transformed from Adapt Activity Diagram_Level 1. It describes the main self-adaptation activities, such as, Monitor and Analyze. In ZNN_Machine1, we defined the variable of *Seq*, which was used to control the triggering order of events, such as the sequential order. ZNN_Machine2 was transformed from Adapt Activity Diagram_Level 2, and it refines ZNN_Machine1 by enriching variables and self-adaptation activities, such as TimeMonitor and AddServer. The expression of $RespTime := RandomTime$ was used to simulate the monitoring response-time of services. Besides, we introduced the auxiliary variables such as $triggerPlanL$ to control the triggering sequence of self-adaptation activities. In the ZNN_Machine2 model, refinement patterns were used to specify complex self-adaptation activities. For example, the events of TimeMonitor and LoadMonitor both refine the abstract event of Monitor, and they executed collaboratively. Thus, the refinement from Monitor to TimeMonitor and LoadMonitor was carried out according to the cooperation refinement pattern. Similarly, the sequential refinement pattern, the branch refinement pattern and the self-adaptation loop refinement pattern were used to formally describe self-adaptation activities.

From the model of ZNN_Machine2, it can be seen that the self-adaptive logic of ZNN.com is complex and difficult to depict. Taking the load balancing part for example, it involves in the processes of LoadMonitor, CompareLoad, JudgeLoad, LightenLoad and so on, and several processes even need to be refined into sub-processes, which cannot be easily depicted. However, with the proposed Event-B refinement patterns, the complex self-adaptive ZNN.com system can be described following a stepwise refinement modeling process, and this stepwise refinement modeling mechanism can effectively deal with system complexity (R4).

### 5.3 Formal Verification of ZNN.com

In this subsection, we would analyze properties of the self-adaptive ZNN.com system from three aspects, by combining model checking technique with PO discharging technique.

*Model Correctness.* Model correctness is guaranteed by checking POs established in the modeling processes. As shown in Table 3, the total modeling process of the self-adaptive ZNN.com system resulted in 41 POs, of which 39 (95.1%) were proved automatically with the Rodin tool, and the remaining 2 (4.9%) POs passed validation after replenishing the established

```
CONTEXT
ZNN_Context1
SETS
Monitor
Analyzer
Planner
Executor
ApplicationLogic
CONSTANTS
TimeSensor
LoadSensor
TimeAnalyzer
LoadAnalyzer
TimePlanner
LoadPlanner
TimeAdjuster
LoadAdjuster
TimeProbe
LoadProbe
PoolManager
ModeManager
probe
triggerMA
triggerAP
triggerPE
AXIOMS
Generalization1:
partition(Monitor, {TimeSensor}, {LoadSensor})
Generalization2:
partition(Analyzer,{TimeAnalyzer},{LoadAnalyzer})
Generalization3:
partition(Planner, {TimePlanner}, {LoadPlanner})
Generalization4:
Partition(Executor, {TimeAdjuster}, {LoadAdjuster})
Generalization5:
Partition(ApplicationLogic , {TimeProbe}, {LoadProbe},
{PoolManager}, {ModeManager})
Relationship_probe:
probe ∈ Monitor↔ApplicationLogic
Relationship_triggerMA:
triggerMA ∈ Monitor↔Analyzer
Relationship_triggerAP:
triggerAP ∈ Analyzer↔Planner
Relationship_triggerPE:
triggerPE ∈ Planner↔Executor
Relationship_adjust:
adjust ∈ Executor↔ApplicationLogic
END
```

See

```
CONTEXT
ZNN_Context2
EXTENDS
ZNN_Context1
CONSTANTS
Time_Up
Time_Low
⋮
RandomTime
⋮
AXIOMS
axm1:   Time_Up = 90
axm2:   Time_Low = 5
⋮
axm6:   RandomTime ∈{3, 32, 62, 82, 89, 99}
⋮
END
```

Extend

```
MACHINE
ZNN_Machine1
REFINES ZNN_Machine0
SEES
ZNN_Context1
VARIABLES
Seq
INVARIANTS
inv1:  Seq ∈ {1,2,3,4}
⋮
EVENTS
INITIALISATION
  act1:  Seq := 4
Monitor ≙
REFINES MAPE
  grd1:  Seq=4
THEN
  act1:  Seq:=3
END
Analyze ≙
REFINES MAPE
  grd1:  Seq=3
THEN
  act1:  Seq:=2
END
AnalyzeEnd
REFINES MAPE
  grd1:  Seq=3
THEN
  act1:  Seq:=1
END
Plan ≙
REFINES MAPE
  grd1:  Seq=2
THEN
  act1:  Seq:=1
END
Execute ≙
REFINES MAPE
  grd1:  Seq=1
THEN
  act1:  Seq:=4
END
END
```

Self-Adaptation Loop

Refine

**Replenishing Model Correctness by Checking POs (Subsection 5.3)**

*Guards Neglected in ZNN_Machine2*

**Mode+ModeChange ≤ 1**

**SerNum+SerChange ≤ 100**

```
MACHINE  ZNN_Machine2
REFINES  ZNN_Machine1
SEES  ZNN_Context2
VARIABLES  RespTime, Load ... Load_DU, Time_DL,
           ... SeqAnalyzeT, SeqAnalyzeL, ..., ServerChange
INVARIANTS inv 18 Mode≤1...  inv 23 SerNum≤100 ...
EVENTS     ...
TimeMonitor ≙
REFINES Monitor
WHEN grd1:  Seq=4
THEN  act1:  Seq:=3  act2 :  RespTime:=RandomTime
END
LoadMonitor ≙
REFINES Monitor
WHEN grd1:  Seq=4
THEN  act1:  Seq:=3  act2 :  Load:=RandomLoad
END
CompareTime ≙
REFINES Analyze
WHEN grd1:  Seq=3
     grd2:  (RespTime>Time_Up)∨(RespTime<Time_Low)
THEN  act1:  SeqAnalyzeT := 1
END
JudgeRespTime ≙
REFINES Analyze
WHEN grd1:  SeqAnalyzeT=1
THEN  act1:  triggerPlan:=1  act2: Time_DU:=RespTime−Time_Up
      act3:  Time_DL:=RespTime−Time_Low  act4: Seq:=2
END
CompareLoad ≙
REFINES Analyze
WHEN grd1:  Seq=3
     grd2:  (Load>Load_Up)∨(Load<Load_Low)
THEN  act1:  SeqAnalyzeL:=1
END
JudgeLoad ≙
REFINES Analyze
WHEN grd1:  SeqAnalyzeL=1
THEN  act1:  triggerPlan:=1  act2: Load_DU:=Loadí Load_Up
      act3:  Load_DL:=Load−Load_Low  act4: Seq:=2
END
CompareTimeEnd ≙
REFINES AnalyzeEnd
WHEN grd1:  Seq=3
     grd2:  (RespTime≥Time_Low)∧(RespTime≤Time_Up)
THEN  act1:  Seq:=1
END
CompareLoadEnd ≙
REFINES AnalyzeEnd
WHEN grd1:  Seq=3
     grd2:  (Load ≥ Load_Low)∧(Load ≤ Load_Up)
THEN  act1:  Seq:=1
END
LightenLoad ≙
REFINES Plan
WHEN grd1:  Seq=2
     grd2:  triggerPlan=1   grd3: (Time_DU>0)∨(Load_DU>0)
THEN  act1:  triggerPlanL:=1
END
ImproveEfficency ≙
REFINES Plan
WHEN grd1:  Seq=2
     grd2:  triggerPlan=1   grd3: (Time_DL<0)∨(Load_DL<0)
THEN  act1:  triggerPlanI:=1
END
AddServer ≙
REFINES Plan
WHEN grd1:  Mode≤0  grd2:  triggerPlanL=1
THEN  act1:  Seq:=1   act2:  SerChange:=1
END
DegraMode ≙
REFINES Plan
WHEN grd1:  Mode>0  grd2:  triggerPlanL=1
THEN  act1:  Seq:=1   act2:  ModeChange:=−1
END
RemoveServer ≙
REFINES Plan
WHEN grd1:  Mode≥2  grd2:  triggerPlanI=1
THEN  act1:  Seq:=1   act2:  SerChange:=−1
END
UpgraMode ≙
REFINES Plan
WHEN grd1:  Mode<2  grd2:  triggerPlanI=1
THEN  act1:  Seq:=1   act2:  ModeChange:=1
END
AdjustServiceMode ≙
REFINES Execute
WHEN grd1:  Seq=1  grd2:  Mode+ModeChange≤1
THEN  act1:  Seq:=4 act2:  Mode:=Mode+ModeChange
END
AdjustServerPool ≙
REFINES Execute
WHEN grd1:  Seq=1  grd2:  SerNum+SerChange≤100
THEN  act1:  Seq:=4 act2:  SerNum:=SerNum+SerChange
END
END
```

Parallel Node

Sequential Node

Branch Node

CompareTime

CompareTimeEnd

Fig.25. Fragment of the Event-B model for ZNN.com.

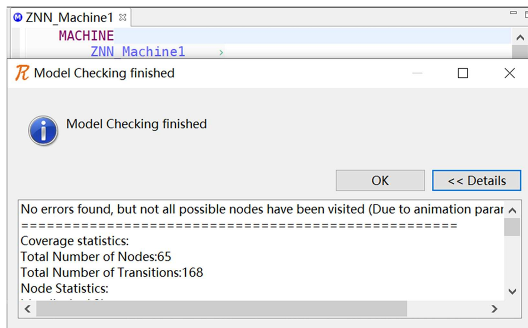
model, as shown in Fig.25. This was because when events AdjustServerMode and AdjustServerPool executed the actions of $Mode := Mode + ModeChange$ and $SerNum := SerNum+SerChange$, the invariants of $Mode \leqslant 1$ and $SerNum \leqslant 100$ would be violated, respectively. The POs discharging technique helped reveal the above deficiency. To remedy these errors, we added two additional guards of $Mode+ModeChange \leqslant 1$ and $SerNum + SerChange \leqslant 100$. Afterwards, all POs were proved correctly. Therefore, the correct-by-construction mechanism of the Event-B model can help guarantee the model correctness of SAS systems (R5).

**Table 3**. Model Correctness Verifying by Discharging POs

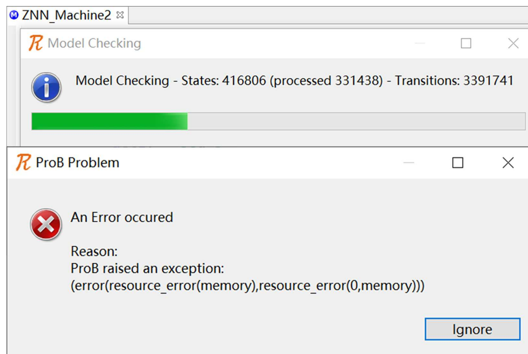
| Model | Number of POs | Automatic POs | Interactive POs |
|---|---|---|---|
| Abstract model | 13 | 13 (100%) | 0 (0.0%) |
| 1st refinement | 22 | 20 (90.9%) | 2 (9.1%) |
| 2nd refinement | 6 | 6 (100.0%) | 0 (0.0%) |
| Total | 41 | 39(95.1%) | 2 (4.9%) |

*Safety Property of Self-Adaptation Logic.* The ZNN.com is responsible to provide continuous service for the public, and we do not expect that it gets stuck in the deadlock state. For this end, we employed the model checking and PO discharging technique to guarantee this property.

Model checking is carried out with the ProB [19] plug-in of Rodin. We checked the machine of ZNN_Machine1 and ZNN_Machine2, and the results are shown in Figs.26(a) and 25(b), respectively. According to Fig.25, ZNN_Machine1 is an abstract and simple model, and it has numerable states. Thus, it is easy to check the deadlock freeness property. However, the ZNN_Machine1 machine cannot be easily checked using ProB. This is because this model has high system




(a)

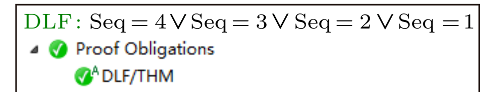



(b)

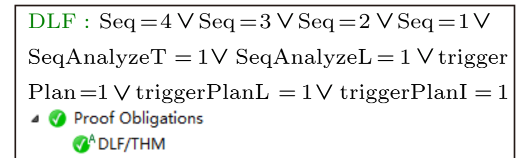
Fig. 26. Deadlock freeness verification with model checking. (a) Model checking of ZNN.com Level_1. (b) Model checking of ZNN.com Level_2.

complexity, and the problem of state-space explosion arises, as shown in Fig.26(b).

In order to make up the inefficiency of mode checking, we have employed POs discharging technique to verify the deadlock freeness property. According to Subsection 4.5.2, if the guards of at least one event are enabled, the deadlock freeness property can be guaranteed. Therefore, this property can be verified by directly discharging POs, as shown in Fig.27.



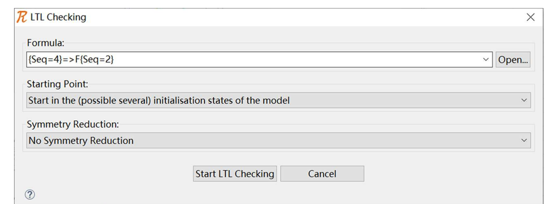

(a)



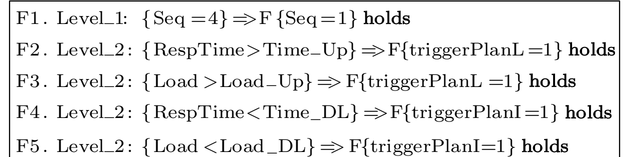

(b)

Fig.27. Deadlock freeness verification with PO discharging. (a) POs discharging of ZNN.com Level_1. (b) POs discharging of ZNN.com Level_2.

*Effectiveness of Self-Adaptation Strategies.* As for self-adaptive ZNN.com, the effectiveness of self-adaptation strategies means that when the service response time or the load level of servers is out of the specified operating range, the self-adaptation actions can be triggered. We have expressed the above properties with the LTL formula, and verified them using ProB, as shown in Fig.28.




(a)




(b)

Fig.28. Verifying effectiveness of self-adaptation strategies. (a) Model checking with ProB. (b) Model checking results.

As shown in Fig.28(b), F1 was property of ZNN_Machine1, and F2–F5 were properties of

ZNN_Machine2. Among them, F1 meant that when Monitor was triggered, the Execute event would be eventually triggered. F2–F5 meant that when the service response time or the load level of servers was out of range, the corresponding self-adaptation strategies would be triggered. The above self-adaptation properties were directly checked using ProB, and all properties held.

### 5.4 Discussion

The above example application has qualitatively demonstrated the effectiveness of the EasyModel approach in modeling and formal verification of SAS systems. However, compared with the related approaches, there is still a lack of quantitative metrics to evaluate the advantages of the EasyModel approach.

## 6 Experimental Evaluation

The objective of this experiment was to quantitatively evaluate the effectiveness of the EasyModel approach. And the effectiveness would be examined from concrete evaluation metrics (e.g., explicit modeling, complexity handling, and correctness assurance).

### 6.1 Experiment Planning and Design

This subsection presents the detailed planning and design of this experiment, including participants, evaluation metrics, case selection, and the experiment design.

#### *6.1.1 Experiment Participants*

This experiment was conducted at Army Engineering University of PLA③, Nanjing, China. The participants were 30 graduate students, who were randomly selected from a graduate course called "Advanced Software Engineering". They were told beforehand that they had free choice to participate or not in the experiment. The participants had similar technical backgrounds, and went through a specific training for this experiment. In this experiment, the above participants would be divided into six groups at random, and each group would be demanded to model and verify SAS systems with one of the six kinds of formal methods (i.e., the EasyModel approach proposed in this paper, the ACML approach [4], the Event-B approach [27], the MV4SAS approach [15] which integrates the extended UML model with the network of timed automata, the EUREMA approach [28, 29], and the approach of MAPE-K Formal Templates [7]).

Besides, two domain experts, who were not involved in the EasyModel project, were asked to score the above 30 participants according to a set of concrete criteria. The scoring processes followed a double-blind procedure.

#### *6.1.2 Evaluation Metrics Selection*

Based on the modeling requirements (R) proposed in Subsection 3.1, we established a set of evaluation metrics for SAS modeling approaches, as shown in Fig.29.

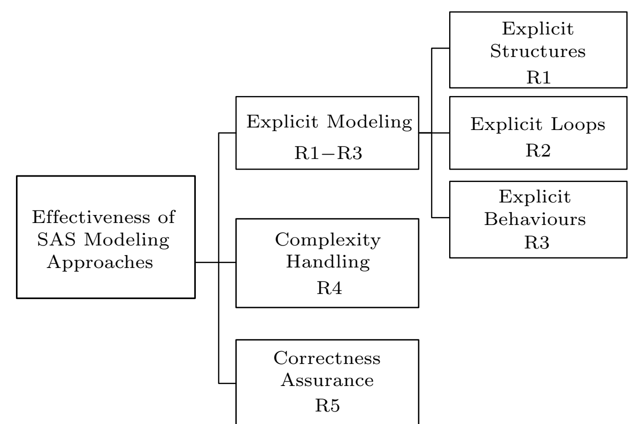



Fig.29. Evaluation metrics for SAS modeling approaches.

The evaluation metrics would evaluate the effectiveness of SAS modeling approaches from three indexes, explicit modeling, complexity handling and correctness assurance. Among them, the index of explicit modeling consists of three sub-indexes, namely explicit structures, explicit behaviours, and explicit self-adaptation loops. These indexes correspond to the modeling requirements proposed in Subsection 3.1.

#### *6.1.3 Evaluation Case Selection*

In this experimental evaluation, we selected the ZNN.com example as the application case. After the example application in Section 5, we checked the validation of the ZNN.com example in evaluating the SAS modeling and verification approaches, by answering the following two questions.

1) Is the ZNN.com example representative enough to evaluate the SAS modeling approaches?

2) Can the ZNN.com example cover all aspects of the modeling requirements (R) proposed in Subsection 3.1?

③http://www.81.cn/jwzl/2017-06/05/content_7748283.htm, July 2020.

As for question 1, the ZNN.com example is a classical scenario of SAS systems, and has been widely accepted to evaluate SAS systems in the community [4, 15, 24, 29, 30]. Therefore, it is representative enough to evaluate the SAS modeling approaches.

As for question 2, the ZNN.com application in Section 5 demonstrated explicit structure modeling (i.e., Subsection 5.1, Fig.23), explicit behaviour modeling (i.e., Subsection 5.1, Fig.24), explicit self-adaptation loop modeling (i.e., Subsection 5.1, Fig.23 and Fig.24), complexity handling (i.e., the decomposition processes in Fig.23 and Fig.24, and the stepwise refinement processes in Fig.25), and correctness assurance (i.e., the formal verification processes in Subsection 5.3). Thus, it can basically cover all aspects of the modeling requirements proposed in Subsection 3.1.

Based on the above analysis, we selected the ZNN.com example as the application case in the following experimental evaluation.

#### *6.1.4 Experimental Design*

According to the experimental design in Table 4, the experiment was carried out following three steps. In the first step, all the six groups of participants were equally trained to understand the six kinds of SAS modeling approaches. In the second phase, each of the participants was asked to finish designing, modeling and formal verification of the self-adaptive ZNN.com example with one specific formal approach, and to submit a project independently. Finally, their projects would be scored by two domain experts in a double-blind way.

### 6.2 Experiment Operation

According to the experimental design in Subsection 6.1.4, the experiment was carried out following three steps: training and grouping, actual conducting, and scoring.

As for training, the participants received a 30-hour. training course from the corresponding author of this paper, including the following topics: 1) detailed explanation on SAS systems and the modeling requirements; 2) detailed explanation and practice on formal methods like UML model, Event-B method, and timed automata model; 3) deep literature study on SAS system modeling and formal analysis; 4) other topics on advanced software engineering.

After training, the participants were divided into six groups at random. They were asked to finish modeling and verifying the ZNN.com example with different modeling approaches (the grouping and corresponding modeling approach were shown in Table 4) independently. Afterwards, each one was asked to submit a project.

As for scoring, two domain experts were asked to score the submitted projects according to the metrics in Fig.29, following a double-blind procedure. Each project was scored by two experts, respectively, and the final scores took the average value. The final evaluation results are shown in Table 5.

**Table 4**. Design of the Experiment

| Step | Group 1 | Group 2 | Group 3 | Group 4 | Group 5 | Group 6 |
|---|---|---|---|---|---|---|
| 1st step: training | The same training | The same training | The same training | The same training | The same training | The same training |
| 2nd step: actual conducting | EasyModel | ACML [4] | Event-B [27] | MV4SAS [15] | EUREMA [28, 29] | MAPE-K formal templates [7] |
| 3rd step: scoring | A double-blind review | A double-blind review | A double-blind review | A double-blind review | A double-blind review | A double-blind review |

**Table 5**. Experiment Evaluation Results

| Index | EasyModel | ACML [4] | Event-B [27] | MV4SAS [15] | EUREMA [28, 29] | MAPE-K Formal Templates [7] |
|---|---|---|---|---|---|---|
| Explicit Structures (R1) | 5.0 | 4.5 | 0.0 | 4.5 | 1.0 | 0.0 |
| Explicit Behaviors (R3) | 4.5 | 5.0 | 2.0 | 4.5 | 5.0 | 5.0 |
| Explicit Loops (R2) | 4.0 | 0.5 | 0.0 | 4.0 | 4.5 | 3.0 |
| Complexity Handling (R4) | 4.5 | 1.0 | 3.5 | 0.0 | 3.0 | 0.5 |
| Correctness Assurance (R5) | 5.0 | 3.0 | 5.0 | 3.0 | 0.0 | 4.0 |
| Total Scores (average) | 23.0 | 14.0 | 10.5 | 16.0 | 13.5 | 12.5 |

Notes: For scoring: 5 represents strong support; 3 represents support; 1 represents weak support; 0 represents no support.

### 6.3 Experimental Results

According to the evaluation results in Table 5, we compared the EasyModel approach with the other five kinds of related approaches from each concrete index, and the results were shown in Fig.30.

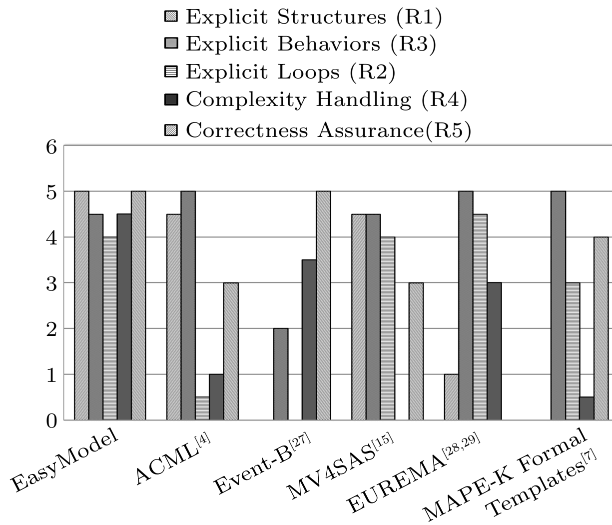



Fig.30. Analyzing related modeling approaches in detail.

#### 6.3.1 Evaluation of Each Index

1) *Aspect of Explicit Structures* ($R1$). The result was that the EasyModel approach, the ACML approach, and the MV4SAS approach all achieved satisfactory results. They can provide an explicit description of the structure characteristics of SAS systems. By contrast, the Event-B approach, the EUREMA approach, and the MAPE-K formal templates provided weak or no support on depicting structures of SAS systems. Thus, from the perspective of explicit structures (R1), the EasyModel approach is an effective SAS modeling approach.

2) *Aspect of Explicit Behaviours* ($R3$). As can be seen in Fig.30, all the approaches, except standard Event-B, gained excellent results in describing self-adaptation behaviours. This was because the standard Event-B model lacks refinement patterns to depict complex self-adaptation behaviours explicitly. But the EasyModel approach makes up this weak point by defining a set of new refinement patterns. Therefore, from the perspective of explicit behaviours, EasyModel is an effective SAS modeling approach.

3) *Aspect of Explicit Self-Adaptation Loops* ($R2$). The ACML approach and the Event-B approach are largely unable to describe self-adaptation loops. In comparison, the EasyModel approach, the MV4SAS approach, the EUREMA approach, and the MAPE-K formal templates all can explicitly describe the self-adaptation loops.

4) *Aspect of Complexity Handling* ($R4$). As shown in Fig.30, the EasyModel approach, the standard Event-B approach, and the EUREMA approach all can effectively handle the modeling complexity of SAS systems, and the EasyModel approach achieved the best results. It was because that the EasyModel approach tackles the modeling complexity from two aspects: the hierarchy structures of AdaptML and the refinement mechanism of the Event-B model. In comparison, the standard Event-B handles modeling complexity only with refinement mechanism.

5) *Aspect of Correctness Assurance* ($R5$). All approaches except EUREMA are formal methods. As a sequence, they can provide correctness assurance to some extent. The difference is that the EasyModel approach systematically defines and verifies three kinds of properties for SAS systems: model correctness, the safety property of self-adaptation logic, and the effectiveness of self-adaptation strategies. As a sequence, the EasyModel approach achieved better results than the other four kinds of approaches.

From the first three points, it can be seen that the AdaptML profile (i.e., the extended UML of EasyModel) is indispensable in designing SAS systems, as for its merit of being intuitive and easy to use; from the last two points, it can be seen that the correct-by-construction and stepwise refinement of the Event-B model also play an essential role in formal modeling and verification of SAS systems.

#### 6.3.2 Comprehensive Evaluation

The comprehensive evaluation results were shown in Fig.31. Overall, the EasyModel approach scored 23 (the total score was 25), and it behaved better than the other five approaches.

Especially, the EasyModel approach gained much better results than the ACML approach and the Event-B approach. This illustrates that only the extended UML or the Event-B model is insufficient to fulfill the modeling requirements of SAS systems, for two reasons. Firstly, both UML and the extended UML are semi-formal models, and cannot be analyzed or verified. Secondly, the Event-B model is strong enough to support the refinement modeling and the formal analysis of SAS systems, but it is created based on the set

theory and lacks visual modeling elements, which make it difficult to master by software engineers.

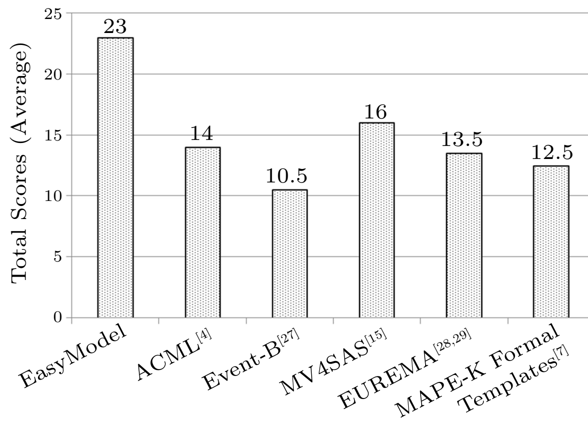



Fig.31. Comprehensive evaluation results.

As a conclusion, only the EasyModel approach, which integrates the extended UML model and the Event-B model, can fulfill the modeling requirements (R) proposed in Subsection 3.1.

### 6.4 Discussion

*Strengths*. The above example application and the experimental evaluation have demonstrated that the EasyModel approach has incorporated the intuitive merit of UML and the correct-by-construction merit of Event-B, providing an explicit description on self-adaptation structures, behaviours and attributes. Furthermore, it provides a stepwise refinement modeling way to handle system complexity and reduce modeling difficulty.

*Weakness*. The above experimental evaluation also reveals some deficiency of the EasyModel approach. Firstly, as shown in Fig.30, the EasyModel approach can still be improved in describing the self-adaptation loops. This is because EasyModel currently focuses on single loop description of SAS systems, and still lacks modeling infrastructures to depict interactions between multi-loops. Secondly, the EasyModel approach focuses on modeling and formal verification of the self-adaptation logic and treats the application logic as a black box. This approach can only guarantee that the self-adaptation logic is workable, but it cannot be used to optimize the self-adaptation strategies. To make up this deficiency, we made a preliminary attempt by integrating model checking with model simulation in optimizing self-adaptation strategies [31]. However, the real-world SAS scenario is far more complex, and further work needs to be done.

## 7 Related Work

Recent researches provide various approaches that support the modeling and designing of SAS systems. In lieu of enumerating all the related studies, we refer the readers to [3, 32] for a comprehensive understanding of this field. This study focuses on modeling and formal verification approaches for SAS systems, and we review studies relevant to the EasyModel approach.

### 7.1 Visual Modeling Approaches for SAS

*UML-Based Modeling Approaches*. With its advantages of being intuitive, easy to use and fine compatible, UML has been widely used to model SAS systems. In order to support the explicit modeling of self-adaptation control loops, Abuseta and Swesi [33] proposed a set of UML-based design patterns. This work treated feedback loops as first-class element, and explicitly described the feedback loops and the interplay between multiple feedback loops (R2). Said *et al.* [6] presented five design patterns for self-adaptive real-time embedded systems. The design patterns allow dealing with concurrency and real-time features with generic and reusable design patterns, lightening modeling complexity (R4), and reducing development cost. Becker *et al.* [34, 35] extended the UML model hierarchically, and proposed a mechatronic UML model for self-adaptive electromechanical system. This method can handle system complexity by specifying SAS systems from the abstract models into the concrete models (R4). In addition, we proposed the FAME framework [5] by extending UML with fuzzy control concepts. The FAME approach focuses on specifying self-adaptive entities and attributes of the fuzzy self-adaptive software systems (R1). Furthermore, da Silva *et al.* [36] provided a systematic review on existing UML modeling methods for SAS systems. To summarize, the above UML-based approaches exhibit superiority in explicitly describing self-adaptation characteristics. However, they are limited in specifying and verifying self-adaptation activities (R3) and properties (R5). Different from existing work, the proposed EasyModel approach has integrated the extended UML model with the Event-B model, making the self-adaptation attributes (R1), feedback loops (R2), and self-adaptation activities (R3) explicit, lightening modeling complexity (R4), and providing guarantees for system behaviours (R5).

*Domain-Specific Modeling Approaches.* Apart from the UML-based modeling approaches, there are several domain-specific modeling approaches for SAS systems. Vogel and Giese[28,29] proposed a runtime and executable modeling approach called EUREMA. It focuses on specifying interactions between runtime models, self-adaptation loops (R2), and self-adaptation activities (R3). In [37] and [38], feature models are used to specify self-adaptation behaviours. Besides, the agent-oriented models, such as [39] and [40], have also been employed to design and optimize SAS systems. These novel approaches are valuable and instructive, but they are created based on domain specific models that are less popular with the general software engineers. By comparison, the EasyModel approach is created based on UML, which is more understandable and acceptable.

### 7.2 Formal Approaches for SAS

*Approaches Based on Automaton Model.* Focusing on formal analysis of SAS systems, the automaton model has been widely used, because of its features of being intuitive and easy to use. On basis of the timed automata model, the AdaptWise research group of Linnaeus University④ proposed the MAPE-K formal templates[7], the ActivFORMS approach[41] and the eARF approach[42,43] for SAS systems. The limitation is that model checking of the above formal models encounters the problem of state-space explosion when the system scale increases. By contrast, our EasyModel approach integrates model checking with PO discharging, which can ease system complexity (R4), and avoid the above problems.

*Approaches Based on Probabilistic Models.* The probabilistic models, such as Bayesian Network[11] and Markov Decision Process[44,45], are gaining high popularity in handling uncertainty within SAS systems. However, the above probabilistic models lack scalability, and cannot be used in complex software systems. In comparison, the Event-B model in our EasyModel approach owns the advantage of stepwise refinement, and performs well in handling system complexity (R4).

*Refinement-Based Approaches.* Göthel *et al.*[46] proposed a CSP-based formal approach for SAS systems, which can decompose system complexity using the refinement mechanism of CSP. Hachicha *et al.*[27] proposed to formalize SAS systems with the Event-B model. This work makes a preliminary attempt, and establishes the structure model of SAS systems using the Event-B model. Different from the above work, the EasyModel approach integrates the intuitive UML model with the refinement Event-B model. It can not only handle system complexity with the refinement mechanism, but also realize an explicit description of self-adaptation characteristics (R1, R2, and R3).

### 7.3 Model Transformation Approaches for SAS

In order to incorporate merits of both UML and formal methods, some researchers proposed to integrate UML and formal methods for modeling SAS systems. Luckey and Engels[4] presented the ACML (Adapt Case Modeling Language) approach by integrating UML and LTS model for SAS systems. In [15], we proposed to integrate UML model with timed automata model, focusing on specifying self-adaptation behaviours and verifying self-adaptation properties. The limitations are that model checking of the LTS model and the timed automata model suffer from state-space explosion, and that the model transformation processes are manual or semi-automatic. In comparison, the EasyModel approach avoids the above problems by providing an integrated model verification method and an automatic model transformation tool.

## 8 Conclusions

Modeling and formal verification of SAS systems becomes increasingly difficult, as the system scale and complexity is rapidly increasing. To tackle this challenge, we presented EasyModel (Ease Modeling Difficulty), a refinement-based modeling and verification approach. EasyModel contributes in integrating the intuitive UML model with the stepwise refinement Event-B model. This integration can bridge the gap between the design model and the formal model of SAS systems, and alleviate modeling difficulty. Software engineers who are familiar with UML can quickly get started with EasyModel, as it requires less mathematical background. At the same time, EasyModel specifies complex self-adaptation activities from abstract models to concrete models, and this stepwise refinement mechanism not only decomposes system complexity but also conforms to the habit of human cognition, which paves the way for software engineers to use the approach in practice. Furthermore, EasyModel guides the formal verification of SAS systems by integrating PO discharging with model checking, providing an efficient way to

④https://lnu.se/en/research/searchresearch/adaptwise/, July 2020.

analyze and guarantee system behaviours. We evaluated EasyModel with an example application and a subject-based experiment. The results demonstrated that the EasyModel approach can satisfy requirements for modeling SAS systems, and can effectively reduce modeling and formal verification difficulty of SAS systems.

As for future work, we will continue the study on formal analysis of self-adaptation properties, simplifying the verification complexity and enriching the verifiable self-adaptation properties.

## References

[1] Salehie M, Tahvildari L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, 2009, 4(2): Article No. 14.

[2] de Lemos R, Garlan D, Ghezzi C *et al*. Software engineering for self-adaptive systems: Research challenges in the provision of assurances. In *Proc. the International Seminar on Software Engineering for Self-Adaptive Systems*, December 2017, pp.3-30.

[3] Weyns D. Software engineering of self-adaptive systems. In *Handbook of Software Engineering*, Cha S, Taylor R N, Kang K (eds.), Springer, 2019, pp.399-443.

[4] Luckey M, Engels G. High-quality specification of self-adaptive software systems. In *Proc. the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, May 2013, pp.143-152.

[5] Han D S, Yang Q L, Xing J C *et al*. FAME: A UML-based framework for modeling fuzzy self-adaptive software. *Information & Software Technology*, 2016, 76: 118-134.

[6] Said M B, Kacem Y H, Kerboeuf M *et al*. Design patterns for self-adaptive RTE systems specification. *International Journal of Reconfigurable Computing*, 2014, 2014: Article No. 536362.

[7] de la Iglesia D G, Weyns D. MAPE-K formal templates to rigorously design behaviors for self-adaptive systems. *ACM Transactions on Autonomous & Adaptive Systems*, 2015, 10(3): Article No. 15.

[8] Iftikhar M U, Weyns D. ActivFORMS: A runtime environment for architecture-based adaptation with guarantees. In *Proc. the IEEE Int. Conf. Software Architecture*, April 2017, pp.278-281.

[9] Camilli M, Gargantini A, Scandurra P. Zone-based formal specification and timing analysis of real-time self-adaptive systems. *Science of Computer Programming*, 2018, 159: 28-57.

[10] Ding Z, Zhou Y, Zhou M. Modeling self-adaptive software systems by fuzzy rules and Petri nets. *IEEE Transactions on Fuzzy Systems*, 2018, 26(2): 967-984.

[11] Almeida A, Bencomo N, Batista T *et al*. Dynamic decision-making based on NFR for managing software variability and configuration selection. In *Proc. the 30th Annual ACM Symp. Applied Computing*, April 2015, pp.1376-1382.

[12] Weyns D, Iftikhar M U. Model-based simulation at runtime for self-adaptive systems. In *Proc. the 2016 IEEE Int. Conf. Autonomic Computing*, July 2016, pp.364-373.

[13] Ahmad M, Belloir N, Bruel J M. Modeling and verification of functional and non-functional requirements of ambient self-adaptive systems. *Journal of Systems & Software*, 2015, 107: 50-70.

[14] Abrial J R. Modeling in Event-B: System and Software Engineering. Cambridge University Press, 2013.

[15] Han D, Yang Q, Xing J. UML-Based modeling and formal verification for software self-adaptation. *Journal of Software*, 2015, 26(4): 730-746. (in Chinese)

[16] Object Management Group, The OMG unified modeling language (OMG UML), superstructure, version 2.4.1, OMG Document No. formal/2011-08-06, 2011.

[17] Gérard S, Dumoulin C, Tessier P, Selic B. Papyrus: A UML2 tool for domain-specific language modeling. In *Proc. Int. Conf. Model-Based Engineering of Embedded Real-Time Systems*, November 2007, pp.361-368.

[18] Abrial J R, Butler M, Hallerstede S *et al*. Rodin an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer*, 2010, 12(6): 447-466.

[19] Leuschel M, Butler M. ProB: A model checker for B. In *Proc. International Symposium of Formal Methods Europe*, September 2003, pp.855-874.

[20] Mentré D, Marché C, Filliâtre J C *et al*. Discharging proof obligations from Atelier B using multiple automated provers. In *Proc. the 3rd Int. Conf. Abstract State Machines, Alloy, B, VDM, and Z*, June 2012, pp.238-251.

[21] Boniol F, Wiels V, Ameur Y A, Schewe K D. The landing gear case study: Challenges and experiments. *International Journal on Software Tools for Technology Transfer*, 2017, 19(2): 133-140.

[22] Su W, Abrial J R. Aircraft landing gear system: Approaches with Event-B to the modeling of an industrial system. *International Journal on Software Tools for Technology Transfer*, 2017, 19(2): 141-166.

[23] Laibinis L, Klionskiy D, Troubitsyna E *et al*. Modelling resilience of data processing capabilities of CPS. In *Proc. the 6th International Workshop on Software Engineering for Resilient Systems*, October 2014, pp.55-70.

[24] Cheng S W. Rainbow: Cost-effective software architecture-based self-adaptation [Ph.D. Thesis]. Carnegie Mellon University, 2008.

[25] Kephart J O, Chess D M. The vision of autonomic computing. *IEEE Computer*, 2003, 36(1): 41-50.

[26] Vandome A F, Mcbrewster J, Miller F P. ATLAS Transformation Language. Alphascript Publishing, 2010.

[27] Hachicha M, Dammak E, Halima R B *et al*. A correct by construction approach for modeling and formalizing self-adaptive systems. In *Proc. the 17th IEEE/ACIS Int. Conf. Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, June 2016, pp.379-384.

[28] Vogel T. Model-driven engineering of self-adaptive software [Ph.D. Thesis]. University of Potsdam, 2018.

[29] Vogel T, Giese H. Model-driven engineering of self-adaptive software with EUREMA. *ACM Transactions on Autonomous & Adaptive Systems*, 2014, 8(4): Article No. 18.

[30] Cámara J, Lopes A, Garlan D *et al*. Adaptation impact and environment models for architecture-based self-adaptive systems. *Science of Computer Programming*, 2016, 127: 50-75.

[31] Han D, Xing J, Yang Q *et al.* Modeling and verification approach for temporal properties of self-adaptive software dynamic processes. *Journal of Computer Applications*, 2018, 38(3): 799-805. (in Chinese)

[32] Krupitzer C, Roth F M, VanSyckel S *et al.* A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 2015, 17: 184-206.

[33] Abuseta Y, Swesi K. Design patterns for self-adaptive systems engineering. *International Journal of Software Engineering & Applications*, 2015, 6(4): 11-28.

[34] Becker S, Dziwok S, Gerking C *et al.* The MechatronicUML method: Model-driven software engineering of self-adaptive mechatronic systems. In *Proc. the 36th Int. Conf. Software Engineering*, May 2014, pp.614-615.

[35] Heinzemann C, Becker S, Volk A. Transactional execution of hierarchical reconfigurations in cyber-physical systems. *Software & Systems Modeling*, 2019, 18(1): 157-189.

[36] da Silva J P S, Ecar M, Pimenta M S *et al.* A systematic literature review of UML-based domain specific modeling languages for self-adaptive systems. In *Proc. the 13th IEEE/ACM Int. Symp. Software Engineering for Adaptive and Self-Managing Systems*, May 2018, pp.87-93.

[37] Moritani B I, Lee J. An approach for managing a distributed feature model to evolve self-adaptive dynamic software product lines. In *Proc. the 21st International Systems and Software Product Line Conference*, September 2017, pp.107-110.

[38] Chen T, Li K, Bahsoon R *et al.* FEMOSAA: Feature-guided and knee-driven multi-objective optimization for self-adaptive software. *ACM Transactions on Software Engineering and Methodology*, 2018, 27(2): Article No. 5.

[39] de la Iglesia D G, Calderón J F, Weyns D *et al.* A self-adaptive multi-agent system approach for collaborative mobile learning. *IEEE Transactions on Learning Technologies*, 2015, 8(2): 158-172.

[40] Wang L, Li Q. A multiagent-based framework for self-adaptive software with search-based optimization. In *Proc. IEEE Int. Conf. Software Maintenance and Evolution*, October 2016, pp.621-625.

[41] Iftikhar M U, Weyns D. ActivFORMS: A runtime environment for architecture-based adaptation with guarantees. In *Proc. Int. Conf. Software Architecture Workshops*, April 2017, pp.278-281.

[42] Abbas N, Andersson J, Iftikhar U M *et al.* Rigorous architectural reasoning for self-adaptive software systems. In *Proc. the 1st Workshop on Qualitative Reasoning about Software Architectures*, April 2016, pp.11-18.

[43] Abbas N, Andersson J. Architectural reasoning support for product-lines of self-adaptive software systems — A case study. In *Proc. the 9th European Conference on Software Architecture*, September 2015, pp.20-36.

[44] Su G, Chen T, Feng Y *et al.* An iterative decision-making scheme for Markov decision processes and its application to self-adaptive systems. In *Proc. the 19th Int. Conf. Fundamental Approaches to Software Engineering*, April 2016, pp.269-286.

[45] Filieri A, Tamburrelli G, Ghezzi C. Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. *IEEE Transactions on Software Engineering*, 2016, 42(1): 75-99.

[46] Göthel T, Jähnig N, Seif S. Refinement-based modelling and verification of design patterns for self-adaptive systems. In *Proc. the 19th Int. Conf. Formal Engineering Methods*, November 2017. pp.157-173.

**De-Shuai Han** received his B.S. degree in electrical engineering and automation from Shandong University, Jinan, in 2012, his M.S. degree in electric system and automation from PLA University of Science and Technology, Nanjing, in 2015, and his Ph.D. degree in information system engineering from Army Engineering University of PLA, Nanjing, in 2018. He is currently a lecturer in Rocket Force University of Engineering, Xi'an. His research interests include self-adaptive software systems, mission-critical system and software, pervasive computing, and cyber-physical systems.

**Qi-Liang Yang** received his Ph.D. degree in computer science from Nanjing University, Nanjing. He is currently an associate professor in Army Engineering University of PLA, Nanjing. His research interests include self-adaptive software systems, mission-critical system and software, pervasive computing, and cyber-physical systems.

**Jian-Chun Xing** received his B.Sc. and M.Sc. degrees in electric system and automation from Engineering Institute of Engineering Corps, Nanjing, in 1984 and 1987, respectively, and his Ph.D. degree in information system engineering from Army Engineering University of PLA, Nanjing, in 2006. He is currently a professor in Army Engineering University of PLA. His research interests include intelligent control, artificial intelligence and information processing.

**Guang-Lian Ma** received her M.Sc. degree in detection technology and automatic equipment from Hohai University, Nanjing, in 2016. She is currently an assistant engineer in Rocket Force University of Engineering, Xi'an. Her research interests include software testing, information processing, and Internet of Things.

## Appendix

### A.1 Proof Obligations for Sequential Control Node

In order to derive the proof obligations, we employ the "forward simulation" refinement mechanism [14], which defines a sufficient refinement condition. According to the "forward simulation", the abstract event *abs* and concrete events *con*1 and *con*2 can be represented in Fig. A1. The concrete events must be a trace of the abstract event,

$$r^{-1};\ \ (re_m \rightarrow re_n) \subseteq ae_i; r^{-1}. \qquad \text{(A1)}$$

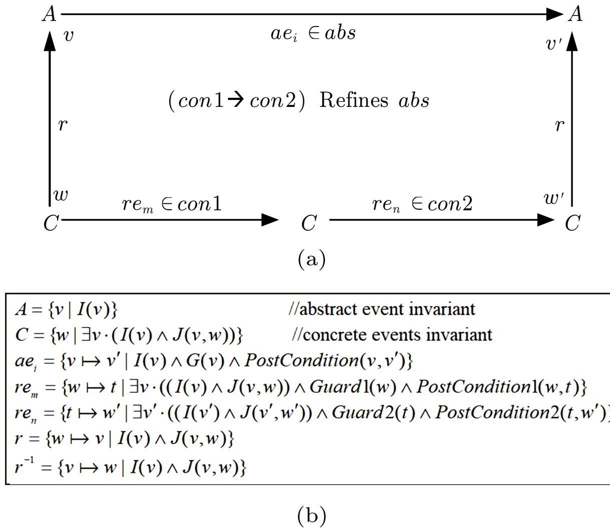



Fig. A1. Formal representation of the Event-B sequential node. (a) Traces of sequential refinements. (b) Set theoretic representation of the Event-B models.

Among them, the sequential events can be expressed as

$$\begin{aligned} & re_m \rightarrow re_n \\ = & \{w \mapsto w' | \exists v \times (I(v) \wedge J(v,w) \wedge \\ & \exists t \times \wedge Guard1(w) \wedge PostCondition1(w,t) \wedge \\ & Guard2(t) \wedge PostCondition2(t,w')\}. \end{aligned}$$

According to the "forward formulation", the traces of the concrete event and the abstract event can be expressed as (A2) and (A3), respectively.

$$\begin{aligned} & r^{-1}; (re_m \rightarrow re_n) \\ = & \{v \mapsto w' | \exists w \times (I(v) \wedge J(v,w) \wedge \exists t \times \wedge Guard1(w) \wedge \\ & PostCondition1(w,t) \wedge Guard2(t) \wedge \\ & PostCondition2(t,w')\}, \end{aligned} \qquad \text{(A2)}$$

$$\begin{aligned} & ae_i; r^{-1} \\ = & \{v \mapsto w' | \exists v' \times I(v) \wedge Guard(v) \wedge \\ & PostCondition(v,v') \wedge I(v') \wedge J(v',w')\}. \end{aligned} \qquad \text{(A3)}$$

Derived from the "forward simulation" condition of (A1), (A2) and (A3), we can achieve (A4),

$$\begin{aligned} & I(v) \wedge J(v,w) \wedge \exists t \times \wedge Guard1(w) \wedge \\ & PostCondition1(w,t) \wedge Guard2(t) \wedge \\ & PostCondition2(t,w') \vdash I(v) \wedge Guard(v) \wedge \\ & \exists v' \times PostCondition(v,v') \wedge \\ & I(v') \wedge J(v',w'). \end{aligned} \qquad \text{(A4)}$$

According to the definition of sequential control node, let us make the following assumption:

$$PostCondition1 \Rightarrow Guard2. \quad \text{(PO1)}$$

On basis of PO1, (A4) can be simplified as:

$$\begin{aligned} & I(v) \wedge J(v,w) \wedge Guard1(w) \wedge PostCondition1(w,t) \wedge \\ & PostCondition2(t,w') \vdash Guard(v) \wedge \\ & \exists v' \times PostCondition(v,v') \wedge I(v') \wedge J(v',w'). \end{aligned} \quad \text{(A5)}$$

By applying the inference rule AND-R [14], (A5) can be simplified as (A6) and (A7),

$$\begin{aligned} & (I(v) \wedge J(v,w) \wedge Guard1(w) \wedge PostCondition1(w,t) \wedge \\ & PostCondition2(t,w') \vdash Guard(v). \end{aligned} \qquad \text{(A6)}$$

By applying the inference rule of Monotonicity [14], PO2 is achieved:

$$Guard1 \Rightarrow Guard. \quad \text{(PO2)}$$

$$\begin{aligned} & (I(v) \wedge J(v,w) \wedge Guard1(w) \wedge PostCondition1(w,t) \wedge \\ & PostCondition2(t,w') \vdash \exists v' \times PostCondition(v,v') \wedge \\ & I(v') \wedge J(v',w'). \end{aligned} \qquad \text{(A7)}$$

Similarly, by applying the inference rule of Monotonicity [14], PO3 is achieved,

$$PostCondition2 \Rightarrow PostCondition. \quad \text{(PO3)}$$

Besides, in order to guarantee that event con2 can only be triggered by event con1, the following constraint is necessary,

$$init(Guard2) = \text{FALSE}, \quad \text{(C1)}$$

which means that Guard2 should be set to be FALSE in the initialization event.

Therefore, PO1, PO2 and PO3 are proof obligations of the sequential node, and C1 is the constraint.

### A.2 Proof Obligations for Branch Control Node

Here the branch node means that exactly one refined concrete event can be triggered at a time; thus, it is an XOR (exclusive or) relationship between the refined concrete events, that is,

$$(con1 \text{ XOR } con2) \text{ Refines } abs.$$

To simplify the derivation and proving process, we decompose this XOR relationship into OR and Exclusive relationships.

$$(con1 \text{ OR } con2) \text{ Refines } abs, \quad \text{(A8)}$$

$$(con1 \text{ Exclusive } con2) \text{ Refines } abs. \quad \text{(A9)}$$

Initially, we only consider the OR refinement pattern in (A8), which is similar to that of the sequential patterns. Similarly, the traces and set theoretic representation can be shown in Fig. A2.

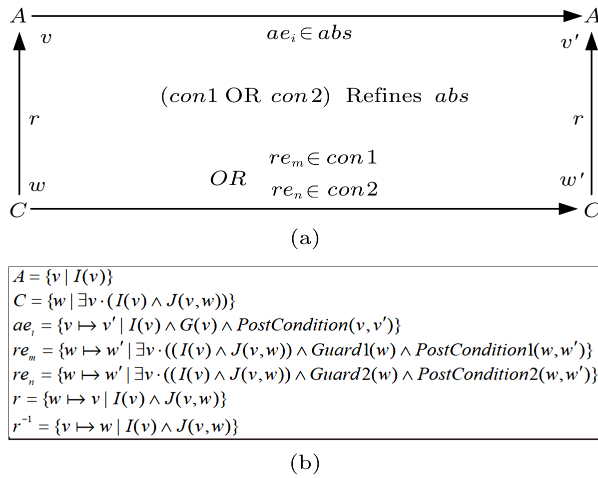



Fig. A2. Formal representation of the Event-B branch node. (a) Traces of branch refinements. (b) Set theoretic representation of the Event-B models.

According to the "forward simulation", the abstract event $abs$ and concrete events $con1$ and $con2$ can be represented as Fig. A2. The concrete events must be a trace of the abstract event,

$$r^{-1}; \ (re_m \text{ OR } re_n) \subseteq ae_i; r^{-1}. \quad \text{(A10)}$$

Among them, the concrete branch events can be expressed as,

$$re_m \text{ OR } re_n = \{w \mapsto w' | \exists v \times (I(v) \wedge J(v,w) \wedge (Guard1(w) \wedge Guard2(w)) \wedge ((Guard1(w) \Rightarrow PostCondition1(w,w')) \vee (Guard2(w) \Rightarrow PostCondition2(w,w')))\}.$$

According to the "forward formulation", the traces of the concrete event and the abstract event can be expressed as (A11) and (A12), respectively.

$$r^{-1}; (re_m \text{ OR } re_n) = \{v \mapsto w' | \exists w \times (I(v) \wedge J(v,w) \wedge (Guard1(w) \wedge Guard2(w)) \wedge ((Guard1(w) \Rightarrow PostCondition1(w,w')) \vee (Guard2(w) \Rightarrow PostCondition2(w,w')))\}, \quad \text{(A11)}$$

$$ae_i; \ r^{-1} = \{v \mapsto w' | \exists v' \times I(v) \wedge Guard(v) \wedge PostCondition(v,v') \wedge I(v') \wedge J(v',w')\}. \quad \text{(A12)}$$

Derived from the "forward simulation" condition of (A10), (A11) and (A12), we can achieve (A13),

$$(I(v) \wedge J(v,w) \wedge (Guard1(w) \wedge Guard2(w)) \wedge ((Guard1(w) \Rightarrow PostCondition1(w,w')) \vee (Guard2(w) \Rightarrow PostCondition2(w,w'))) \vdash I(v) \wedge Guard(v) \wedge \exists v' \times PostCondition(v,v') \wedge I(v') \wedge J(v',w'). \quad \text{(A13)}$$

Applying the inference rules: OR-L [14], IMP-L [14] and MON [14] we can simplify (A13), and get the following sequents:

$$(I(v) \wedge J(v,w) \wedge Guard1(w) \wedge PostCondition1(w,w') \vdash I(v) \wedge Guard(v) \wedge \exists v' \times PostCondition(v,v') \wedge I(v') \wedge J(v',w'), \quad \text{(A14)}$$

$$(I(v) \wedge J(v,w) \wedge Guard2(w) \wedge PostCondition2(w,w') \vdash I(v) \wedge Guard(v) \wedge \exists v' \times PostCondition(v,v') \wedge I(v') \wedge J(v',w'). \quad \text{(A15)}$$

Firstly, we need two sufficient POs to guarantee (A14):

$$Guard1(w) \Rightarrow Guard(v), \text{ and}$$
$$PostCondition1 \Rightarrow PostCondition.$$

In Fig.5(a), it is obvious that $Guard1(w) \Rightarrow Guard(v)$ can be satisfied, and we only need the latter PO, that is,

$$PostCondition1 \Rightarrow PostCondition. \quad \text{(PO1)}$$

Similarly, according to (A15), we can get PO2,

$$PostCondition2 \Rightarrow PostCondition. \quad \text{(PO2)}$$

Then, back to (A8) and (A9), PO1 and PO2 can guarantee the OR relationship, and we need other POs to guarantee the exclusive relationship, i.e.,

$$PostCondition1 \Rightarrow \neg Guard. \quad \text{(PO3)}$$

$$PostCondition2 \Rightarrow \neg Guard. \quad \text{(PO4)}$$

At last, in order to guarantee that the abstract event can only be triggered once, we need another PO,

$$PostCondition \Rightarrow \neg Guard. \quad \text{(PO5)}$$

### A.3 Proof Obligations for Cooperation Control Node

Similarly, according to the "forward simulation", the abstract event *abs* and concrete events *con*1 and *con*2 can be represented in Fig. A3. And the concrete events must be a trace of the abstract event,

$$r^{-1}; (re_m || re_n) \subseteq ae_i; r^{-1}. \quad \text{(A16)}$$

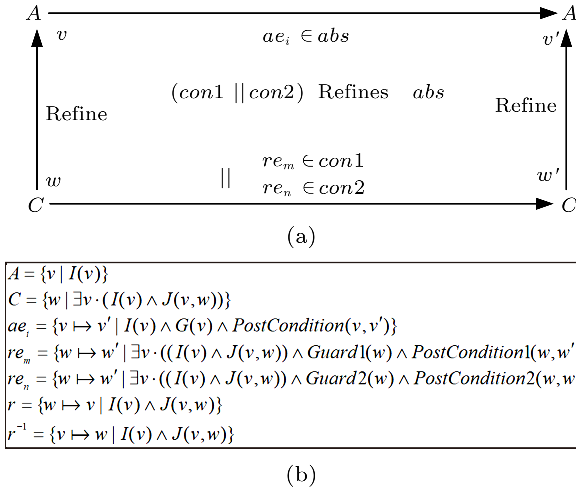



Fig. A3. Formal representation of the Event-B cooperation node. (a) Traces of cooperation refinements. (b) Set theoretic representation of the Event-B models.

Among them, the cooperation events can be expressed as,

$$\begin{aligned} re_m || re_n = \{ & w \mapsto w' | \exists v \times (I(v) \wedge J(v,w) \wedge \\ & (Guard1(w) \wedge Guard2(w)) \wedge \\ & ((Guard1(w) \wedge Guard2(w)) \\ \Rightarrow & (PostCondition1(w,w') \wedge \\ & (PostCondition2(w,w'))\}. \end{aligned}$$

According to the "forward formulation", the traces of the concrete event and the abstract event can be expressed as (A17) and (A18), respectively.

$$\begin{aligned} & r^{-1}; (re_m || re_n) = \{v \mapsto w' | \exists w \times (I(v) \wedge \\ & J(v,w) \wedge (Guard1(w) \wedge Guard2(w)) \wedge \\ & ((Guard1(w) \wedge Guard2(w)) \\ \Rightarrow & (PostCondition1(w,w') \wedge \\ & (PostCondition2(w,w'))\}, \end{aligned} \quad \text{(A17)}$$

$$\begin{aligned} & ae_i; r^{-1} \\ = & \{v \mapsto w' | \exists v' \times I(v) \wedge Guard(v) \wedge \\ & PostCondition(v,v') \wedge \\ & I(v') \wedge J(v',w')\}. \end{aligned} \quad \text{(A18)}$$

According to (A16), (A17) and (A18), we can achieve (A19),

$$\begin{aligned} & (I(v) \wedge J(v,w) \wedge (Guard1(w) \wedge Guard2(w)) \wedge \\ & ((Guard1(w) \wedge Guard2(w))) \\ \Rightarrow & \{PostCondition1(w,w') \wedge \\ & (PostCondition2(w,w'))\} \vdash \\ & I(v) \wedge Guard(v) \wedge \exists v' \times PostCondition(v,v') \wedge \\ & I(v') \wedge J(v',w'). \end{aligned} \quad \text{(A19)}$$

With the inference rule of IMP-L [14], the above sequent can be simplified as,

$$\begin{aligned} & I(v) \wedge J(v,w) \wedge Guard1(w) \wedge Guard2(w) \wedge \\ & PostCondition1(w,w') \wedge PostCondition2(w,w') \vdash \\ & I(v) \wedge Guard(v) \wedge \exists v' \times PostCondition(v,v') \wedge \\ & I(v') \wedge J(v',w'). \end{aligned} \quad \text{(A20)}$$

It needs three sufficient POs to guarantee (A20),

$$Guard1 \Rightarrow Guard. \quad \text{(PO1)}$$

$$Guard2 \Rightarrow Guard. \quad \text{(PO2)}$$

$$\begin{aligned} & PostCondition1 \wedge PostCondition2 \\ \Rightarrow & PostCondition. \quad \text{(PO3)} \end{aligned}$$