Journal of Systems Engineering and Electronics
Vol. 32, No. 4, August 2021, pp.889 – 906

An executable framework for modeling and validating cooperative capability requirements in emergency response system

CHAI Lei, WANG Zhixue, HE Ming*, HE Hongyue, and YU Minggang
Institute of Command Control Engineering, Army Engineering University of PLA, Nanjing 210007, China

Abstract: As the scale of current systems become larger and larger and their complexity is increasing gradually, research on executable models in the design phase becomes significantly important as it is helpful to simulate the execution process and capture defects of a system in advance. Meanwhile, the capability of a system becomes so important that stakeholders tend to emphasize their capability requirements when developing a system. To deal with the lack of official specifications and the fundamental theory basis for capability requirement, we propose a cooperative capability requirements (CCR) meta-model as a theory basis for researchers to refer to in this research domain, in which we provide detailed definition of the CCR concepts, associations and rules. Moreover, we also propose an executable framework, which may enable modelers to simulate the execution process of a system in advance and do well in filling the inconsistency and semantic gaps between stakeholders' requirements and their models. The primary working mechanism of the framework is to transform the Alf activity meta-model into the communicating sequential process (CSP) process meta-model based on some mapping rules, after which the internal communication mechanism between process nodes is designed to smooth the execution of behaviors in a CSP system. Moreover, a validation method is utilized to check the correctness and consistency of the models, and a self-fixing mechanism is used to fix the errors and warnings captured during the validation process automatically. Finally, a validation report is generated and fed back to the modelers for system optimization.

Keywords: executable model, capability requirement, consistency validation, Alf, epsilon.

DOI: 10.23919/JSEE.2021.000077

1. Introduction

With the explosive development of social diversification, interactions and cooperation between different domains tend to be more and more frequent, as a result, the scale of a software system supporting these interactions and cooperation become larger and larger and its complexity is increasing gradually. When developing large scale complex systems, especially for those applied into some vital application domains, e.g., in emergency response system (ERS), in military domain, whose reliability, efficiency and security are main concerns and it is very difficult to evaluate their capabilities and identify their bugs and defects. For example, if an ERS is developed for carrying out emergency rescue missions after earthquakes, only an earthquake actually occurs can we evaluate the capability of this ERS and capture its defects and errors, so it is unwise to capture some typical errors (such as inconsistencies, unreachable processes and deadlocks) until the testing phase and return the system back for redevelopment. Consequently, an executable framework in the requirement analysis phase, which is helpful to simulate the execution process and capture the defects of a system in advance, is considered as a significantly important approach to evaluate the capability of the to-be system, for it is beneficial for requirement analysts or modelers to fix these defects reasonably and effectively in the requirement analysis phase or the modeling design phase so as to ensure the to-be ERS is rational and scientific. Take the 2020 novel corona virus for example, which has affected many countries and caused huge loss of life and property, it is mainly because that many countries fail to evaluate the capability of their bio-safety emergency response plans ahead of time. If these countries has constructed executable models when making their bio-safety emergency response plans, they would be able to simulate the execution behaviors of their emergency response activities beforehand, then repair defects captured during the simulation process and finally improve the capability of their bio-safety emergency response. If they had done so, maybe there would not be such a huge loss.

Nowadays, when developing systems, especially large scale and complex systems, the capability of these to-be systems has drawn increasingly more stakeholders' atten-

Manuscript received September 01, 2020.
*Corresponding author.
This work was supported by the National Key R&D Program of China (2018YFC0806900), the China Postdoctoral Science Foundation Funded Project (2018M633757), the Primary Research & Development Plan of Jiangsu Province (BE2016904; BE2017616; BE2018754; BE2019762), and Jiangsu Province Postdoctoral Science Foundation Funded Project (2019K185).

tion, which will visually represent some vital features (functions and performances) of the systems, and stakeholders can evaluate the real value of the systems directly. Therefore, when developing a large and complex system, stakeholders tend to emphasize their capability requirements to modelers and developers of the to-be system. However, the “Capability” concept firstly comes from some military architecture frameworks, such as the Department of Defense Architecture Framework (DoDAF [1]) and the Ministry of Defense Architecture Framework (MODAF), most of which consider that capability is a significantly important element to represent the value of a system. The term “Capability Requirement” frequently appears in various official documents and research papers, during which there is no precise official definition for it, our research group refers to it as the requirements of functions and performances of a large and complex system [2]. In addition, there are two principal categories of capability: complex capability and simple capability. According to the capability decomposition characteristic [1], a complex capability usually can be decomposed into other several simple capabilities, and we use the term “atomic capability” to represent the smallest capability unit so that it cannot be decomposed into other capabilities again.

At present, as the frequency and the severity of the disasters around the world increase, there is an urgent requirement for more efficient emergency rescue ways to respond to small or large scale emergent incidents. Such a requirement gradually forms a trend that cohesive cooperation among all performers is increasingly required in ERS, especially in vital cases of crises and disasters. In an ERS, only multiple performers cooperate with each other smoothly and efficiently, can we give full play to the maximum effectiveness of the cooperative capability of ERS, which is beneficial to make up for the deficiency of a single atomic capability and form capability support as an organic whole in ERS. However, the evaluation of the cooperative capability of a whole ERS is not a simple superposition or integration of atomic capabilities, because there are complex relationships between atomic capabilities existing in ERS, actually, it involves an organic composition of multi atomic capabilities under several composition rules [3].

Therefore, in order to evaluate the capability of an ERS, the capability of cooperation or interaction between multiple performers should be fully considered. Consequently, stakeholders tend to be accustomed to emphasize their cooperative capability requirements (CCR) to modelers and developers when developing an ERS. However, when studying the CCR, what elements (or concepts) researchers should take into consideration? How to define the associations between elements (or concepts) and what does the meta-model of CCR look like? There is no official specification for them. In this study, we design a CCR meta-model in the emergency response domain, in which the definitions of CCR concepts, some essential elements of CCR meta-model and the associations between elements are included. Moreover, our research group wishes that the CCR meta-model we propose could help to deal with the lack of official specifications about CCR at present and provide some references for researchers in this domain.

As the issue of executable models has already drawn much attention of researchers and much study also have been done in this research domain [4−6], even though the models modelers have designed can correctly represent the requirements of stakeholders, the executable models cannot be executed directly. Review paper [7] introduces that there are two execution ways to make models executable, interpretive execution and translational execution. For interpretive execution, there should be some compilers that can parse the models based on the modeling language and execute the models directly, and during the very execution process, there will not be such problems as inconsistency and semantic gaps; while for translational execution, modelers should translate the modeling language into a specific high-level programming language (such as Java and C++) and execute on its target platform, during which process the problem of inconsistency and semantic gaps will arise again. However, the current situation for this execution issue is that few compilers are equipped for most modeling languages to parse models and execute them directly, and modelers usually tend to adopt to translate the modeling language into a high-level programming language to make their models executable.

Generally speaking, after stakeholders propose their cooperative capability requirements, modelers will abstract the requirements and design models. Since stakeholders’ requirements are usually textual descriptions expressed by human-readable nature languages, there will be several semantic or inconsistency gaps between the modeling language and the natural language [4]. During various modeling languages in the modeling domain at present, the unified modeling language (UML) [8], released by the Object Management Group (OMG), is considered as a popular and practical modeling standard in actual industry modeling practice [9], and especially is widely used to describe the architecture of some large and complex systems in the specific domain [10]. In spite of this, UML can only depict the most static properties and fail to depict the dynamic behavior of the system, which makes UML models unable to execute directly. To make up for this deficiency of the UML, the OMG releases fUML [11] and Alf [12] to provide precise definition of behavior semantics for UML models, which enable UML to depict dynamic behaviors of systems, therefore, application models modeled with fUML or Alf are executable theoretically. However, the behaviors semantics fUML can provide is limited to a subset of UML (such as composite structures, activities and classes), while Alf does not need to conform to the scope limitations; moreover, compared to the graphical notation features of fUML modeling elements, the textual notation features of Alf modeling elements enable Alf models more suitable to be interpreted and executed on a target platform, which is significantly helpful to eliminate (or alleviate) the semantic and consistency gaps between models transformation to a large extent. In this paper, we choose Alf to model the CCR of ERS, and translate Alf models into communicating sequential processes (CSP) models to make them executable.

The reasons we choose CSP as the target language are that: firstly, CSP is good at describing the execution sequence and internal event triggering mechanism of a system in detail, it can provide sufficiently powerful syntax and semantics to simulate the execution processes of cooperative activities in an ERS [13,14]. Secondly, as a rigorous mathematical method, CSP utilizes algebraic laws for processes specification and formal logic reasoning to analyze the behaviors of a system [15,16]. During the models transformation from translating Alf into CSP, based on CSP powerful algebra theoretical laws, we are enabled to examine whether there are inconsistency and semantic gaps between models transformation.

In this paper, we propose an executable framework for modeling and validating CCR in ERS, in which a formal description of CCR in ERS, model transformation for simulating the execution process of cooperative activities in ERS and models validation are included. The framework is helpful to evaluate the cooperative capability of an ERS in advance, since it is able to simulate the execution process of various cooperative activities (behaviors) in ERS automatically at the requirement analysis phase.

In summary, the contributions of this study are provided as follows:

(i) Propose a CCR meta-model in the emergency response domain, in which we try our best to define CCR concepts, internal associations between elements in the model and meta-rules for constructing the cooperative capability.

(ii) Propose an executable framework of CCR in ERS, which is significantly helpful to simulate the execution process of cooperative activities in ERS and ensure the consistency between the executable models and the stakeholders' requirements.

(iii) Utilize the Alf language, for its extensive and comprehensive executable semantics, as a semi-formal modeling language to design CCR models in ERS.

(iv) Implement the detailed transformation of transforming Alf models into CSP models by using the Epsilon language as a middleware. As various cooperative behaviors (sequential, parallel, conditional and iterative) exist in ERS at present, CSP does well in capturing these behaviors for its high-level descriptions of synchronizations, communications, and interactions between process nodes.

(v) Provide consistency validation between models and stakeholders' requirements in order to examine that if these models satisfy their requirements and if some constraint rules (logical rules and business rules) are broken during the modeling process. And finally generate a validation report, which records the errors and warnings captured during the validation process, and feed it back to modelers for optimizing the executable framework.

The rest of this study is organized as follows: in Section 2 we give a brief background introduction of the tools we select to conduct our study; in Section 3 we introduce the meta-model of CCR we design, which provides detailed definition of CCR of meta-concepts, meta-association and meta-rule involved; in Section 4 we give an overview of the executable framework of CCR in ERS, which will assist us to illustrate the working mechanism of the framework; in Section 5, we describe the process of model transformation; in Section 6, we introduce the communication mechanism between process nodes briefly and finally, a case study is led to illustrate the feasibility and effectiveness of the method in Section 7.

2. Related work

In this section, we introduce the tools (the modeling language, the formal method and the integrated development environment platform) we select to facilitate our study, during which process we will illustrate what advantage the tools have compared to the current popular tools in their application domain.

2.1 Alf

Although both fUML [11] and Alf are released by the OMG to provide precise definition of behavior semantics for UML models, each of which has their own features. Compared to their advantages and disadvantages, our research group considers that in this research paper Alf is more suitable than fUML.

As a subset of UML, fUML is able to provide precise definition of behavior semantics by adding some abstract syntax elements (constraint structure, behaviors, activities and actions) to UML models. Consequently, these features enable fUML to describe the dynamic behaviors of a system, as a result, application models modeled with fUML are executable theoretically, as long as there is a complier which could parse fUML models directly, which does not exist actually at present. Therefore, modelers and researchers tend to translate the fUML models to a format file expressed by a high-level programming language so as to make the file run on a specific target platform (e.g., translate the fUML models to Java code and make the models run on a Java virtual machine)[7]. Although the fUML models can be executed, another problem may arise that there may be semantic and inconsistency gaps between models transformation.

Similar to fUML, Alf [12] is released by the OMG to provide precise execution semantics to UML models, but fUML is limited to a subset of UML to provide execution semantics to composite structures, activities and classes, while Alf is utilized primarily for providing execution semantics for UML models and does not need to conform to the scope limitation. Alf is a textual surface notation for UML model elements, which implements its execution semantic by mapping Alf concrete syntax to UML abstract syntax. Additionally, not only can Alf specify the execution behaviors, its extended notations also can be used for representing structural modeling elements [7]. That is to say, modelers and researchers are able to model a system and describe its behaviors entirely using a unified modeling language Alf, which enables Alf models to be easily interpreted and executed on a target platform and is helpful to eliminate or alleviate the semantic and consistency gaps between models transformation to a large extent.

In this study, Alf is utilized to model the activities (behaviors) of an ERS, for the features of its textual notation make Alf models more easily interpreted and executed on a target platform than fUML.

2.2 CSP

For popular formal methods of architecture frameworks, most researchers tend to adopt Petri net (PN), finite state machine (FSM) and CSP [13].

Among these methods, an FSM model can describe the state transitions and business logic of a system in detail, but the scope it can describe is very limited and a key disadvantage is that there is a lack of descriptions for some relevant dynamic semantics of activities and behaviors. That is to say, FSM just can describe some static properties of a system, not including the dynamic behaviors of the system.

Better than FSM, a PN model can capture and describe the dynamic state of a system, but PN is only suitable for modeling small and medium scale systems. When facing large-scale complex systems, PN fails to control the large number of emerging behaviors of its models in state transition and event triggering, which may easily cause the state spaces explosion problems in real application.

CSP [14], as a branch of process algebra, is different from classic FSM and PN methods and it can describe systems without considering the scale of systems. When describing a system, the CSP models will generate Markov transition processes whose unique equivalent combination technology can effectively compress the size of the first-order state space of the continuous time Markov chain, whose technology can improve the state space explosion problem to a large extent.

Moreover, as a rigorous mathematical method, CSP utilizes algebraic rules for processes specification and formal logic reasoning to analyze the behavior of a system [15,16], which is helpful to examine whether inconsistency and semantic gaps arise between models transformation. In addition, not only can CSP capture the dynamic state of a system, but also can describe concurrent systems, which is utilized to reason about systems biology, communication protocols, and business logic [17]. Our research subject, the cooperative capability in ERS, includes many concurrent behaviors (such as parallel executions) because there are interactions between multiple subsystems. Furthermore, CSP provide a family of some relevant classic stochastic process algebras, such as timed process and performance (TIPP) evaluation and performance evaluation process algebra (PEPA) [18], which have perfect formal semantic definition languages for researchers to select to satisfy their research needs.

In this study, we aim at forming several CSP formal executable models, based on its powerful algebra theoretical laws [19], which are sufficiently enough to simulate the execution process of an ERS. To obtain CSP executable models, we will transform Alf models into corresponding CSP models by using the model operation language Epsilon.

2.3 Epsilon

Epsilon is a novel open source programming language which is designed mainly for model management tasks such as code generation, model-to-model transformation, models validation, comparison, merging and refactoring [20]. Epsilon provides a set of eclipse-based development tools and an interpreter which can execute programs written in this language, as well as several ANT workflows of different tasks (e.g., a validation followed by a transformation after code generation).

Epsilon provides a family of languages to fulfill the specific model management task as follows: epsilon object language (EOL), epsilon transformation language

(ETL), epsilon validation language (EVL), epsilon generation language (EGL), epsilon comparison language (ECL), epsilon model generation language (EMG), epsilon merging language (EML), epsilon pattern language (EPL) and epsilon wizard language (EWL). From the features of these specific-task languages, we may draw a conclusion that the significant distinguish between epsilon and a high-level programming language is that the epsilon language manages and operates a model directly as long as the model has already been registered into its model library. Generally speaking, epsilon is a model-oriented language; the features of epsilon may help to avoid (at least alleviate) the semantic and inconsistency gaps between models transformation and execute behaviors of the model on the basis that the model has been correctly designed. Now we introduce two specific-task epsilon language briefly, ETL and EVL, which are relevant to our study [21].

ETL is a hybrid, rule-based model-to-model transformation language which provides all the standard features and enhanced flexibility of a transformation language. Consequently, it can transform an arbitrary number of input models to an arbitrary number of output models, and modify and update both source and target models once the models are changed. In this study, we will adopt ETL to transform Alf models into CSP models.

EVL is a validation language that supports generating customizable validation report, which records error and warning messages during model transformation, and feeding the validation report back to modelers for system optimization. Moreover, EVL provides a self-fixing mechanism which modelers can make full use of to repair inconsistencies and semantic gaps automatically. In this study, EVL is utilized to validate the CSP models transformed from Alf models and generate validation report, as well as specify quick fixes for errors and warnings.

3. The CCR meta-model

3.1 Definition of CCR meta-model

Since the "Capability" concept roots from the Data Meta-Model Group (DM2 Group) of DoDAF [1], according to the previous results [2,3] of our research group, in the specific emergency response domain, according to our research need, we propose a CCR meta-model as a theoretical basis for cooperative capability modeling in this research field, by extracting some related meta-concepts and associations from the DM2 Group. Fig. 1 shows the CCR meta-model we design, and we also construct the internal relationships, as well as some basic constraint rules of the cooperative capability, among meta-concepts according to their attributes and properties.

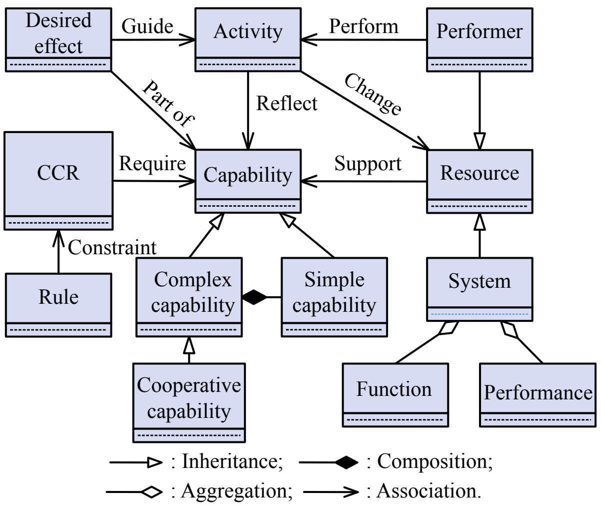


Fig. 1 CCR meta-model in emergency response domain

Definition 1 The CCR meta-model is a formal framework to detail the emergency response domain concepts at an architectural level, which is composed of three parts: meta-concept, meta-association and meta-rule.

Meta-concept is an extensible finite set of fundamental concepts which represent some essential elements related to CCR extracted from the DM2 Group, such as activity, capability, resource, performer, and rule, and the definitions of these meta-concepts conform to those in the DM2 Group, readers can refer to [8] for detailed introduction.

Meta-association is an extensible finite set of fundamental associations among all the meta-concepts. The meta-associations of CCR meta-model can be summarized as follows:

(i) CCR is a requirement description that requires several capabilities to composite a cooperative capability under the constraints of several rules;

(ii) Capability has two principle categories: complex capability and simple capability;

(iii) A complex capability is composited by several simple capabilities (according to the capability decomposition characteristic) under several composition rules, and a complex capability can be utilized as a simple capability to composite other complex capabilities;

(iv) The cooperative capability is a kind of complex capability, which should be composited by several simple capabilities under several composition rules;

(v) The evaluation of a capability is reflected by the execution effect of several activities;

(vi) Activity should be performed by performers;

(vii) Activity consumes and changes the status of some resources;

(viii) Desired effect is part of the capability and guides the activity;

(ix) Resource supports capability, and the function and performance of resources will affect the evaluation of a capability;

Meta-rule is an extensible finite set of fundamental rules, including logic rules and business rules, which represents the constraints that need to be held by all concepts and associations, and which provides a set of domain general rules for cooperative capability modeling in the emergency response domain. The meta-rules of CCR meta-model can be summarized as follows:

(i) The object CCR requires must be a capability, not a resource, an activity or any other;

(ii) Every capability must be an execution result of one or more activities;

(iii) Every activity must be performed by at least one performer;

(iv) A performer can perform one or more activities;

(v) Every rule can constraint one or more capabilities.

Moreover, researchers and modelers are allowed to extend some essential associations and rules depending on their study needs.

3.2 Application for modeling in emergency response domain

Based on the formal definition of CCR meta-model, we are able to model the cooperative capability in the emergency response domain. We will illustrate the application of the CCR meta-model in the modeling field through a simple example of emergency rescue response for a marine accident [22]:

Basic scenario: suppose a marine accident occurs at 1 000 km from a coastal city, the rescue coordination center (RCC) should detect the warning signal in 5 min, then notify the coast guard force (CGF) to carry out the search and rescue task and the medical assistance center (MAC) to prepare to rescue the people in the accident.

During this emergency rescue response process, the RCC, CGF and MAC should perform cooperatively, to model the cooperative capability in this ERS, there may be some basic ERS domain models of concepts, associations and rules, which are shown in Table 1.

Table 1 An application of CCR meta-model for modeling in ERS domain

Element	Meta-model	In ERS domain
Concept	CCR	Emergency rescue capability in marine accident (ERCMA)
	Capability	Search and rescue capability (SaRC) Medical care capability (MCC)
	Activity	Air-based rescue (ABR) Water-based rescue (WBR) Rescue victims in hospital (RVH)
	Performer	CGF RCC MAC
	Resource	Search and rescue cutter (SRC) Search and rescue plane (SRP) Medical apparatus and instruments (MAI) Early-warning radar (EWR)
	System	Communication tools (CT) Emergency rescue platform (ERP)
	DesiredEffect	Get to the marine accident site in 20 mins (GMAS2M) EWR detects warning signals in 5 mins (DWS5M)
Association	Activity should be performed by performers	CGF performs WBR and ABR MAC performs RVH
	Activity consumes and changes the status of some resources	WBR needs 20 SRCs and ABR needs 3 SRPs
	Resource supports capability, and the function and performance of resources will affect the evaluation of a capability	SRC supports SaRC, but not supports ABR SRP supports ABR
	Cooperative capability is a kind of complex capability and should be composited by several simple capabilities	ERCMA needs SaRC and MCC to work cooperatively
Rule	The object CCR requires must be a capability, not a resource, an activity or any other	ERCMA needs SaRC, not SRP
	A performer can perform one or more activities	CFG can perform WBR by using SRC, and also can perform ABR by using SRP
	Every rule can constraint one or more capabilities	WBR and ABR should work simultaneously RVH should work after WBR or ABR

4. Executable framework overview

In this study, we propose an executable framework that transforms Alf models into CSP models, which is significantly helpful for simulating the execution process of ERS automatically and eliminating the semantic [23] and inconsistency [24] gaps between models transformation. The framework also generates a consistency validation report which will capture the error and warning messages about inconsistency and semantic gaps during the transforming process. Please refer to Fig. 2 which is leveraged to illustrate the working process of overall executable framework visually.

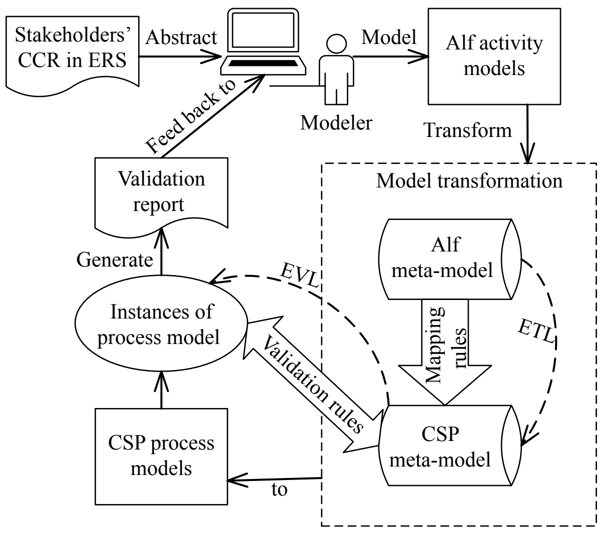


Fig. 2 Executable framework overview

Initially, modelers analyze the stakeholders' CCRs in the emergency response domain and then abstract them into Alf activity models. To capture various kinds of behaviors in an ERS, the models are represented in the form of the Alf activity model which is able to describe the active objects (ActivityNode) and behaviors (Action) during its execution process. As we illustrate in the previous section, the evaluation of the cooperative capability in an ERS is reflected by the activities executed by performers and embodied in the processes of the executing behaviors, as a result, we may evaluate the cooperative capability of an ERS by analyzing the execution effect of activities (or behaviors) of an ERS. Consequently, in Alf models, we abstract a CCR in ERS into an "Activity" meta-model, with the functionality and the performance requirements into its "Action" and "ActivityParameterNode" elements respectively. The "ActivityNode" and "ActivityEdge" are essential elements to completely describe an execution of cooperative behaviors in an ERS, the "ActivityNode" element contains all kinds of active objects in an ERS system, while the "ActivityEdge" element will provide control connection or data passing between active objects.

We will get a corresponding textual Alf "Activity" meta-model by mapping Alf concrete syntax with textual Alf code to CCR meta-model in the form of UML abstract syntax [12]. Models in the style of Alf textual code is convenient to, as Alf language is designed by use of the Java-like syntax, be interpreted and executed on a target platform. Please note that Alf is a model aware action language, not a programming language, since modeling with a programming language will hide some significant behavior semantics and result in inconsistency and semantic gaps which are not conductive to satisfy stakeholders' requirements and simply maintain a system.

Secondly, we will transform the textual Alf activity meta-model into a CSP process meta-model, by using ETL [20], which are beneficial for reasoning business logic and consistency validation based on several mapping rules designed depending on the features of Alf and CSP. As modelers create instances of CSP process models to simulate the execution process of an ERS, validation rules are designed, by using EVL, to support consistency validation between these instances and CSP meta-model according to their research and analyzing needs. Then a validation report will be generated to capture the error and warning messages of inconsistency and semantic gaps which arise during the validation process. In addition, EVL is utilized to provide a self-fixing mechanism to fix the inconsistency and semantic gaps automatically. And finally the validation report will be fed back to modelers, which is significantly helpful to optimize and maintain the executable framework for improvement.

5. Model transformarion of the framework

The primary functionality of model transformation of the framework is to transform the Alf activity meta-model into a corresponding CSP process meta-model, after which rigorous algebra rules [19] are able to be utilized for formal logical reasoning about and consistency validation between system models, we will accomplish these tasks in three stages:

(i) Complete the transformation from Alf activity meta-model into CSP process meta-model based on several mapping rules using ETL;

(ii) Configure the internal communication mechanism between CSP process nodes to smooth the working mechanism of a CSP system;

(iii) Validate the CSP models resulted from step (i) using EVL and generate a validation report to be fed back to the modeler for system optimization.

We will illustrate each of these stages in detail in the following sections.

5.1 Mapping rules of model transformation

To accomplish the task, transforming the Alf activity meta-model into the CSP process meta-model, mapping rules should be declared first for us to comply with during the transformation process.

Generally speaking, activities are classes and may also have attributes and operations in UML [8]. However, in Alf, the specification does not provide a textual notation for features and specializations on activities. Consequently, in our study, the "Activity" element in Alf is used to represent a specific behavior in ERS, just a single behavior, which is equivalent to the element "Action" in UML models, and to be transformed into a process in the CSP model. Just like an activity is accomplished through executing several actions by active objects in UML models, in the CSP model, several processes can be combined into a single process to represent the whole behavior of an ERS. For modeling active objects in real world, which are described in UML as "ActivityNode", we model them by using the "Active Class" element in Alf. When come to the interactions between objects in an ERS, we summarize them into two kinds of system behaviors (sending signals and accepting signals), and any one of specific behaviors in ERS can be characterized by defining the format of the signal and sending it out to a specific accepting object. For example, by defining the format of the signal "submitRescueOrder" shown in Fig. 3, the RCC is able to introduce the information about a marine accident to the CGF and order the CGF to rescue the victims of the accident. Consequently, the coordinate activity is completed by sending the signal, which also reflects that the RCC possesses the coordinate capability.

```
Signal submitRescueOrder() {

        //marine accident information
        public marineAccidentSite:          Location;
        public marineAccidentTime:          Date;
        public NumOfVictims:                Integer;
        public marineAccidentDescription:   String;

        // Capability Requirement
        public RescueTime:            Date;
        public requiredMaterials:     Resource;
        public performer:             Performer;
        public CooperativeRule:       Rule;

    }
```

Fig. 3 An example of the format of a signal

In Fig. 4, we exhibit the mapping rules from Alf metamodel to CSP meta-model, in order to visualize the mapping relationships between the two different domains models, we also list the corresponding UML models out. From the mapping rules shown in the table, A_{NS} and A_{NR} represent an instance of the active object who is in charge of sending signals and accepting signals respectively [16]. It is fundamental for an executable system that each of the objects of a CSP system should be uniquely identified, otherwise, it will make analysts confused when they are analyzing the system or a deadlock phenomenon will arise during the system execution. The interactions between objects will be carried out by sending and accepting signals, through defining the format of signals, which will enable active objects to execute a specific domain behavior. We now illustrate the mapping rules concretely as follows:

(i) Mapping an "activity" element to a process in the CSP model

The "activity" element is a fundamental mechanism for behavior modeling in Alf models [12]. Modelers usually use the "activity" element to model a single behavior in ERS. During the model transformation process, we transform an "activity" in Alf into a process "Act" in CSP. Both "activity" and "Act1" represent a single behavior in their application domain, and can be merged or combined into a composite activity model or a composite process respectively to represent multiple behaviors of a whole system.

(ii) Mapping an active class to a process node in the CSP model

An active class is an instantiation of an activity object in the UML model, it has its own attributes, operations as well as nested classifiers [8], the most significant feature of the active class in Alf is that only an active class has receptions and, just because of this, we transform an active class into a process node which is able to satisfy the requirement of sending and accepting signals in the CSP model.

(iii) Mapping the interactions between activity nodes "sending signals" and "receiving signals" to parameterized events "send" and "receive" respectively in the CSP model

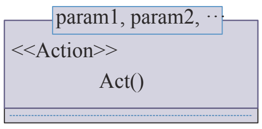
Signals with a specific format, with arguments matched with attributes in the parameter list, will be sent and received between activity nodes. In a CSP system, the sending (or accepting) process will trigger a send (or accept) event, which will activate a process node to execute a process.

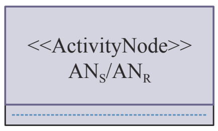
(iv) Transformation of the sequential execution in Alf models

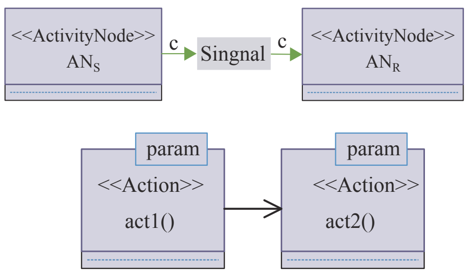
Generally speaking, in Alf syntax, behaviors included in a block (such as a do block statement) are executed sequentially in order unless such a notation "@parallel" is labeled ahead of the block, which is coincident with that in CSP process models, for example, the format "P; Q" represents that process Q will continually to be executed only after the process P is terminated successfully.

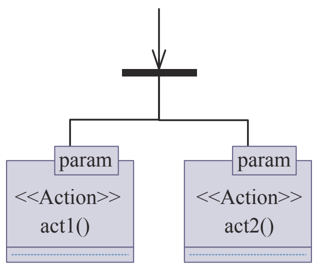
(v) Transformation of the parallel execution in Alf models

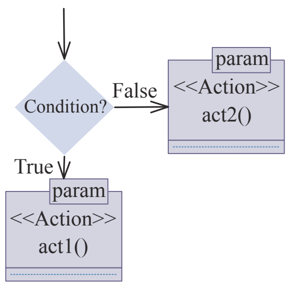
UML model	Alf element	CSP representation

	activity Act(param1,param2)
	{
	…
	}
		Act: ==
		Let process body
		within Act

	active class AN_S (AN_R) {
	…
	}
		$AN_S \| AN_R$

 <<ActivityNode>>
AN_S
c
Singnal
c
<<ActivityNode>>
AN_R
param
<<Action>>
act1()
param
<<Action>>
act2()
	AN_S. send (signal, ANR);
	AN_R. accept (signal);
	do{
	act1();
	act2();
	} while (true)
		Act1:=send!AN_S!AN_R!signal
		→⋯
		Act2:=accept! AN_R!signal →⋯
		Act1;Act2

 param
<<Action>>
act1()
param
<<Action>>
act2()
	do{
	//@parallel
	act1();
	act2();
	}while (true)
		Act1‖Act2

 Condition?
False
param
<<Action>>
act2()
True
param
<<Action>>
act1()
	//@assured @determinate
	if (condition==a){
	act1();
	}else if (condition ==b){
	act2();
	}else{……}
		((a→Act1) Π (b→Act2))
		∨
		((a→Act1) | (b→Act2))

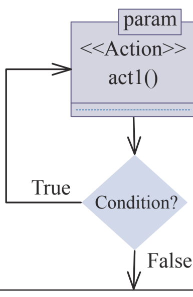
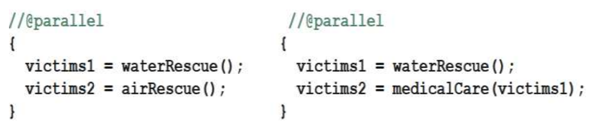
 param
<<Action>>
act1()
True
Condition?
False
	for (condition== true){
	act1 ();
	}
		μX:{X in condition}·Act1

Fig. 4 Mapping rules from Alf to CSP model

In Alf syntax, a notation "@parallel" is applied in a block statement to represent that several behaviors in the block are executed concurrently, usually on the basis that activities in the block are data independent (e.g., Fig. 5 (a)), if there is a data dependence (or more) between the behaviors, they will be executed sequentially (e.g., Fig. 5 (b)). In the CSP model, the parallel execution in the Alf model is mapped to a symbol "‖" which is used to represent two processes are executed in parallel, please note that parallel processes are executed cooperatively on some events, but their behaviors are not able to be merged or combined to act as one process.



```
//@parallel
{
  victims1 = waterRescue();
  victims2 = airRescue();
}
```

```
//@parallel
{
  victims1 = waterRescue();
  victims2 = medicalCare(victims1);
}
```

(a) Data independent (b) Data dependent

Fig. 5 Examples of parallel execution

(vi) Transformation of the conditional execution in Alf models

An if statement or a switch statement (which is not showed in Table 1) may be used to execute a set of concurrent behaviors based on the evaluation of the conditions, a significant distinguish between the two kinds of conditional execution is that the evaluation of the conditional expression in the if statement only can be true or false, while in the switch statement it can be other values conforming to the type of the condition expression. If an evaluation conforms to the conditional expression, an associate block will be executed, but if there are more than one evaluation conforming to the conditional expression, one of the associate blocks will be chosen non-deterministically to execute. Notations "@assured" and "@determinate" are used to indicate that there are at least one or at most one evaluation conforming to the conditional expression respectively. Moreover, both of the two notations can also be used simultaneously to represent that there is exactly one evaluation conforming to the conditional expression. Usually in the CSP model, under a conditional execution context, the operate symbol "|" is used, corresponding to the situation "@assured @determinate" in the Alf model, to represent that the process "Act1" will be executed if the event is "a", otherwise the "Act2" will be executed if the event is "b". While corresponding to the situation "@assured" in Alf, the operate symbol "Π" is used to represent that one process will be chosen to execute, but non-deterministically, from multiple associate processes whose evaluation conforms to conditional expression.

(vii) Transformation of the iterative execution in Alf models

In Alf syntax, a "for" statement or a "do…while" statement (which is not showed in Fig. 4) can be used to represent iterative execution of behaviors. On each iteration, the loop variable will continually increment or decrement its value until it no longer satisfies the conditions to go on executing. In the CSP model, we use such a format as "μX: A•F(X)" to represent that a process is iteratively executed, and the A in this format is a collection of the evaluation of loop variable X under the loop conditions.

5.2 Implementation of transformation

The family of epsilon languages and tools, during the transformation process, is leveraged to accomplish the task of model-to-model transformation and consistency validation within the eclipse modeling framework (EMF) [25]. We will accomplish the task in two steps and Fig. 6 will assist us to illustrate the steps [21]: firstly, since the Alf activity meta-model and CSP process meta-model are not existing in the meta-model library of EMF, we should create the activity meta-model (in Alf) and the process meta-mode (in CSP) in the beginning, and the EMF will parse both meta-models into XMI format [26] in order that both meta-models can be easily identified by arbitrary MDE framework for transformation because XMI is a general purpose format in the software modeling domain. Secondly, on the basis of source/target meta-models, transforming the activity meta-model into the process meta-model with ETL is going to be carried out, which should conform to the fundamental mapping rules configured in Fig. 4.

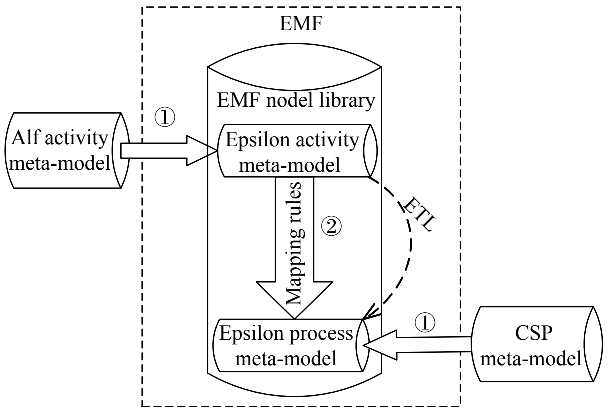
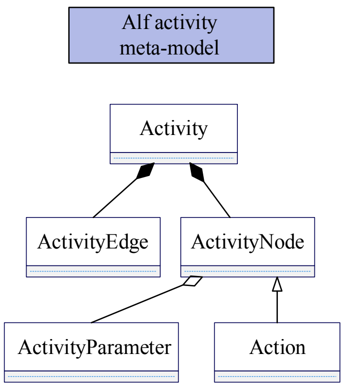


Fig. 6 Steps of model transformation

(i) Generating meta-models

To create Alf activity meta-model and CSP process meta-model with EMF, we adopt the method that uses annotated Java interfaces with model properties, to be as specific as possible, by placing the tag "@model" ahead of some significant Java interfaces (and also an "@param" tag ahead of some elements of the model, e.g., attributes and operations), the EMF generator can identify the tag automatically and generate corresponding models conforming to the interfaces. Consequently, we should code Java interfaces with the "@model" tag ahead of them first, which can wholly represent the features of Alf activity meta-model and CSP process meta-model. Fig. 7 will assist us to demonstrate the design process of both corresponding interfaces of Alf activity meta-model and CSP process meta-model.

Meta-models	Annotated Java interfaces

 Alf activity meta-model
Activity
ActivityEdge
ActivityNode
ActivityParameter
Action
	/** @model*/
	public interface Activity extends EObject {
	/** @model*/
	String get Activity Name ();
	/** @param*/
	void setActivityName (String value);
	/** @model*/
	Signal getSignal ();
	/** @param*/
	void setSignal (Signal value);
	/** @param*/
	void setActivityNode (ActivityNode node);
	/** @param*/
	void setActivityEde (ActivityEdge edge);
	/** @model containment="true"*/
	EList<ActivityNode> getActivityNodes ();
	/** @model containment="true" */
	EList<Edge> getEdges ();
	/** @model containment="true"*/
	EList<Signal> getSignals();
	} // Activity

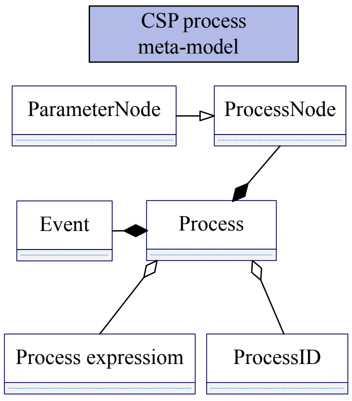
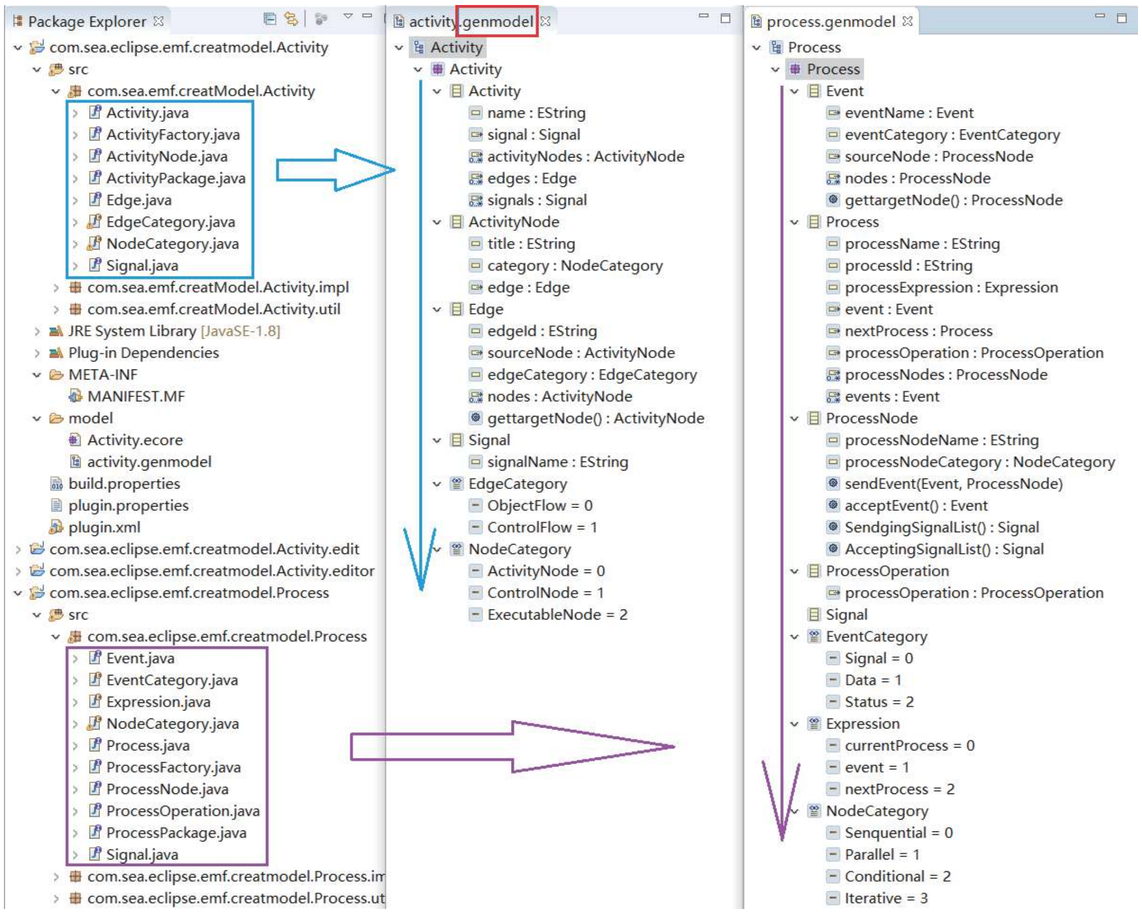
 CSP process meta-model
ParameterNode
ProcessNode
Event
Process
Process expressiom
ProcessID
	/** @model*/
	public interface Process extends EObject {
	/** @model*/
	String getProcessExpression ();
	/** @param*/
	void setProcessExpression (Expression value);
	/** @model*/
	String getProcessId ();
	/** @param*/
	void setProcessId (String value);
	/** @model*/
	ProcessOperation getProcessOperation ();
	/** @param*/
	void setProcessOperation (ProcessOperation operation);
	/** @model*/
	Event getEvent();
	/** @param*/
	void setEvent (Event value);
	/** @model*/
	EList<ProcessNode> getProcessNodes();
	/** @model*/
	EList<Event> getEvents();
	} // Process

Fig. 7 Annotated Java interface of meta-models

An Alf activity meta-model should include an "ActivityNode" and an "ActivityEdge" whose functions and features have been already introduced in our previous presentation. As a result, in the interface we design for the Alf activity meta-model, some functions to get attributes of the two elements (the "ActivityNode" and the "ActivityEdge") should be declared, for example, setters and getters operation. Moreover, interfaces of the two elements, as well as some specific domain category classes should also be created too. For the limitation of literal space, we will not exhibit detailed information of the interfaces and classes we design.

Similar to Alf activity meta-model, a CSP process meta-model should include some information about "processNode" and "Event", as well as a process expression, which will make a process execute with a certain manner under some specific conditions. In the context of a CSP system, once an "Event" is triggered through sending (or accepting) some signals or passing some data (or messages), the event will activate a "processNode" to execute a behavior of a process. Consequently, we design some annotated Java interfaces for "Process", "ProcessNode" and "Event", as well as some classes about the detailed domain category. We also declared some process operation functions to facilitate the execution and logical reasoning of processes.

The whole hierarchical structures of activity and process meta-models generated in the EMF model library are shown in Fig. 8, in which the left part shows detailed information of the interfaces and classes we design for generating activity and process meta-models, while the right two parts show the hierarchical structures of the meta-models generated by the EMF generator. To mention that a file with the suffix “.genmodle” represents that it is a model result generated by the EMF generator through parsing the annotated Java interfaces and classes modelers have designed. Then we are enabled to run the meta-models on the eclipse platform as an application, and Fig. 9 shows that the meta-models we design have been imported into the EMF models library and applied as plugins of eclipse, modelers are facilitated to create instances of the meta-models and override the attributes and functions of the models to satisfy their research requirements.

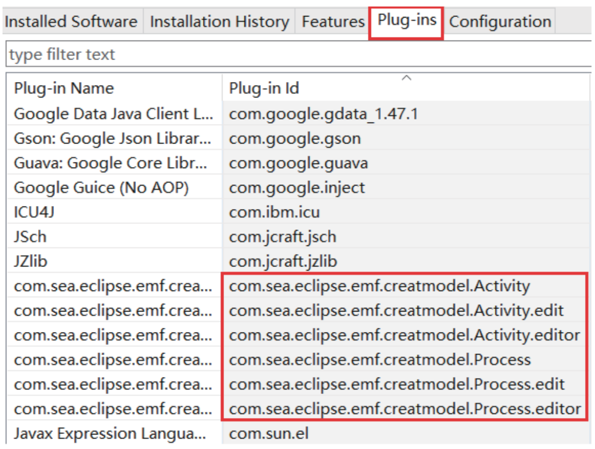


```
Package Explorer
 com.sea.eclipse.emf.creatmodel.Activity
   src
     com.sea.emf.creatModel.Activity
       Activity.java
       ActivityFactory.java
       ActivityNode.java
       ActivityPackage.java
       Edge.java
       EdgeCategory.java
       NodeCategory.java
       Signal.java
     com.sea.emf.creatModel.Activity.impl
     com.sea.emf.creatModel.Activity.util
   JRE System Library [JavaSE-1.8]
   Plug-in Dependencies
   META-INF
     MANIFEST.MF
   model
     Activity.ecore
     activity.genmodel
   build.properties
   plugin.properties
   plugin.xml
 com.sea.eclipse.emf.creatmodel.Activity.edit
 com.sea.eclipse.emf.creatmodel.Activity.editor
 com.sea.eclipse.emf.creatmodel.Process
   src
     com.sea.eclipse.emf.creatmodel.Process
       Event.java
       EventCategory.java
       Expression.java
       NodeCategory.java
       Process.java
       ProcessFactory.java
       ProcessNode.java
       ProcessOperation.java
       ProcessPackage.java
       Signal.java
     com.sea.eclipse.emf.creatmodel.Process.im
     com.sea.eclipse.emf.creatmodel.Process.ut
```

```
activity.genmodel
 Activity
   Activity
     Activity
       name : EString
       signal : Signal
       activityNodes : ActivityNode
       edges : Edge
       signals : Signal
     ActivityNode
       title : EString
       category : NodeCategory
       edge : Edge
     Edge
       edgeId : EString
       sourceNode : ActivityNode
       edgeCategory : EdgeCategory
       nodes : ActivityNode
       gettargetNode() : ActivityNode
     Signal
       signalName : EString
     EdgeCategory
       ObjectFlow = 0
       ControlFlow = 1
     NodeCategory
       ActivityNode = 0
       ControlNode = 1
       ExecutableNode = 2
```

```
process.genmodel
 Process
   Process
     Event
       eventName : Event
       eventCategory : EventCategory
       sourceNode : ProcessNode
       nodes : ProcessNode
       gettargetNode() : ProcessNode
     Process
       processName : EString
       processId : EString
       processExpression : Expression
       event : Event
       nextProcess : Process
       processOperation : ProcessOperation
       processNodes : ProcessNode
       events : Event
     ProcessNode
       processNodeName : EString
       processNodeCategory : NodeCategory
       sendEvent(Event, ProcessNode)
       acceptEvent() : Event
       SendgingSignalList() : Signal
       AcceptingSignalList() : Signal
     ProcessOperation
       processOperation : ProcessOperation
     Signal
     EventCategory
       Signal = 0
       Data = 1
       Status = 2
     Expression
       currentProcess = 0
       event = 1
       nextProcess = 2
     NodeCategory
       Senquential = 0
       Parallel = 1
       Conditional = 2
       Iterative = 3
```

Fig. 8 Hierarchical structure of meta-models



Installed Software | Installation History | Features | Plug-ins | Configuration

type filter text

Plug-in Name	Plug-in Id
Google Data Java Client L...	com.google.gdata_1.47.1
Gson: Google Json Librar...	com.google.gson
Guava: Google Core Libr...	com.google.guava
Google Guice (No AOP)	com.google.inject
ICU4J	com.ibm.icu
JSch	com.jcraft.jsch
JZlib	com.jcraft.jzlib
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Activity
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Activity.edit
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Activity.editor
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Process
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Process.edit
com.sea.eclipse.emf.crea...	com.sea.eclipse.emf.creatmodel.Process.editor
Javax Expression Langua...	com.sun.el

Fig. 9 Meta-models applied in EMF model library

(ii) Transformation from Alf to CSP model

As activity and process meta-models are generated into the EMF models library, according to our research program, model transformation from activity meta-model into process meta-model is carried out by using ETL. Before the transformation task, a previous work should be completed, based on the mapping rules shown in Fig. 4, that is configuring the mapping rules (with the keyword “rule”) and operations (with the keyword “operation”) in ETL.

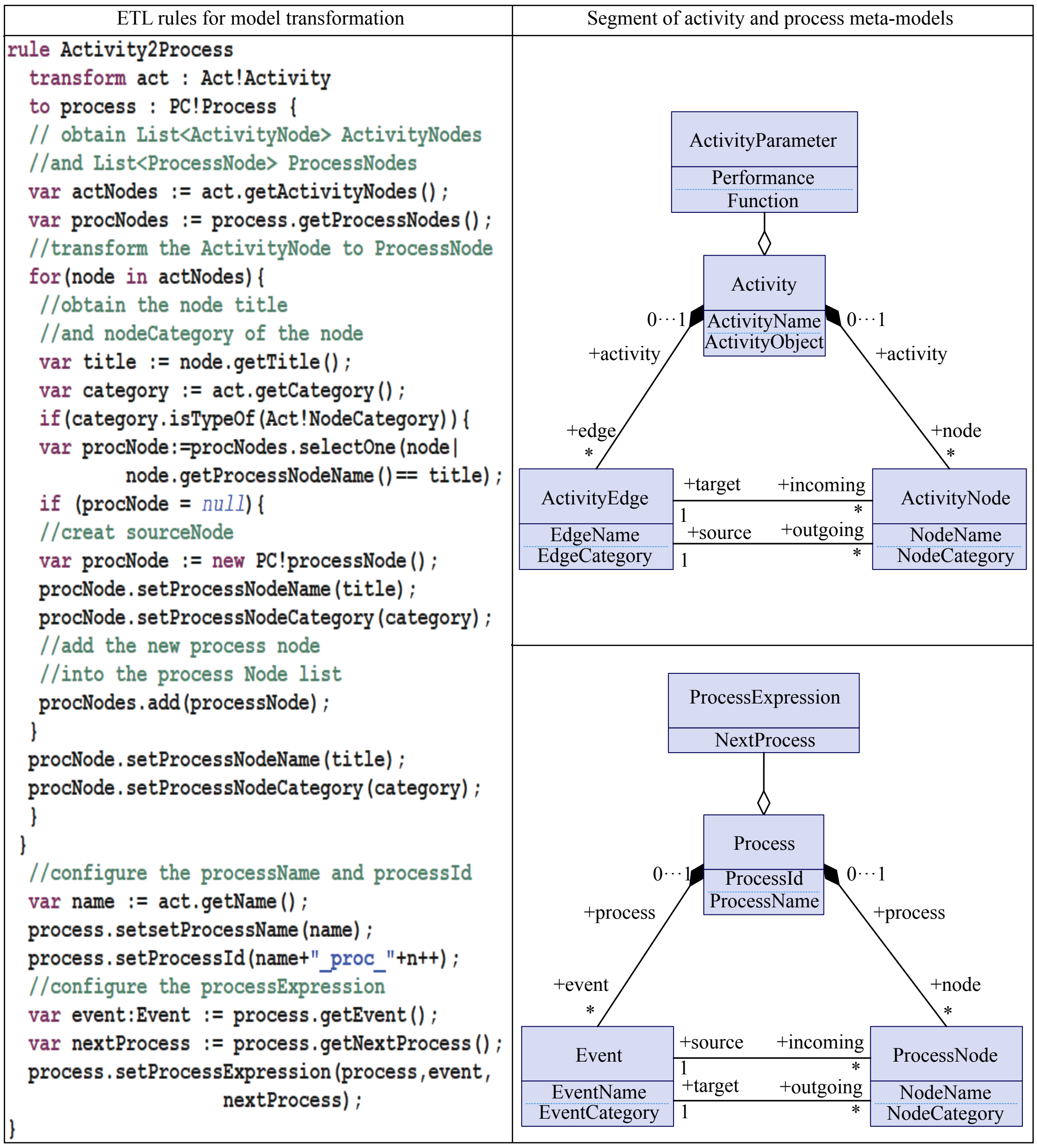
In order to make our explanation clear and coherent, readers need to be aware of the following issues:

i) In ETL, Alf elements and CSP elements can be accessed by the prefix “ACT” and “CSP”, respectively, followed by the operator “!”.

ii) According to the mapping rules proposed in Fig. 4, an activity meta-model is transformed into a process meta-model, with an ActivityNode transformed into a processNode, and an ActivityEdge will be transformed into an event in the CSP model.

iii) In order to take the executions such as sequential execution, parallel execution, conditional execution and iterative execution into account, we consider them as a kind of activity node, that is the executable node, which can be inherited by a derived class and overriding the elements.

Since there are so many elements, rules and operations we propose, considering the literal space limitation, we will take a representative element transformation, from source element "Activity" into target element "process", for example. We will illustrate the detailed transformation process with the assistant of Fig. 10, which displays an ETL transform code description on the left and a visual description of the meta-model transformation mapping relationships on the right. In Section 6, the internal communication mechanism between process nodes is to be introduced.


ETL rules for model transformation

```
rule Activity2Process
  transform act : Act!Activity
  to process : PC!Process {
  // obtain List<ActivityNode> ActivityNodes
  //and List<ProcessNode> ProcessNodes
  var actNodes := act.getActivityNodes();
  var procNodes := process.getProcessNodes();
  //transform the ActivityNode to ProcessNode
  for(node in actNodes){
   //obtain the node title
   //and nodeCategory of the node
   var title := node.getTitle();
   var category := act.getCategory();
   if(category.isTypeOf(Act!NodeCategory)){
   var procNode:=procNodes.selectOne(node|
           node.getProcessNodeName()== title);
   if (procNode = null){
   //creat sourceNode
   var procNode := new PC!processNode();
   procNode.setProcessNodeName(title);
   procNode.setProcessNodeCategory(category);
   //add the new process node
   //into the process Node list
   procNodes.add(processNode);
  }
  procNode.setProcessNodeName(title);
  procNode.setProcessNodeCategory(category);
  }
 }
 //configure the processName and processId
 var name := act.getName();
 process.setsetProcessName(name);
 process.setProcessId(name+"_proc_"+n++);
 //configure the processExpression
 var event:Event := process.getEvent();
 var nextProcess := process.getNextProcess();
 process.setProcessExpression(process,event,
                   nextProcess);
}
```

Fig. 10 Model transformation from activity to process with ETL rules

When transforming an activity element into a process element, as is introduced in the previous section, the two elements ProcessExpression and ProcessId should be evaluated before since we can uniquely identify a process through the two elements, including what the process is and what it will behave in a system.

In a CSP system model, there may be many process nodes, every of which may execute many behaviors, that is to say, one process node may execute many processes. Before generating a process model from an activity model, we should ensure that a corresponding process node has already existed in the processNodes list, which represents a collection of process nodes in the CSP model. If there truly is such a corresponding process node, a process model may be generated on this base; however, if there is no such a process node, a new process node should be created first and added into the process nodes list, after which a corresponding process model is able to be generated continually. The "for" block shown in Fig. 10 is used to realize the function of navigating the processNodes list and creating an activity node described above. It is possible that the function may also be used in other transformation rules, consequently, we will encapsulate it into an operation for consideration of code optimization in later stages.

To evaluate a processId, we define a global integer variable n. Every time a process model is transformed from an activity model, we will generate a processId by concatenating the corresponding activity name and the value of the variable n with a string "_proc_" in their middle, after which the variable n will be increased by one automatically. Consequently, we can ensure that every process in a CSP system has a unique processId to be identified.

A processExpression consists of three parts: a current process name, an event name and its next process name, with a symbol "→" used for concatenation between the event and the next process. Because in a CSP system, the process is activated by triggering an event. To evaluate a processExpression, a specific format of a processExpression should be defined first, we define it in the "Expression" class, which we have declared at the meta-models generating step. Fig. 11 shows the format definition extracted from the "Expression" class, from which we can see the format conforms to that shown in Fig. 4.

```
/**
 * @model
 */
public String setExpression( Process currentProcess,
                             Event event, Process nextProcess) {
    return (currentProcess.getProcessName()+":="+
                     event.getEventName() + "→" +
                     nextProcess.getProcessName());
}
```

Fig. 11 Format definition of process expression

6. Internal communication mechanism

In this section, we intend to introduce the communication mechanism between process nodes in a CSP system. The interactions between process nodes are usually conducted by sending and accepting signals or data with specific formats via a channel. In a CSP system, sending a signal or data means triggering an event, which may activate a process node to execute its process, whether the passing information is a signal or data can be characterized by setting function setEventCategory(). As to how to deal with the mechanism of sending and accepting information, we bring in the design of the structure of a process node and the ETL transformation rules of transforming a sending (or accepting) action in Alf into a send (or accept) event in a CSP system. Fig. 12 is utilized to assist us in illustrating the communication mechanism in a CSP system.

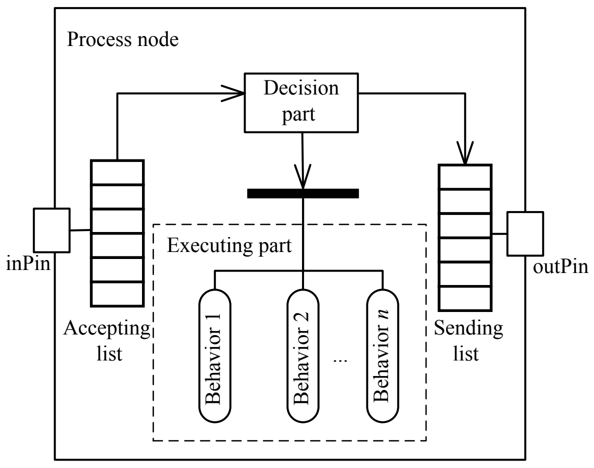


Fig. 12 Structure of a process node

As is shown in Fig. 12, each process node in a CSP system consists of a decision part and an executing part, as well as two queues which are implemented by using the list collection in epsilon for handling information; one list is connected to the inPin for accepting information while the other list is connected to the outPin for sending information out. The decision part is the head of a process node, which will be in charge of processing the information to be sent out or examining whether accepting information is required by itself, as well as controlling the executing part to execute its behaviors process; while the executing part is the hand of a process node, which is in charge of maintaining several behaviors the process node can execute and executing its behaviors process under the decision part's command. With regard to the information dispatching mechanism of a queue, we will not take it into consideration for the time being and assume that it is configured with the default first-in first-out (FIFO) dispatching mechanism, since the issue we do research on is an executable method for models, and we may fix this problem in the system optimization stage later.

When a process node intends to send information out to other process nodes such as passing data or asking for cooperative behaviors, before it executes the "send the information out" behavior, the decision part of the process node should deal with the information and encapsulate it into a specific format which meets the requirements depending on its application domain. Once the information is in place, the decision part executes the "send the information out" behavior and the CSP system creates an event to be triggured at the same time. Nevertheless, the information is not sent out and passed along a channel immediately but stored in the sending list, as in a large scale CSP system multiple signals or data may be sent simultaneously. Unless a channel is free to send the information, thereupon the event is triggered and it will invoke the method "load (Signal signal)" to execute the "send the information out" behavior. At the moment, the information is released from the list and loaded onto the event, goes out from the outPin of the sending process node and passes along the channel connecting the two communication process nodes.

When a process node intends to accept information, initially the information is in place to arrive in the accepting process node, the event will invoke the method "unload (Signal signal)" and the information is released from the event and passed in the inPin of the accepting process node. Nevertheless, the decision part of the accepting node does not receive the information and read it immediately too but stores it in the accepting list, waiting for the scheduling of the dispatching mechanism of the queue and the working mechanism of the decision part.

When the decision part intends to read the information from the accepting list, firstly it will examine whether the information conforms to the format constraint which is right to activate the process node to execute the behaviors of its own, preventing that the information is sent to a wrong process node. If the information format is right and able to activate the process node, the decision part will accept the information and invoke corresponding behaviors to execute. After the execution has been done, the decision part will create an event which records the status of its execution and send it back to the sending process node. If the information format is wrong or it is sent to a wrong accepter for mistake, the decision part of the process node will discard the information and do create an event which records the information that "the information is wrongly sent" and send it back to the sending process node.

7. A case study for consistency validation

In this section, a consistency validation between emergency response domain models and the CSP meta-model is illustrated, which will examine that whether a contradiction of consistency and semantics gaps arises during the modeling process [27]. For example, whether the associations between emergency response process instances and a CSP process meta-model break the rule constraints (logical constraints and business constraints) declared in the process meta-model. Firstly, we also take the marine accident case for example to carry out the consistency validation work, we configure the process meta-model [28] with several essential rules constraints [29], and modelers are also able to configure more rule constraints according to their research requirements:

(i) Every class of a process instance must have a class name which starts with an upper case letter, and the features (name, attribute, parameter) of a process instance must starts with a lower case letter;

(ii) The object CCR requires must be a capability, not a resource, an activity or any other;

(iii) Every process instance must inherit the "Process" meta-model we create in the EMF model library;

(iv) Every process must be executed by at least one processNode;

(v) The standard format of a process is defined as "eventName: String → processName: String ", in which the "eventName" represents a name of an event while the "processName" represents a name of a process. It means that in a CSP system every process is executed by triggering an event [10]. However, a process defined by the format that only with a single process name, like "processName : String", which commonly appears in several papers is not correct in fact.

We configure these rules above with EVL code, and Fig. 13 shows several code segments of rules as there are so much code that we cannot show all of them out for space limitations, which will examine all the models in the ERS of the marine accident.

```
context Process!NamedElement {
// Every NamedElement must define a name
  constraint MustHasName {
  check : self.name <> ""
  message:"NameElement " + self +
           " must define a name"
  }}
context Process!Feature {
  // The name of a feature(name,attribute,parameter)
  //should start with a lower case letter
  critique FeatureMustStartWithLowerCase {
  guard : self.satisfies("MustHasName")
  check : self.name.substring(0,1).isLowerCase()
  message:"The feature '"+self+" ' of the class"+
  self.inherits.name+"should start with a lower case"
  fix {
       title : "Rename process name " + self.name +
               " to " +self.name.firstToLowerCase()
     do {
         self.name = self.name.firstToLowerCase();
  }}}}
  // A class must not directly or indirectly
  // inherit from itself
  constraint MustInheritProcessMetaModel {
     check : not self.inherits(Process)
     message : "the process '" + self.name +
              "' doesn't inherits 'Process' meta-model"}
  context Process!Class {
  // The name of a class should start
  //with an upper case letter
  critique ClassNameShouldStartWithUpperCase {
  guard : self.satisfies("MustHasName")
  check : self.name.substring(0,1).isUpperCase()
  message:"The name of the class '" + self.name +
          "'' should start with an upper case letter"
  fix {
       title : "Rename class name " + self.name +
               " to " + self.name.firstToUpperCase()
       do {
           self.name = self.name.firstToUpperCase();}
 }}
```

Fig. 13 Segments of rules for consistency validation

Suppose that a modeler is asked to model the ERS of the marine accident, and he constructs the models as Fig. 14 shows, in which some models break the constraint rules.

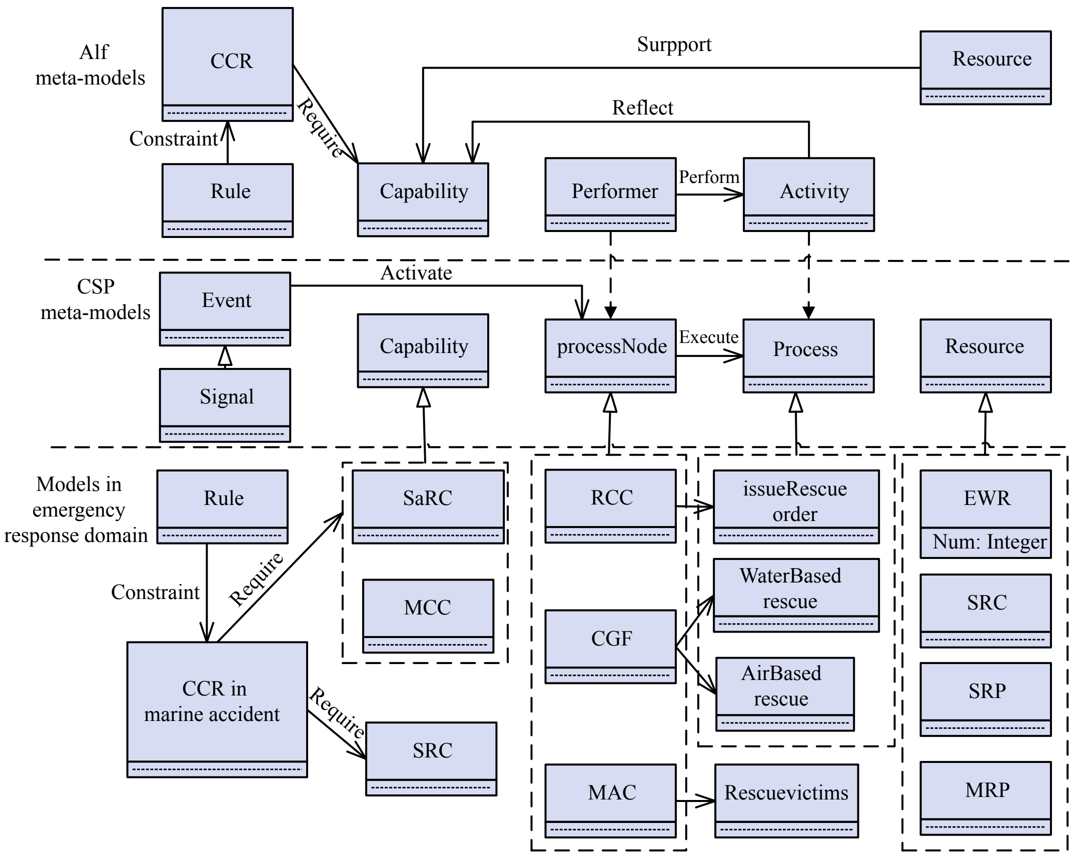
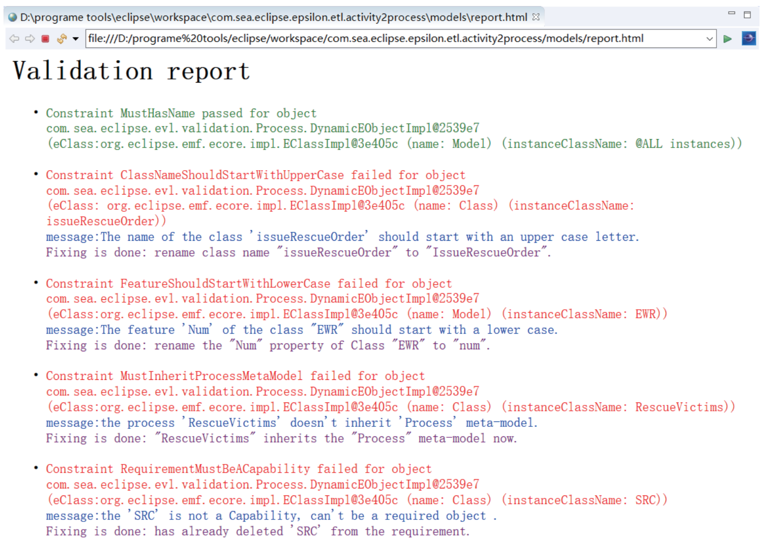


Fig. 14 A modeling example of the executable framework in emergency response domain

When an error message or a warning message is captured because of inconsistency during the validation process [30], the EVL self-repairing mechanism is leveraged to repair the identified inconsistency automatically by defining a fixing functionality in a fix block, which can significantly improve the usability of the code and consequently enhance users' productivity. Fig. 15 shows a validation report with html format, which will be generated and fed back to modelers for system optimization.



```
D:\programe tools\eclipse\workspace\com.sea.eclipse.epsilon.etl.activity2process\models\report.html
file:///D:/programe%20tools/eclipse/workspace/com.sea.eclipse.epsilon.etl.activity2process/models/report.html

Validation report

• Constraint MustHasName passed for object
  com.sea.eclipse.evl.validation.Process.DynamicEObjectImpl@2539e7
  (eClass:org.eclipse.emf.ecore.impl.EClassImpl@3e405c (name: Model) (instanceClassName: @ALL instances))

• Constraint ClassNameShouldStartWithUpperCase failed for object
  com.sea.eclipse.evl.validation.Process.DynamicEObjectImpl@2539e7
  (eClass: org.eclipse.emf.ecore.impl.EClassImpl@3e405c (name: Class) (instanceClassName:
  issueRescueOrder))
  message:The name of the class 'issueRescueOrder' should start with an upper case letter.
  Fixing is done: rename class name "issueRescueOrder" to "IssueRescueOrder".

• Constraint FeatureShouldStartWithLowerCase failed for object
  com.sea.eclipse.evl.validation.Process.DynamicEObjectImpl@2539e7
  (eClass:org.eclipse.emf.ecore.impl.EClassImpl@3e405c (name: Model) (instanceClassName: EWR))
  message:The feature 'Num' of the class "EWR" should start with a lower case.
  Fixing is done: rename the "Num" property of Class "EWR" to "num".

• Constraint MustInheritProcessMetaModel failed for object
  com.sea.eclipse.evl.validation.Process.DynamicEObjectImpl@2539e7
  (eClass:org.eclipse.emf.ecore.impl.EClassImpl@3e405c (name: Class) (instanceClassName: RescueVictims))
  message:the process 'RescueVictims' doesn't inherit 'Process' meta-model.
  Fixing is done: "RescueVictims" inherits the "Process" meta-model now.

• Constraint RequirementMustBeACapability failed for object
  com.sea.eclipse.evl.validation.Process.DynamicEObjectImpl@2539e7
  (eClass:org.eclipse.emf.ecore.impl.EClassImpl@3e405c (name: Class) (instanceClassName: SRC))
  message:the 'SRC' is not a Capability, can't be a required object .
  Fixing is done: has already deleted 'SRC' from the requirement.
```

Fig. 15 An example of consistency validation report

8. Conclusions

In this study, to deal with the lack of official specification and fundamental theory basis for CCR, we propose a CCR meta-model as a theory basis for researchers to refer to in this research domain, in which we provide detailed definition of the CCR meta-concepts, meta-associations and meta-rules; and an executable framework, which may enable modelers to simulate the execution process of a system in advance and ensure the model verification and consistency validation when they try their best to fill the inconsistency and semantic gaps between stakeholders' requirements and executable models.

The framework bases on the model transformation from Alf activity models into CSP process models with ETL, as well as the inter-object communication mechanism between process nodes. Moreover, the framework also provides consistency validation and self-fixing mechanism with EVL to make the created instances conform to the rule constraints of meta-models.

We have created meta-models of Alf activity and CSP process and imported them into the EFM model library to work as a plug-in in the case tool Eclipse. Then the mapping rules are designed by using ETL to facilitate the transformation from Alf activity meta-model into CSP process meta-model, as well as the implementation of communication mechanism between process nodes.

Consistency validation is significantly important in capturing the errors and warnings between the instances and the meta-model under the rule constraints with EVL, in this paper, we just take some essential rules for example, researchers and modelers are able to design their own rule constraints (logical constraints or business constraints) according to their research requirements. A validation report will be generated and fed back to the modelers for reference and the EVL self-repairing mechanism will be leveraged to improve the usability of code and consequently enhance modelers' productivity.

Currently, just work as a theoretical basis, the CCR meta-model and executable framework are used in emergency response domain. More rules and associations should be taken into consideration depending on the specificity of different area domains (such as the medical domain and the military domain) in our future work, which is significantly helpful for us to enrich our CCR meta-model and the executable framework optimization. We also will adapt the framework to include some safety and security specifications checking too.

References

[1] DoD Architechture Framework Working Group. DoD architechture framework version V2.0. https://dodcio.defense.gov/Library/DoD-Architecture-Framework/.

[2] DONG Q C, WANG Z X, CHEN G Y, et al. Domain-specific modeling and verification for C4ISR capability requirements. Journal of Central South University, 2012, 19(5): 1334–1340.

[3] ZHANG T T, LIU X M, WANG Z X, et al. Capability-oriented architectural analysis method based on fuzzy description logic. Computer Science and Information Systems, 2015, 13(1): 287–308.

[4] CICCOZZI F, MALAVOLTA I, SELIC B. Execution of UML models: a systematic review of research and practice. Software and Systems Modeling, 2019, 18(3): 2313–2360.

[5] ABDOLI S, SAMI K. A modelling framework to design executable logical architecture of engineering systems. Modern Applied Science, 2017, 11(9): 75–91.

[6] BERGMAYR A, BRUNELIERE H, CABOT J. fREX: fUML-based reverse engineering of executable behavior for software dynamic analysis. Proc. of the 8th Workshop on Modelling in Software Engineering—Co-located with ICSE 2016, 2016: 20–26.

[7] CICCOZZI F. On the automated translational execution of the action language for foundational UML. Software and Systems Modeling, 2018, 17(4): 1311–1337.

[8] Object Management Group. Unified modeling language (UML) V2.5. 1. https://www.omg.org/spec/UML/.

[9] SOLTANA G, SANNIER N, SABETZADEH M, et al. Model-based simulation of legal policies: framework, tool support, and validation. Software and Systems Modeling, 2018, 17(3): 851–883.

[10] DRAGOMIR I, OBER I, PERCEBOIS C. Contract-based modeling and verification of timed safety requirements within SysML. Software and Systems Modeling, 2017, 16(2): 587–624.

[11] Object Management Group. Semantics of a foundational subset for executable UML models. https://www.omg.org/spec/FUML.

[12] Object Management Group. Action language for foundational UML (Alf) V1.1. https://www.omg.org/spec/ALF/.

[13] BAI Y, ZHANG Y X, ZHOU Y Z. Process algebra-based formal service description method. Journal of Tsinghua University, 2012, 52(12): 1769–1775. (in Chinese)

[14] VANGLABBEEK R J. Communicating sequential processes. http://theory.stanford.edu/people/rvg/abstracts.html#1.

[15] LIN K P, CHAO W S. The structure-behavior coalescence approach for systems modeling. IEEE Access, 2019, 7(1): 8609–8620.

[16] ABDELHALIM I, SCHNEIDER S, TREHARNE H. An integrated framework for checking the behaviour of fUML models using CSP. International Journal on Software Tools for Technology Transfer, 2013, 15(4): 375–396.

[17] MICHAEL H. Essential business process modeling. Sebastopol, California: O'Reilly Media, 2009.

[18] CIOCCHETTA F, HILLSTON J. Bio-PEPA: an extension of the process algebra PEPA for biochemical networks. Electronic Notes in Theoretical Computer Science, 2008, 194(3): 103–117.

[19] HOARE T, STADEN S V. The laws of programming unify process calculi. Science of Computer Programming, 2014, 85(Part B): 102–114.

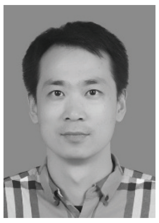
[20] DIMITRIS K, LOUIS R, ANTONIO G D, et al. The Epsilon book, 2018. https://www.eclipse.org/epsilon.

[21] LIU C, TANG T. Epsilon-based model transformation and verification of train control system specification. Proc. of the 30th Chinese Control Conference, 2011: 5562–5567.
[22] ZHANG Z H, ZHANG X L, XU Z J, et al. Emergency countermeasures against marine disasters in Qingdao City on the basis of scenario analysis. Natural Hazards, 2015, 75(2): 233–255.
[23] LIMA L, MIYAZWA A, CAVALCANTI A, et al. An integrated semantics for reasoning about SysML design models using refinement. Software and Systems Modeling, 2015, 16(3): 875–902.
[24] LUO R B, GAO S S, LI H L, et al. Modeling and verification of reconfigurable printing system based on process algebra. Mathematical Problems in Engineering, 2018, 2018: 9189836.
[25] DAVE S, FRANK B, MARCELO P, et al. EMF: eclipse modeling framework. 2nd ed. New Jersey: Addison-Wesley Professional, 2008.
[26] Object Management Group. XML metadata interchange (XMI). https://www.omg.org/spec/XMI/2.5.1/PDF.
[27] BRUNNER J, LAMMICH P. Formal verification of an executable LTL model checker with partial order reduction. Journal of Automated Reasoning, 2018, 60(1): 3–21.
[28] GLABBEEK R V, HOFNER P H, MARKL M. A process algebra for link layer protocols. Programming Languages and Systems, 2019, 11423: 668–693.
[29] BALABAN M, MARA E A. Removing redundant multiplicity constraints in UML class models. Software and Systems Modeling, 2019, 18(4): 2717–2751.
[30] XIANG S Q, WU X, ZHU H B, et al. Modeling and verifying basic modules of floodlight. Mobile Networks and Applications, 2019, 24(1): 100–114.

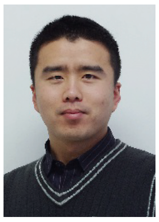
Biographies

CHAI Lei was born in 1986. He received his M.S. degree from the Institute of Software Engineering, the University of Science and Technology of China. He is a Ph.D. student of the Institute of Command and Control Engineering, the Army Engineering University of PLA. His research interests are requirements engineering, software engineering, focusing on specification and formal verification.
E-mail: heyuekunhong@126.com

WANG Zhixue was born in 1961. He received his B.S. degree from Hefei Polytechnic University, M.S. degree from the National University of Defense and Technology, and used to be a visiting researcher in the Faculty of Information Technology, University of Brighton, England. He is a professor of the Institute of Command and Control Engineering, Army Engineering University of PLA. His research interests are software engineering, requirements engineering, and theory and technology of command automation, currently focusing on domain-specific modeling and formal verification.
E-mail: wzx_cx@163.com



HE Ming was born in 1978. He received his B.S., M.S. and Ph.D. degrees from the Army Engineering University of PLA, Nanjing, in 2000, 2003 and 2007, respectively. He is a professor in the Army Engineering University of PLA. His main research interests focus on emergency command, big data analytics, Internet of Things and public safety.
E-mail: 1456167138@qq.com



HE Hongyue was born in 1986. He is a lecturer of the Institute of Command and Control Engineering, Army Engineering University of PLA. His research interests are system of systems (SoS) engineering and the theory of command and control, specification and formal verification.
E-mail: hehy2008@sina.com

YU Minggang was born in 1986. He is a lecturer of the Institute of Command and Control Engineering, Army Engineering University of PLA. His research interests are system of systems (SoS) engineering and the theory of command and control, focusing on evolutionary game.
E-mail: yuminggang8989@163.com