# A Formal Approach for Modeling and Verification of Distributed Systems

Gang Ren[1,2], Pan Deng[1,2](✉), Chao Yang[1,2], Jianwei Zhang[3], and Qingsong Hua[4]

[1] Institute of Software, Chinese Academy of Sciences, Beijing 100091, China
{rengang2013,dengpan,yangchao}@iscas.ac.cn
[2] University of Chinese Academy of Sciences, Beijing 100091, China
[3] Beihang University, Beijing 100091, China
zhangjw@nlsde.buaa.edu.cn
[4] Qingdao University, Qingdao 266071, China
8988596@qq.com

**Abstract.** In recent year, distributed systems have become a mainstream paradigm in industry and how to ensure correctness and reliability is a great challenge for practicing engineers. Therefore, in this paper a formal approach is proposed for modelling and verification of distributed systems, which integrates UML sequence diagram, $\pi$-calculus and NuSMV within one framework. Moreover, the practicality of the proposed approach is illuminated though a case study of scheduling road emergency service.

**Keywords:** UML seqence diagram · $\pi$-calculus · Model checking · Formal methods

## 1 Introduction

In recent year, with the rapid development of distributed systems, they have become a mainstream paradigm in industry and how to ensure correctness and reliability is a great challenge for practicing engineers [1]. They are investigating various methods and tools to address this issue.

In these methods, there are three methods widely used. One is UML sequence diagram [2], which is graphical notation for specifying dynamic interaction behaviors among system components. As it is intuitive and simple in notation and semantics, it is appealing to practicing engineer.

Another method widely used is $\pi$-calculus [3], which is a process algebra and well reputed for modelling concurrent systems and mobile systems. Due to excellent ability of expression, it has been widely applied to modelling of system dynamic behavior.

Project supported by National Nature Science Foundation of China (No. 61100066).

The third is NuSMV [4], which is a symbolic model checker which can automate verification process checking whether or not the desired temporal properties can be met.

Generally speaking, these methods improved correctness and reliability of safety-critical systems to some extent. Nevertheless, these methods and tools are always used separately. Because different methods focuses on different aspects of system, only one method can't solve this challenge very well. For example, due to intuition and simplicity, UML sequence diagram [2] is widely used in modeling of systems. However, it is a semi-formal notation and not suitable very well for distributed systems. $\Pi$-calculus [3], a process algebra, and NuSMV [4], a symbolic model checker, are another two methods used widely. However, $\pi$-calculus is only good at system specification, but not at system verification. NuSMV [4] is just the reverse.

Therefore, in this paper, a formal approach is proposed for modeling and verification of distributed systems. As shown in Fig. 1, there are 3 layers in this framework. The first is graphical layer, which utilizes sequence diagram to model system. The middle is formal specification layer which is adopted $\pi$-calculus to formalize UML sequence diagram. The third is verification layer, which adopts NuSMV as verification tool.

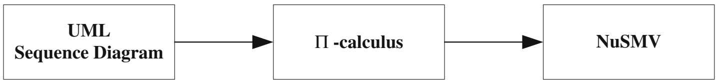


**Fig. 1.** A formal approach for modelling and verification of distributed systems.

## 2 Modelling a Scenario Using Sequence Diagram

Recently, road emergency services in Intelligent Transportation Systems have draw much attention [5,6]. In this paper, a scenario of scheduling emergency services is considered as an examination to illustrate the use of the proposed framework. As Fig. 2 shows, there are three kinds of Emergency Services (ESs): Hospital, Fire Bridge (FB) and Police Office (PO) in road network. When Road Side Unit (RSU) discoveries accident, it'll inform it to Emergency Center (EC). EC is in charge of forwarding this message into relative ESs automatically. When EC receives a request from RSU, it'll forward the message into relative ESs. Once ESs receives a reqeust from EC, it'll respond is at once. The UML sequence diagram of the scenario is described as shown in Fig. 3. Exchanging message between devices is in an asynchronous way. The combined fragment of parallel is used in sending "Request" in parallel.

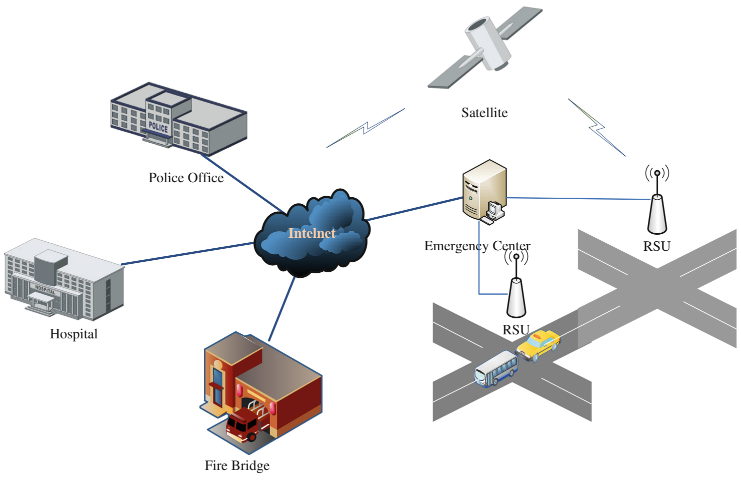


**Fig. 2.** A scenario of scheduling emergency services.

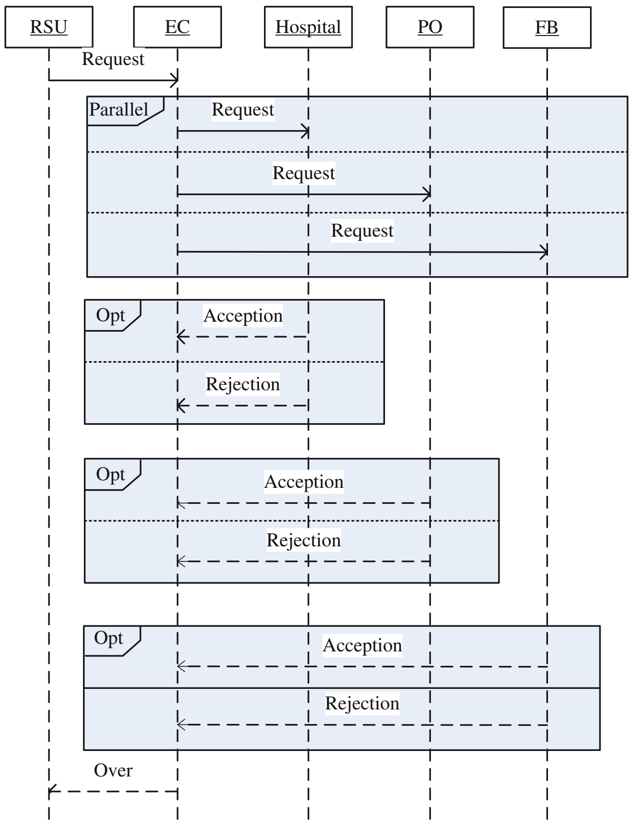


**Fig. 3.** Sequence diagram of emergency service.

## 3 From Sequence Diagram to *Π*-calculus

In this section, first a set of rules are defined. Then according to the above rules, $\pi$-calculus presentations of the road emergence service are specified.

**Rule 1 (Sending message).** *Given a device, the sending of a message is specified as an output action in $\pi$-calculus.*

**Rule 2 (Receiving message).** *Given a device, the receipt of a messages is specified as an input action in $\pi$-calculus.*

**Rule 3 (Parallel combined fragment).** *Given a device, a parallel combined fragment is specified concurrent action in $\pi$-calculus.*

**Rule 4 (Optional combined fragment).** *Given a device, a optional combined fragment is specified choice action in $\pi$-calculus.*

(1) **System**
$System = RSU|EC|Hopital|FB|PO$

(2) **RSU**
$RSU = \overline{re}\langle Request\rangle.re(msg).[msg = Over]0$

(3) **EC**
$EC = EC_0|EC_1|EC_2|EC_{syn}$
$EC_0 = re(msg).[msg = Request]\overline{eh}\langle Request\rangle.0$
$EC_1 = re(msg).[msg = Request]\overline{ef}\langle Request\rangle.0$
$EC_2 = re(msg).[msg = Request]\overline{ep}\langle Request\rangle.0$
$EC_{syn} = eh(msg).([msg = Acception]ef(msg).[msg = Acception]ep(msg).$
$[msg = Acception]\overline{re}\langle Over\rangle.0$

(4) **Hospital, FB and PO**
$Hospital = eh(msg).[msg = Request](\overline{rh}\langle Acception\rangle + \overline{rh}\langle Rejection\rangle).0$
$FB = ef(msg).[msg = Request](\overline{rf}\langle Acception\rangle + \overline{rf}\langle Rejection\rangle).0$
$PO = ep(msg).[msg = Request](\overline{rp}\langle Acception\rangle + \overline{rp}\langle Rejection\rangle).0$

## 4 From *Π*-Calculus into NuSMV

The translation rules are formally defined as follows:

**Rule 1.** *System process are device process are translated into main module and sub-module program and device process is translated into sub module respectively.*

**Rule 2.** *Given a process, its concurrent processes are translated into sub-processes executing in parallel.*

**Rule 3.** *In every module, there is a local enumerative scalar variable "state" which represents the transition of process states.*

**Rule 4.** *As for an input action, its previous state and match structure are both translated into a transition condition of "state" and its successor is translated into a next state of "state".*

**Rule 5.** *As for an output action, its previous state and subsequent state are respectively translated into a transition statement of "state".*

## 5 Model Checking for Temporal Property

In the following lists, the temporal properties to be verified are defined and translated into Computation Tree Logic (CTL) formulas which is a branching temporal logic and extends propositional logic by incorporating path quantifiers and temporal operators. For more details about syntax and semantics of CTL, please refer to [7].

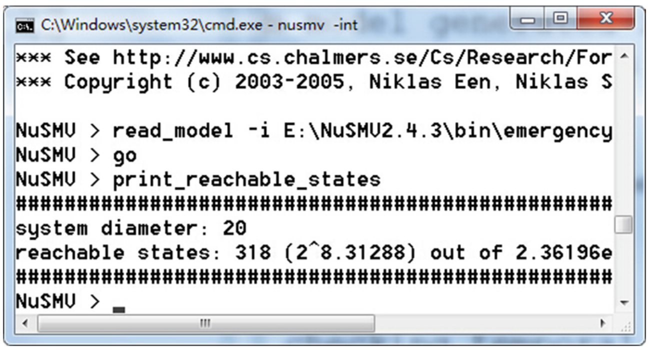
**Property 1 (Non-Blocking).** *EC must reach the "Over" status in the future.*
$AF(re = Over)$

**Property 2 (Result Reachability).** *It is possible that All ECs can reach "Acception".*
$AF(eh = Acception\&ef = Acception\&ep = Acception)$

**Property 3 (Forwarding Integrity).** *Once EC receives a "Request" from Hospital, it must forward "Request" into ESs.*
$AF((re = Request) - > AF(eh = Request\&ef = Request\&ep = Request))$

**Property 4 (Forwarding Stability).** *EC can't forward message until it receives a message.*
$A[(eh! = Request\&ef! = Request\&ep! = Request)U(re = Request)]$

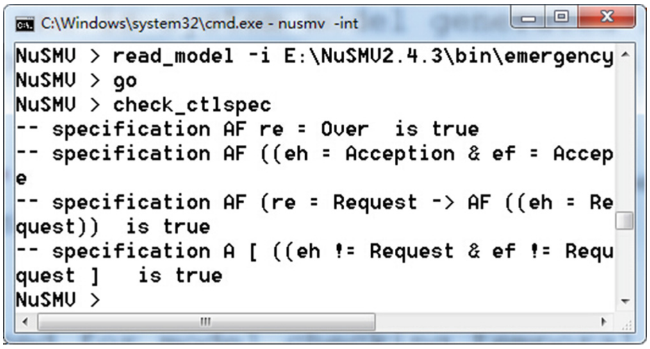
The execution environment comprises 8 core CPU of Intel Core i7-2600(3.40 GHz) with 4 G Memory, running Windows 7 and NuSMV 2.4.3. Firstly, the command of "*read_model*" is used in reading model file "ES.svm", which contains two parts: the one is system model generated in Sect. 4. The other one is temporal properties defined above. Secondly, the command of "*print_reachable_state*" is executed to count the number of the reachable states. As shown in Fig. 4, there are totally 318 reachable states. Finally, the command of "*check_ctlspec*" can be used in model checking temporal properties. Figure 5 shows temporal properties are all met.



```
*** See http://www.cs.chalmers.se/Cs/Research/For
*** Copyright (c) 2003-2005, Niklas Een, Niklas S

NuSMV > read_model -i E:\NuSMV2.4.3\bin\emergency
NuSMV > go
NuSMV > print_reachable_states
######################################################
system diameter: 20
reachable states: 318 (2^8.31288) out of 2.36196e
######################################################
NuSMV > _
```

**Fig. 4.** Reachable states



```
NuSMV > read_model -i E:\NuSMV2.4.3\bin\emergency
NuSMV > go
NuSMV > check_ctlspec
-- specification AF re = Over  is true
-- specification AF ((eh = Acception & ef = Accep
e
-- specification AF (re = Request -> AF ((eh = Re
quest))  is true
-- specification A [ ((eh != Request & ef != Requ
quest ]   is true
NuSMV >
```

**Fig. 5.** Verification results

## 6 Conclusion

In this paper, a novel 3-layer framework is proposed for modelling and verification of safety-critical systems, which integrates three different methods and tools, and leverages their respective advantages to collaborate each other. It can improve correctness and reliability of safety-critical systems such as Industrial IoT system and applications. The major advantage of this work is that it frees the designer to know about mathematical formalisms where the learning curve might be high.

As future work, an translator from sequence diagram to $\pi$-calculus is considered to develop in order to improve automation of this framework.

## References

1. Knight, J.C.: Safety critical systems: challenges and directions. In: Proceedings of the International Conference on Software Engineering, pp. 547–550, Orlando, May 2002
2. Omg unified modeling language (omg uml). Normative Reference (2013). http://www.omg.org/spec/UML/2.5
3. Milner, R., Parrow, J., Walker, D.: A calculus of mobile processes, i. Inf. Comput. **100**(1), 1–40 (1992)
4. Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.: NuSMV 2: an opensource tool for symbolic model checking. In: Brinksma, E., Larsen, K.G. (eds.) CAV 2002. LNCS, vol. 2404, pp. 359–364. Springer, Heidelberg (2002)
5. Chen, C., Chen, P., Chen, W.: A novel emergency vehicle dispatching system. In: Vehicular Technology Conference, pp. 1–5, June 2013
6. Rajamaki, J.: The mobi project: designing the future emergency service vehicle. IEEE Veh. Technol. Mag. **8**(2), 92–99 (2013)
7. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge (2000)