

On the Problem-Oriented Verification of Cyber-Physical Systems Using System-Level Test Sequences

Changlan Fu, Xiao Zhang, Zhi Li*, Ziyang Zhao, Chao Wang, Yuekun Yu
College of Computer Science and Information Technology
Guangxi Normal University
Guilin, Guangxi, China
*corresponding author, zhili@gxnu.edu.cn

Abstract—The heterogeneous implementations of Cyber-Physical Systems (CPS), including complex behaviors of physical devices and human users, pose significant challenges for verifying such systems. Since Jackson’s Problem Frames approach (PF) provides facilities for representing interactions between the computing and physical components of CPS, it is applicable to the requirements analysis and modeling of CPS. In this work, we propose an approach to verifying whether the requirements are satisfied or not using system-level testing methods for CPS. A set of supporting tools have been developed for modeling and verifying CPS with test obligations and acceptance criteria for system-level testing prior to design time, which aims at reducing defects in requirements elicitation and documentation, thus supporting backtracking activities in the requirements analysis of a software development process. The work is illustrated in a real-world example.

Keywords—Problem Frames; requirements verification; test sequences; cyber-physical systems;

I. INTRODUCTION

With the development of information technology, we will see increasing use of the Internet of Things, big data analytics platforms, and appearances of Cyber-Physical Systems (CPS)^[1], which is a series of tight integration of computational processes and physical processes. Deep integration and real-time interaction are achieved through feedback loops in which computational processes and physical processes interact. With the increasing complexity of CPS, the testing and maintenance of such systems become very difficult, which may cause faults, failures or even great security risks.

Recently, some researchers use heterogeneous model fusion technology to achieve integration of the computing and physical systems^[2]. For example, modeling languages such as UML, Modelica, and Simulink have extended their modeling elements. Although existing CPS modeling and simulation techniques reflect some advances in its verification technology, they are far from being able to meet the needs for verifying large-scale CPS design processes.

The Problem Frames (PF)^[3] approach, which was first proposed by Michael A. Jackson in the field of software engineering, has established that software development problems can be divided into three parts: software S , real world W , and user requirements R . S represents a software system to

be built; W represents the real-world environment in which the software system runs; W can be regarded as the physical component in the CPS architecture. PF provides facilities for representing the interactions between the computing and physical components of CPS, thus supporting the modeling and verification of complex CPS behaviors^[4,5].

In this paper, we model the behavior of CPS and its requirements based on PF, and verify whether the integration of its computing and physical systems meets the user’s requirements in terms of completeness and correctness, which can help find system defects and avoid serious failures. We propose a system-level testing method for CPS, and develop a set of supporting tools which can systematically assist software developers in modeling, and verifying cyber-physical systems prior to design time.

II. SYNTACTIC CHECKING OF PF DIAGRAMS

In PF, the computing machine domain is represented by the symbol \square , the problem domain is represented by the symbol \square and the requirement is represented by the symbol \circ . The connecting lines between the machine domain and problem domains are called interfaces, represented by the symbol --- ; the connecting lines between the requirements and problem domains can either be requirements references (represented by the symbol ---) or requirement constraints (represented by the symbol ---), as shown in Figure 1 (on the next page).

A. Integrity and correctness in PF syntax

The integrity of PF diagram refers to a set of basic completeness rules which are fundamental to PF models. Table 1 lists a sample of rules for PF. Note that rules can be accumulated and added as practitioners gain more experience of using the PF modeling.

Table 1. Integrity conditions of the problem diagram

No.	Integrity conditions
1	The name of the domain must be not be empty.
2	There is at least one machine domain in the diagram.
3	A problem diagram has at least one requirement.
.....

The correctness of PF diagram refers to the fundamental principles that must be obeyed in PF^[3], as shown in Table 2.

Table 2. Correctness conditions of the problem diagram

No.	Correctness Conditions
1	The name of the domain must be unique.
2	The requirement cannot directly constrain the machine domain.
3	Each machine domain controls at least one interface.
.....

When system analysts model the requirements, they hope that the model can help the requirements analysts to check if the design is complete and correct, so as to avoid the situation in which the design is difficult for the developers to understand. Here is an example of an incomplete and incorrect problem diagram, as shown in Figures 1 and 2. One problem domain in Figure 1 has no name, and the requirements in Figure 2 directly constrains the machine domain (by definition, a requirement that directly constrains the computing machine is called a "specification").

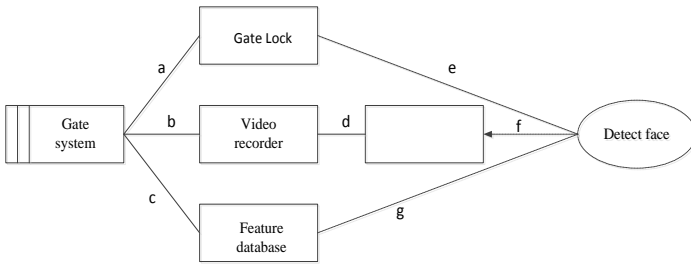


Fig. 1. A security gate control problem diagram (domain nameless)

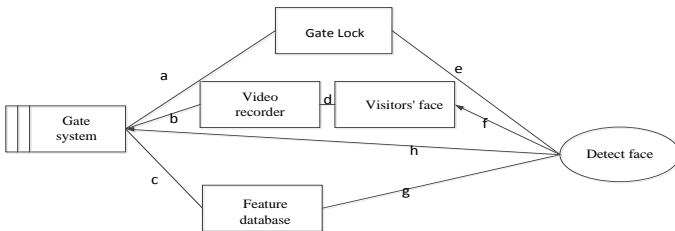


Fig. 2. A security gate control problem diagram (requirements directly constrain the machine)

B. A brief description of OCL

The object constraint language (OCL) is a language for applying constraints on specified model elements. OCLs express constraints on the object with the conditions and constraints attached to the model elements, including invariant or constraint expressions attached to the model elements, pre-conditions and post-attachments attached to operations, and methods and conditions, etc^[6-7].

C. Using OCL to implement integrity and correctness check

We adopted the OCL's consistency verification method for UML models. We use OCLs to define PF constraints, and to develop the integrity and correctness verification modules by

analyzing and checking OCL expressions^[8]. The problem diagram constraint conditions in Tables 1 and 2 are represented by the following OCLs.

Constraint: The name of the domain must be set

OCL expression:

```
Class.allInstances()-
>select(c|c.oclAsType(Class).getValue(c.oclAsType(Class).ge
tAppliedStereotypes()->asSequence()->first(),'value')=null) -
>size()=0
```

Constraints: There is at least one machine domain in the context diagram

OCL expression:

```
Class.allInstances()-
>forAll(p|p.oclAsType(Class).getAppliedStereotypes().name-
>includes('Machine')->size())>=1)
```

Constraint: A problem diagram or framework has at least one requirement

OCL expression:

```
Class.allInstances()-
>forAll(p|p.oclAsType(Class).getAppliedStereotypes().name-
>includes('Requirement')->size())>=1)
```

Constraint: The name of the domain must be unique

OCL expression:

```
Class.allInstances()-
>select(getAppliedStereotypes().name->includes('Domain'))-
>isUnique(name)
```

Constraint: The requirement cannot directly constrain the machine domain

OCL expression:

```
Interface.allInstances()-
>select(a|a.oclAsType(Dependency).getAppliedStereotypes().
name->includes('constrains'))-
>forAll(source.getAppliedStereotypes().name-
>includes('Requirement')implies not
target.getAppliedStereotypes().name->includes('Machine'))
```

Constraint: Each machine domain controls at least one interface

OCL expression:

```
Interface.allInstances()-
>select(getAppliedStereotypes().name-
>includes('observes')).target-
>forAll(ot|Interface.allInstances()-
>select(getAppliedStereotypes().name->includes('controls'))-
>select(target->exists(ct|ct=ot))->size())=1)
```

III. COMPLEX PROBLEM DECOMPOSITION

Hall provides a de-notational semantics for Problem Frames in [9], in which a generic problem diagram can be expressed as follows:

$c, o : [K, R] = \{S \mid S \text{ controls } c \wedge S \text{ observes } o \wedge K, S \mid \text{-}_{DRDL} R\}$,
where S represents the software *solution* to be built, K represents knowledge about S 's context (i.e., physical devices or human-beings) and R represents user requirements. Figure 3 shows the corresponding generic problem diagram (Note: $S!c$ denotes " S controls c ", $S?o$ denotes " S observes o ", and $DRDL$ is the short for a requirement and domain description language.

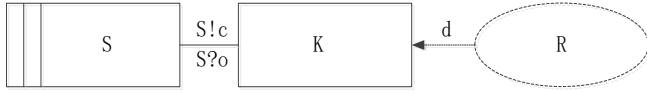


Fig. 3. A generic problem diagram

In this paper, a chain of causality is introduced to facilitate the understanding of complex problem diagrams and to automate the search for causal chains^[10].

A. Causal chain

From Figure 3, we can see that if the domain sharing phenomenon o results in the occurrence of c , then this is a causal relationship.

In a problem diagram, if we find a path from R to S , or from R to S and to R , then we say that we have found a solution to the problem. The elements of this path populate the set of solutions to the problem. There are multiple such paths in a complex problem diagram, that is, there are multiple chains of causality from R to S , and each path may be respectively represented as $R_1, S_1, R_2, S_2, \dots, R_n, S_n$.

We extend Hall's de-notational semantics by introducing the causality chain concept, as follows:

$$c, o : [K, R] = \{ S \mid S ! c \wedge S ? o \wedge K, S \mid \text{-DRDL } R \} \\ = \{ S1 \parallel S2 \parallel \dots \parallel Sn \}$$

Figure 4 corresponds to a partial problem diagram, which shows a set of solution.

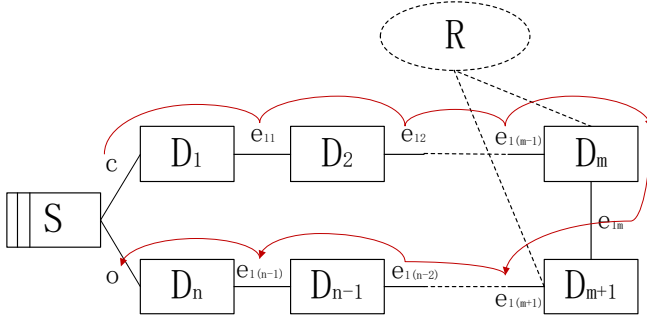


Fig. 4. A partial problem diagram

IV. SECURITY GATE CONTROL PROBLEM DIAGRAM: A CASE STUDY

A security gate control problem is used to illustrate the work presented in this paper. The following is a rough sketch of the problem:

A computer that recognizes facial features is required to control the security gate. The face of each person who wants to enter the security gate is captured on a video tape. The records in the database are compared with the captured face features. These records contain facial features that have been explicitly accessible. Figure 5 is a diagram of the security gate control problem.

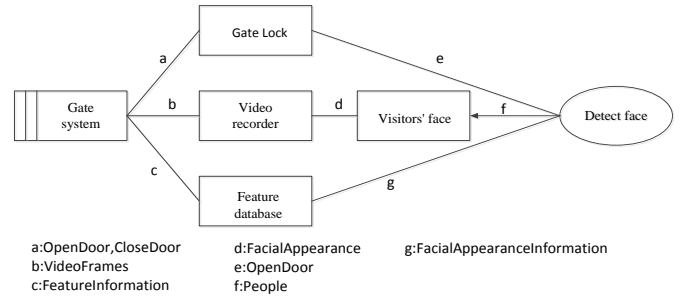


Fig. 5. Security gate control problem diagram

By comparing Figures 3 with 5, we can find the relationship between the problem diagram and Hall's de-notational semantics and two chains of causality, as shown in Figure 6.

$$S = \{ \text{Gate system} \},$$

$$c = \{ a \} (b \text{ is initialized by } S \text{ and therefore controlled by } S),$$

$$o = \{ b, c \} (b, c \text{ is received by } S, \text{ so it is observed by } S),$$

$$K = \{ \text{Gate lock, Video recorder, Visitors' face, feature database} \},$$

$$R = \{ \text{Detect face} \},$$

$$d = \{ e, f, g \}.$$

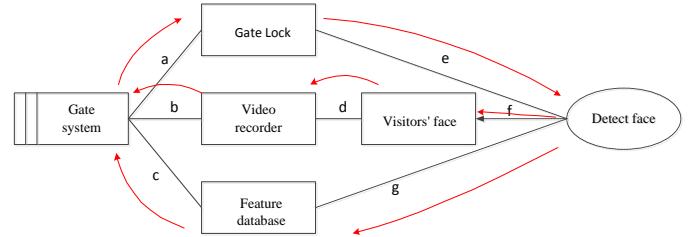


Fig. 6. Security gate control problem diagram causality chain

V. REQUIREMENTS SATISFACTION VERIFICATION STUDY

Precise mathematical methods and technologies are used in formal requirements verification, in which requirement models, are represented by mathematical expressions, operations and derivations so that ambiguities, incompleteness, unachievable expressions in requirement models can be detected or discovered. The research in this paper is based on the Communication Sequential Process (CSP) algebraic theory and CSP scripts are generated for the description and verification of the requirements model.

A. Overview of Communication Sequential Process

The Communication Sequential Process is an algebraic theory proposed by the well-known computer scientist C.A.R. Hoare^[11]. It is an abstract description language for parallel algebraic systems and specifically describes message interactions in concurrent systems. Because CSP is suitable for modeling and analyzing systems and describes complex message interactions, it is widely used.

B. Mapping of Problem Diagrams to Sequence Diagrams

Sequence diagrams are used to describe the sending of messages between objects. It can intuitively convey the interaction of various parts of the system^[12]. For example, in the chains of causality in Figure 6, $f \rightarrow d \rightarrow b \rightarrow a \rightarrow e$ and $g \rightarrow c \rightarrow a \rightarrow e$, the requirement is used as a starting point and an ending point, and it is converted into a sequence diagram, as shown in Figures 7 and 8.

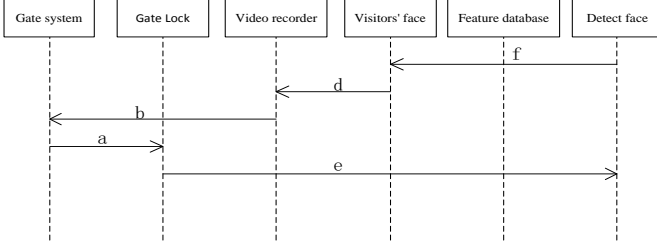


Fig. 7. A Sequence diagram

In these two figures, you can clearly see the triggering sequence of the sharing phenomena between various objects. The objects in the sequence diagram correspond to the domain and requirements of the problem diagram, while the objects in the sequence diagram can also be mapped to the processes in the CSP^[13]. Here is an example of a vending machine for the reader to understand the mapping relationship between the problem diagram and CSP, as follows:

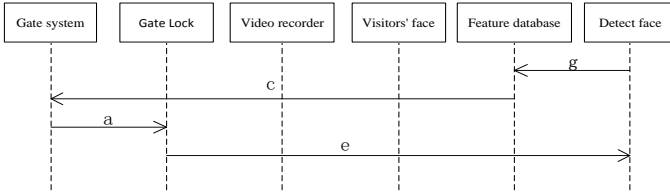


Fig. 8. A Sequence diagram

A vending machine (VM) receives a coin inserted by a customer (CUST) and automatically gives chocolate (choc) or coffee according to the customer's purchase request and choice.

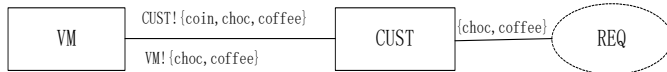


Fig. 9. The vending machine problem diagram

According to the problem description, the vending machine problem can be described as follows:

$$VM = \text{coin} \rightarrow (\text{choc} \rightarrow VM \mid \text{coffee} \rightarrow VM),$$

$$CUST = \text{coin} \rightarrow (\text{choc} \rightarrow CUST \mid \text{coffee} \rightarrow CUST),$$

which defines two process VM and CUST, where coin, choc and coffee are all events. From the CSP description and Figure 9, we can see that the domains in the problem frames can be mapped to the CSP processes, and the sharing phenomenon can be mapped to messages or events passed between processes. By studying this kind of mapping relationship, we can convert the sub-problems obtained through problem de-composition, that is, the causality chain, into a sequence diagram, and we have developed a tool which can automatically generate a CSP

script based on the objects in the sequence diagram and message passing.

C. CSP script generation based on sequence diagram

The object in the sequence diagram is regarded as a process in CSP. For example, in Figure 7, the process of detecting face begins first, and a new process is generated by the process and the event of the operation. The CSP is expressed as: Detect face =f-> Visitors' face. Similarly, the sequence diagram in Figure 7 can be described using CSP as the following script code:

```
Detect face =f-> Visitors' face
Visitors' face =d-> Video recorder
Video recorder =b-> Gate system
Gate system =a-> Gate lock
Gate lock =e->STOP
```

In summary, the process of converting a problem diagram into an algebraic expression is demonstrated, and this abstract algebraic symbol can be run by a specialized CSP verification tool, FDR^[17].

D. Requirements verification

A complex problem diagram is transformed into CSP scripts that run in the FDR tool. The requirements engineer can write code to verify that the requirements are satisfied or not. The support tools implemented in this paper integrate the FDR4 tool to automate the verification of CSP scripts. The FDR4 is a tool that can be used for model verification developed by scholars at Oxford University in the UK.

VI. RESEARCH ON REQUIREMENTS TESTING METHOD BASED ON PROBLEM FRAMES

Today's software systems are large and complex, and the task of software testing becomes error-prone and complicated. If the test task is clearly identified as early as possible, then the quality of requirements will be greatly improved.

A. Definition of Requirements Test

The requirements analysis model can be used as the guidance for documenting specifications in order to help the requirements analyst understand and help developers in system development. The requirements model can also be used as a test model^[14].

B. Extending Problem Frames

This study needs to add a causal relationship attribute to each domain to record the triggering relationships between the sharing phenomenon related to the domain. For example, if there are two shared phenomena a and b in the machine domain and phenomenon a triggers the occurrence of the b phenomenon, then $a \rightarrow b$ is recorded. We need to expand the domain constraint attributes to record one-to-many or many-to-many trigger conditions. For example, in an ATM system, where the depositor withdraws money less than the amount of the account, then the cash is ejected; if the withdrawal amount is greater than the amount of the account, then a display will show that it cannot be withdrawn. This study uses the syntax of

object-constraint language to record these domain-constraint attributes. For example, the record is as follows:

Account (balance), Withdrawal amount (amount)

Pre process: $balance > amount$ and $amount > 0$

Post process: $(balance = balance @ Pre - amount)$ and $balance > 0$

C. Generating test scenarios

The causality chain is a method of splitting complicated problem diagrams. The sub-problems obtained are a use case of the problem. Therefore, the sequence of causality chains is a test scenario. Testers can design test cases based on test scenarios generated by causality chains and constraints. Once a system fails, only the physical systems and computing systems related to the fault need to be tested. For example, the test trail $A \rightarrow B [@balance > amount \text{ and } amount > 0 @ (balance = balance - amount) \text{ and } balance > 0] \rightarrow C \rightarrow D$ that with OCL constraint description and test trail $A \rightarrow B \rightarrow C \rightarrow D$ that without OCL constraint description, where A, C, and D are physical components and B is control software. Assume that the C physical device has a failure, the tester only needs to test all the devices on the causal chain containing the shared phenomenon triggered by B. If the system is working properly and only the result of the operation is different from what is expected, the design of the test case can be based on the pre-constraint and post-constraint conditions of B, thus facilitating the testing and maintenance of the later system.

D. Generating Test Cases

The requirement references and constraints of PF are respectively represented by dashed lines without arrows and dashed lines with arrows. The dashed line with an arrow indicates that this requirement refers to the phenomenon in the problem domain. The dashed line with an arrow indicates that the requirement reference is a constraint reference, this requirement not only refers to the domain phenomenon, but also provides some desired relationships or behaviors that involve these relationships. In layman's terms, the former refers to a desired value or event, the latter defines a value or event to be obtained, like the input and output in the program. Therefore, we can develop the use case template shown in Table 3 below.

Table 3 Test Case Template

Test scenarios	Requirement reference	Requirement constraints

VII. IMPLEMENTATION OF SUPPORT TOOLS

The support tools developed in this study employ a client/server (C/S) and browser/server (B/S) hybrid architecture, which contains features such as good openness, easy expansion and transplantation^[15].

A. Software Architecture Diagram of C/S and B/S Mixed

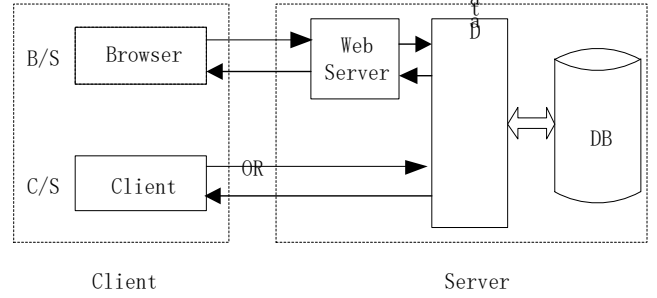


Fig.10. Support tool architecture diagram

B. Main functions of the tool

In addition to the basic function of drawing problem diagrams, this tool can also implement the function of checking the diagram for integrity and correctness. It can also automatically search and find causal chains and convert them into CSP scripts and verify the model automatically. Then test scenarios with constraints can be generated from these causal chains.

The tool can allow the problem diagram model to be saved in the XML format. Users can upload their own drawings on to the cloud server database. When modification is needed, the XML file can be opened from the database again. We first use the tool to draw the security gate control problem diagram (shown in Figure 11), and then the tool can automatically check the correctness and integrity of the diagram .

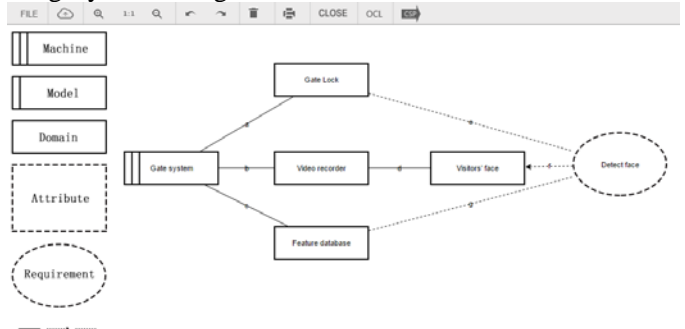


Fig.11. Drawing the right question diagram

After drawing the problem diagram, we click on the OCL button in the Tools menu to automatically verify that the problem diagram is complete and correct, as shown in Figure 12 below. If an incomplete or incorrect problem diagram is drawn, as shown in Figure 13, a domain has no name. The problem diagram is checked for completeness and correctness. The results are shown in Figure 14.

Once the diagram is verified to be complete, we can use the tool to find all the causal chains from the problem diagram (Figure 15).

rows	OCL rules	
1	Name of domains must be set	✓
2	Name of domains must be unique	✓
3	A context diagram has at least one machine domain	✓
4	A problem diagram/frame contains at least one requirement	✓
5	A requirement does not constrain a machine domain	✓
6	A controlled interface must be observed by at least one domain	✓
7	Each machine controls at least one interface	✓
8	The abbreviation of a domain must be set	✓
9	The abbreviation of a domain must be unique	✓

Fig.12. Screenshot of OCL check result

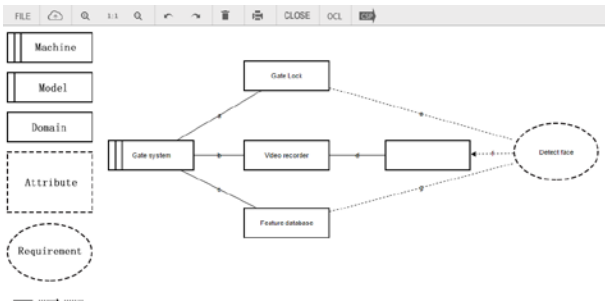


Fig.13. Draw error question diagram

rows	OCL rules	
1	Name of domains must be set	✗
2	Name of domains must be unique	✓
3	A context diagram has at least one machine domain	✓
4	A problem diagram/frame contains at least one requirement	✓
5	A requirement does not constrain a machine domain	✓
6	A controlled interface must be observed by at least one domain	✓
7	Each machine controls at least one interface	✓
8	The abbreviation of a domain must be set	✓
9	The abbreviation of a domain must be unique	✓

Fig.14. Screenshot of the check question diagram

Fig.15. The result of Finding causality chain

```

    Welcome to FDR 4.2.0 (6dac8661f79b49fb3b8343691adda12fe05318fc)
    FDR Version 4.2.0 copyright 2016 Oxford University Innovation Ltd. All Rights Reserved.
    License: Academic license for non-commercial use only
    Type :help for help
    Insulin injection Demo.csp> assert PATIENT[FD= REQ1
    Insulin injection Demo.csp> assert PATIENT[FD= REQ2
    Insulin injection Demo.csp>
  
```

Fig. 16. The result of FDR check

```

    causal chainf->d->b->a->e
    causal chainf->c->a->e
    test scenario:start->Visitors' face->Video recorder->Gate system[@record_syn = 1@open_cmd = 1]->Gate Lock->Detect face->end
    test scenario:start->Feature database->Gate system[@record_num = 0@open_cmd = 1]->Gate Lock->Detect face->end
  
```

Fig.17. generated test leads

Our tool can help system analysts to check if user requirements are satisfied or not by running the FDR4 tool, as shown in Figure 13. Based on those causality chains we can generate test scenarios, as shown in Figure 17.

VIII. CONCLUSIONS

In this paper, we provide a solution to the problem of automatically verifying CSP behaviors and user requirements, and searching and finding causal chains which helps de-compose a complex problem into sub-problems, and transform a problem diagram into a formal scripting language to verify whether the cyber-physical system design can satisfy end-to-end requirements^[16]. Test scenarios for the system can be generated based on the causality chains. Testers can derive test cases from these test scenarios with constraints to improve the test efficiency. This paper demonstrates the feasibility of the proposed method by applying the support tools we develop in the case study of a safety gate control problem. Our case study shows that our method contributes to reducing defects in the requirements analysis phase and increasing the success rate of software development projects.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for their valuable comments and suggestions. This work is partially supported by the National Natural Science Foundation of China (61262004), Guangxi “Bagui Scholar”

Teams for Innovation and Research, the Project of the Guangxi Key Lab of Multi-source Information Mining & Security (Director's grant 14-A-03-01, 15-A-03-01), Guangxi Collaborative Innovation Center of Multi-source Information Integration and Intelligent Processing, the Innovation Projects of Guangxi Graduate Education (No. XYCSZ2017066), (No. XYCSZ2018075), (No. XJGY201809), 2017 Guangxi Normal University Bilingual Course Project (No. A-0201-00-00013F).

REFERENCES

- [1] Wen J R, Mu-Qing W U, Jing-Fang S U. Cyber-physical System[J]. Acta Automatica Sinica, 2012, 38(4):507-517.
- [2] Dongfang Liang, Yuying Wang, Xingshe Zhou, et al. Simulation modeling method based on heterogeneous model fusion for CPS system[J]. Journal of Computer Science, 2012, 39(11):24-28.
- [3] Jackson M. Problem frames: analyzing and structuring software development problems[M]. Addison-Wesley Longman Publishing Co. Inc. 2000.
- [4] M. Jackson. System Behaviours and Problem frames: Concepts, Concerns and the Role of Formalisms in the Development of Cyber-physical Systems[M]. Dependable Software Systems Engineering, 2015:79-104.
- [5] Xiaohong Chen, Bin Yin, Zhi Jin. Demand Modeling Based on Problem Frames: A Method of Present System Guidance[J]. Journal of Software, 2011, 22(2):177-194.
- [6] Gogolla M. Object Constraint Language[J]. 2016.
- [7] Zefan Jiang, Linzhang Wang, Xuandong Li, et al. Test method based on UML sequence diagram[J]. Computer Science, 2004, 31(7):131-136.
- [8] Queralt A, Teniente E. Verification and Validation of UML Conceptual Schemas with OCL Constraints[J]. Acm Transactions on Software Engineering & Methodology, 2012, 21(2):1-41.
- [9] Li Z, Hall J G, Rapanotti L. On the systematic transformation of requirements to specifications[J]. Requirements Engineering, 2014, 19(4):397-419.
- [10] Jackson M. Where, Exactly, Is Software Development?[M]// Formal Methods at the Crossroads. From Panacea to Foundational Support. Springer Berlin Heidelberg, 2003:115-131.
- [11] C. A. R. Hoare. "Communicating Sequential Processes," The Origin of Concurrent Programming, Springer New York, pp 413-443, 2002.
- [12] Yuzhen Wang, Wei Dong, Huowang Chen. Automatic verification of UML sequence diagram[J]. Computer Engineering and Applications, 2003, 39(29):80-83.
- [13] Shuo Zhang, He Zi, Zhi. Utilizing the Sequence chart in problem frames[J]. Research Reports: se, 2013, 2013: 1-8.
- [14] Gao M, Zhong D, Lu M, et al. Research on test requirement modeling for software-intensive avionics and the tool implementation[C]// International affiliation, maintenance, safety study meeting session. 2007:6.D.2-1-6.D.2-10.
- [15] Chen X, Liu J L. Analysis and Comparison between the Structures of Client/Server and Browser/Server[J]. Journal of Chongqing Institute of Technology Management, 2000.
- [16] Seater R, Jackson D. Problem frames transformations: deriving specifications from requirements[C]// International Workshop on Advances and Applications of Problem frames. ACM, 2006:71-80.
- [17] Gibson-Robinson T, Armstrong P, Boulgakov A, Roscoe A.W, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, volume 8413, pages.187-201, 2014.