A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF INFORMATION SCIENCE
IN THE GRADUATE SCHOOL OF INFORMATION SCIENCE AND TECHNOLOGY
OF HOKKAIDO UNIVERSITY
平成 29 年度　博士論文

Scenario Verification for Proximity-Based Federation of Smart Objects Using Model Checking

モデル検査によるスマートオブジェクトの近接連携シナリオの形式検証

Reona Minoda

蓑田　玲緒奈

Large-Scale Knowledge Processing Laboratory,
Division of Computer Science,
Graduate School of Information Science and Technology,
Hokkaido University

北海道大学　大学院情報科学研究科
コンピュータサイエンス専攻　大規模知識処理研究室

February 2018

Abstract

This thesis proposes a verification method of ubiquitous computing scenarios using model checking. Model checking is one of the most popular formal verification approach and it is often used in various fields of industry. Model checking is conducted using a Kripke structure which is a formal state transition model. In this thesis, we present following three verification frameworks for ubiquitous computing scenarios.

First, we introduce a model called *Context Catalytic Reaction Network* (CCRN) to handle this federation model as a formal state transition model. We also give an algorithm to transform a CCRN into a Kripke structure and we conduct a case study of ubiquitous computing scenario verification, using this algorithm and the model checking. Finally, we discuss the advantages of our formal approach by showing the difficulties of our target problem experimentally.

Second, we propose an efficient verification framework of CCRN as a new application field of symbolic model checking. To do so, we illustrate a method how to transform a scenario written in CCRN into a symbolic model checking problem. We also show experimentally that our framework makes it possible to verify large scale ubiquitous com-

puting scenarios which could not be verified in realistic time by our previous method. Additionally, we show experimentally the usefulness of fault detections using bounded model checking in the same framework.

Third, we propose the method of reliability analysis of ubiquitous computing scenarios. In ubiquitous computing scenarios, various devices communicate with each other through wireless network, and this kind of communications sometimes break due to external interferences. To discuss the reliability in such situation, we introduce the notion of probability into CCRN, which is a description model of ubiquitous computing scenarios. This enables us to conduct quantitative analyses such as considerations of a trade-off between the reliability of ubiquitous computing scenarios and the costs which may be necessary for their implementations. To conduct a reliability analysis of ubiquitous computing scenarios, we use the technique of probabilistic model checking. We also evaluate our method experimentally by conducting a case study using a practical example assuming a museum.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Nowadays, we are surrounded with a lot of devices with computation and communication capabilities. These devices are called *smart objects* (SOs). SOs include PCs, smart phones, embedded computers, sensor devices and RFID tags. By embedding RFID tags in physical things such as mugs, food and medicine bottles, we can also treat them as SOs. The notion of ubiquitous computing assumes that a lot of these SOs surround enough around users. Here we use the term *federation* to denote the definition and execution of interoperation among resources that are accessible either through the Internet or through peer-to-peer ad hoc communication. For example, let us consider that there are a phone, a medicine bottle and food; and RFID tags are embedded in a medicine bottle and food. Imagine that this food and the medicine have a harmful effect when eaten together. If all these things are close to each

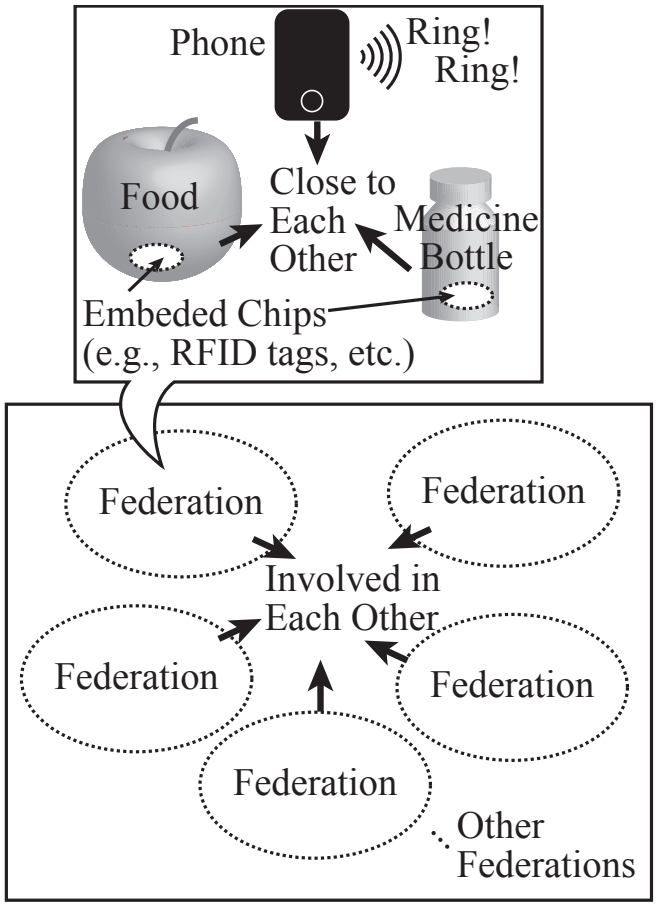


Figure 1.1 Example of Ubiquitous Computing Application Scenario

other, a phone rings to inform a user *to warn not to eat them together*. This phenomenon is a federation. Of course, we can also consider other federations related to other SOs and these federations may be involved in each other. And we call these federations "*ubiquitous computing application scenarios*" (see Fig. 1.1). As various kinds of devices and physical things can be treated as SOs thanks to technological innovation, our real world environment is now steadily laying the foundation for the concept of ubiquitous computing which Mark Weiser looked beyond [37].

Since Weiser proposed the notion of ubiquitous computing, it has been almost quarter of century. In the meantime, a lot of differ-

ent frameworks have been proposed to realize ubiquitous computing. However, regardless of specific research areas in ubiquitous computing, Yuzuru Tanaka pointed out that these researches typically only consider two types of application scenarios [35]. One is "*location transparent service continuance*" (i.e., a user can use a service wherever the user goes; See Fig. 1.2). The other one is "*context-aware service provision*" (i.e., a user can use different kinds of services depending on where the user is; See Fig. 1.3). Robin Milner thought that the lack of models for describing ubiquitous computing application scenarios caused to prevent from considering various types of application scenarios [29]. Besides, according to Milner, it is not possible to describe all concepts of ubiquitous computing by using a single model [29]. Milner argued that the hierarchical structure of models (Milner called it "*a tower of models*") was necessary. In a tower of models, each higher model should be implemented by a lower model.

Following the notion of a tower of models, Tanaka once proposed the basic idea for describing ubiquitous computing application scenarios using catalytic reaction network model [35]. This idea includes following three models:

- At the first (lowest) level, the port matching model describes the federation mechanism between two SOs in close proximity to each other.
- At the second (middle) level, the graph rewriting model describes the dynamic change of federation structures among SOs.
- At the third (highest) level, the catalytic reaction network model

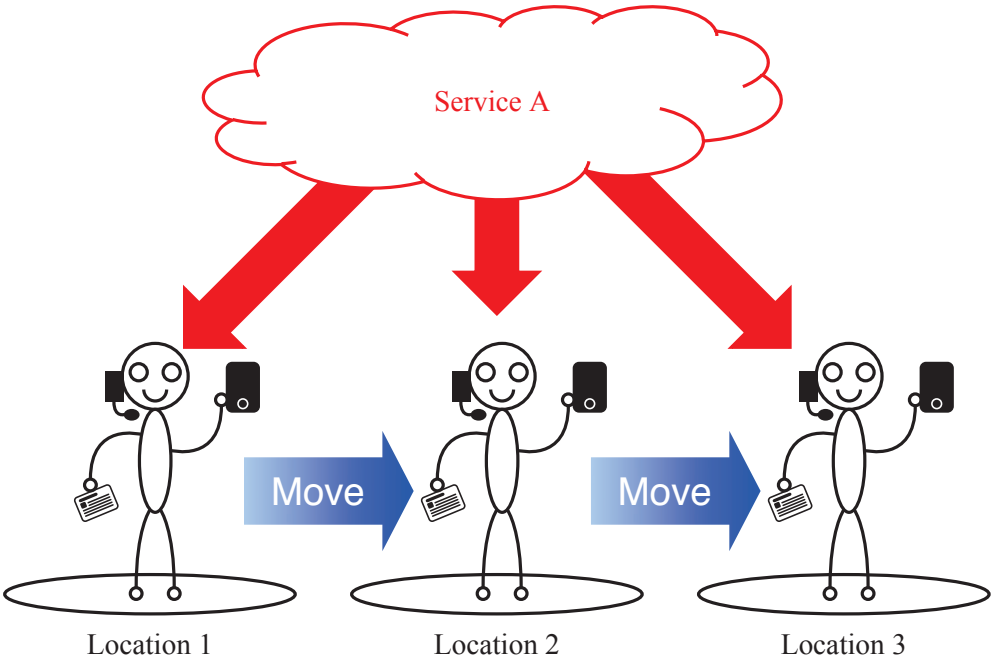


Figure 1.2 Example of location transparent service continuance

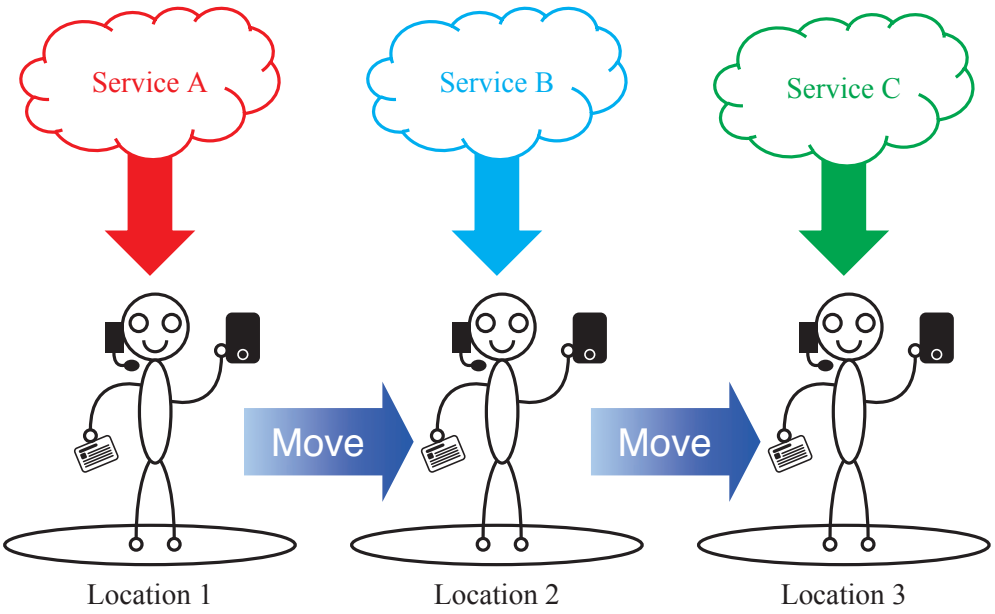


Figure 1.3 Example of context-aware service provision

describes application scenarios involving mutually related multiple federations.

Then, Julia and Tanaka brushed up these three models and established a concrete tower of models by proving that a higher model is surely implemented by a lower model [22]. Moreover, Julia's model implementation has error handling mechanisms assuming unexpected situations such as the connection failures between two SOs. Therefore we can focus on the catalytic reaction network model for describing application scenarios of ubiquitous computing.

However, there are still challenges of establishing the verification method of the catalytic reaction network model. So far, when we made a scenario using the catalytic reaction network model, we could not prove easily whether a particular federation would occur because federations of multiple devices are formed by proximity sensitive connections between SOs. So when we discuss a scenario using the catalytic reaction network, we also need to consider the proximity relations of SOs.

1.2 Our Contributions

In this thesis, we propose three frameworks for scenario verification of ubiquitous computing.

In chapter 3, we propose a verification method of device-federation model based on catalytic reaction network. Basically we transform a scenario into the well-known state-transition model such as Kripke structure. This enables us to apply existing model checking

verifiers. With this method, we can discuss the following things:

- Determining whether a property described in a temporal logic specification (e.g., a particular federation *finally* occured) is satisfied or not in the given scenario described by the catalytic reaction network model.
- Showing a counterexample if there is any case violating the property described above.

In a scenario using original catalytic reaction network model, there are so many proximity relations among SOs (n SOs would have 2^n proximity relations). This sometimes causes the state explosion problem in the model checking. We need to constrain the proximity relations in the original catalytic reaction network model. For this reason, we will first define the constrained model called "*Context Catalytic Reaction Network* (CCRN)." Then, we will propose the method to transform CCRN into the well-known state transition model such as a Kripke structure that can apply existing model checking verifiers.

In chapter 4, we propose a verification method using symbolic model checking to improve the scalability. Symbolic model checking is a one of model checking technique to verify very large scale of state transition systems [7]. We show the method how to take advantage of symbolic model checking techniques when we reduce ubiquitous computing scenario verification problems to model checking problems. We also show experimentally that our method can verify larger ubiquitous computing scenario verification problems compared to our previous naive approach. As a result, this method enables us to discuss more practical

ubiquitous computing scenario verification problems.

Additionally, if we establish a reduction method using a symbolic model checking approach, we can also take advantage of bounded model checking in the same framework [4]. Bounded model checking can detect counterexamples from a given model checking problem faster than symbolic model checking in most cases (Note that bounded model checking is not good at proving that there is no counterexample of a given model checking problem). Bounded model checking is widely used for the counterexample detection from model checking problems. In this chapter, we also evaluate experimentally how practical is bounded model checking for ubiquitous computing scenario verification problems.

In chapter 5, we show an approach to reliability analysis of ubiquitous computing scenarios by introducing a notion of probability to CCRN, which is a description model of ubiquitous computing scenarios. To analyze this kind of reliability, we use the technique of probabilistic model checking [25]. For example, ubiquitous computing scenarios are assumed that SOs typically communicate with each other by the wireless communication. This framework enables us to consider these communications which sometimes break due to various external causes. With this method, we discuss this kind of interference formally.

1.3 Related Works

1.3.1 Formal Verification of Cyber Physical Systems

Similarly to ubiquitous computing, a lot of devices such as sensors measure physical phenomena such as temperature, humidity, acceleration and so on, while actuators manipulate the physical world, like in automated robots. The combination of an electronic system with a physical process is called cyber physical system (CPS). In the field of CPS, Drechsler and Kühne use *timed automata* [2] as a state transition model to conduct formal verifications of given systems' properties [18][12].

Hasuo et al. introduce a metamathematical strategy to verify properties of CPS systems [17]. They generalize a methodology of verification for formal models and they try to establish meta-level verification theories which they can apply to no matter what kind of a model they verify. Hartmanns et al., one of their groups, extend timed automata by introducing a notion of probability to handle with a formal model for real-time systems with discrete probabilistic and non-deterministic choices [15].

For appropriate CPS reactions to the physical world, Sun et al. propose a framework to verify the consistency of a rule set for detecting physical world situations, which can be judged from information gathered from various sensors [34].

1.3.2 Context Inconsistency Detection

In the field of ambient computing, Xu and Cheung propose a method of context inconsistency detection [38]. This method detects inconsistencies from a series of gathered events such as "a user entered a room" and "the temperature of room is 30°C" by logical deduction. Unlike a formal verification, this method can be applied only after the system begins to work. Instead, a formal verification can find the failed cases from a given system *in advance*.

1.3.3 Reconfigurable hardware verification for Ubiquitous Systems

In the field of implementations for ubiquitous systems, Guellouz, et al. use probabilistic model checking to verify the behavior of devices [14]. These devices have reconfigurable function blocks, which is standardized as IEC 61499 and a part of each blocks behaves probabilistically. Guellouz, et al. analyzed this behavior by using probabilistic model checking.

1.4 Structure of this thesis

The rest of this thesis is organized as follows. Chapter 2 provides preliminaries of this thesis, such as basic definitions and notations. Using them, we define a CCRN and we propose the verification method of a CCRN in chapter 3. To improve the scalability of our method, we

propose a more efficient verification method of CCRN using symbolic model checking in chapter 4. In chapter 5, we propose the method of reliability analysis of probabilistic CCRN using probabilistic model checking. Finally, we summarize the results of this thesis in chapter 6.

Chapter 2

Preliminaries

In this section, we give definitions and notations which is necessary for this thesis. We also introduce an model of catalytic reaction network for proximity-based federation of SOs. Finally, we introduce formal verification methods which we use in later chapters.

2.1 Basic Definitions and Notation

Let X and Y be any two sets, we use $X \cup Y$, $X \cap Y$ and $X \setminus Y$ to denote the union, intersection and difference of X and Y respectively. For a set X, we denote its power set (i.e., all subsets) by 2^X and its cardinality by $|X|$. For a set X, we denote a set of k-elements subsets of X by $\binom{X}{k}$. For a family M of sets (i.e., a set of sets), we denote the union and the intersection of all sets in M by $\bigcup M$ and $\bigcap M$ respectively. Let X be a set,

we denote a set of all set partitions of X by $\mathfrak{P}(X)$. For example, given $X = \{1,2\}$, we have $\mathfrak{P}(X) = \{\{\{1\},\{2\}\},\ \{\{1,2\}\}\}$.

2.2 Catalytic Reaction Network

A catalytic reaction network is originally proposed by Stuart Kauffman in the field of biology to analyze protein metabolism [23]. Based on this model, Tanaka applied it to the field of ubiquitous computing as the way to describe an application scenario involving mutually related multiple federations among SOs [35]. In this thesis, we mean the latter by the term "catalytic reaction network".

A catalytic reaction network is a set of catalytic reactions. Each catalytic reaction takes input materials and transforms them into output materials. And each catalytic reaction has a catalyst which is called *context*. It may be also possible to include a catalyst in input materials. We call this kind of catalyst *stimulus*. A catalytic reaction is occurred when all required SOs are in the proximity of each other. We use the term "*scope*" to denote the inside of the proximity area (we assume a range of Wi-Fi radiowave, and so on). The scope of a SO *o* is represented as a set of SOs which are accessible from the SO *o*. Tanaka assumed that all scopes of the context and SOs involved in a catalytic reaction are considered [35].

However, as we mentioned in previous section, this causes the state explosion problem during the model checking. For this reason, in this thesis, we assume that only the scopes of contexts are considered instead. In other words, we consider that the catalytic reaction is oc-

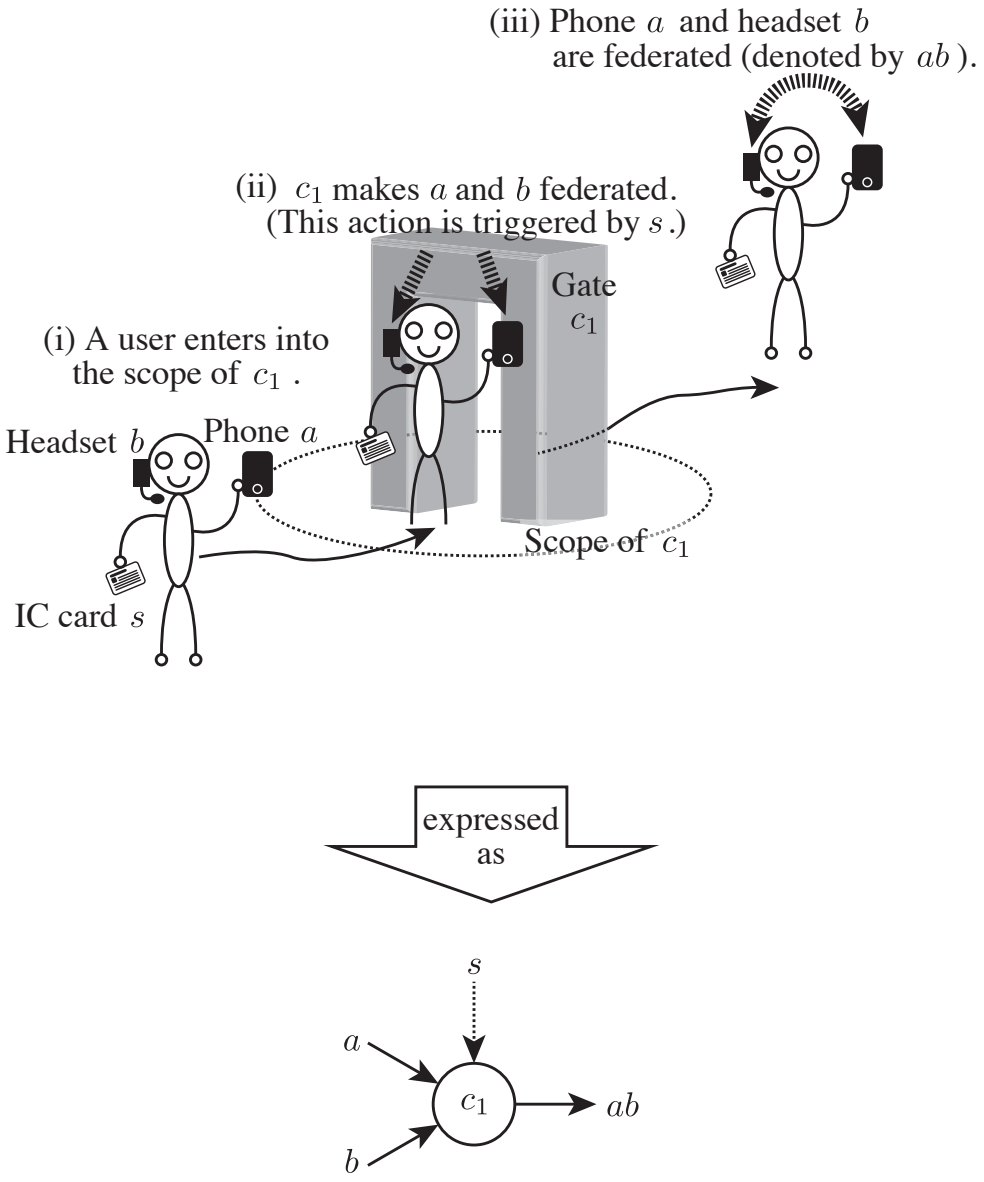


Figure 2.1 Example of a Catalytic Reaction

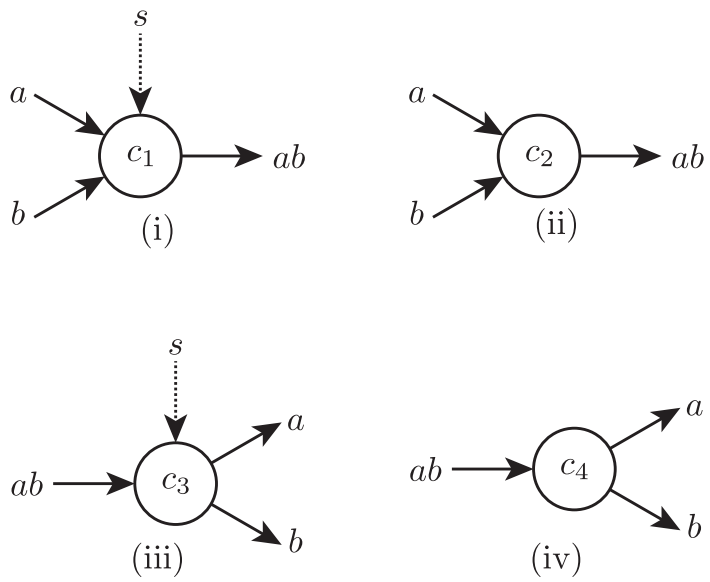


Figure 2.2 Four Types of a Catalytic Reactions

curred if all required SOs just enter into the scope of the corresponding context.

Figure 2.1 shows an example of single catalytic reaction. In this example, there is a gate c_1 regarded as a context and a user has three SOs i.e., a phone a, a headset b and an IC card s. If the user enters into the scope of c_1, c_1 makes a and b federated. This action is triggered by s. After that, phone a and headset b are federated. We denote federated SOs such as a and b by a concatenation of a and b, i.e., ab. During this process, c_1 and s work as catalysts. In particular, s is a stimulus in this reaction. We express this reaction as the right hand side diagram of Fig. 2.1.

In catalytic reaction networks, there are four types of catalytic reactions as we show in Fig. 2.2. We categorize these four types of reactions into two groups. One group is the *composition* reaction group (Fig. 2.2 (i) and (ii)), the other group is the *decomposition* reaction group (i.e.,

Fig. 2.2 (iii) and (iv)). A catalytic reaction of Fig. 2.1 is a type (i) catalytic reaction. We also consider the catalytic reaction without a stimulus such as Fig. 2.2 (ii). In type (ii), if a user who has SO a and SO b enters into the scope of context c_2, c_2 makes a and b federated *without a stimulus*. In a similar way, we consider the decomposition reactions such as Fig. 2.2 (iii) and (iv). In type (iii), if a user who has two SOs that are federated into ab enters into the scope of context c_3, c_3 decomposes these SOs ab into a and b triggered by SO s. Type (iv) is a decomposition reaction without a stimulus.

The output SO of a reaction may promote other reactions as a stimulus or become an input SO of other reactions. In this way, catalytic reactions form a network of reactions.

Now we define a catalytic reaction network formally. First, let O be a set of SOs, we give a definition of a federated SO o_f by $o_f \in 2^O \setminus \{\emptyset\}$ where $|o_f| > 1$. If $|o_f| = 1$, we treat o_f as a single SO. Next, we define a catalytic reaction as follows:

Definition 2.2.1 (Catalytic Reaction) Let O and C be a set of SOs and a set of contexts respectively, a catalytic reaction is defined as a tuple (c, M, N) where

- $c \in C$, $M \subseteq 2^O \setminus \emptyset$, $N \subseteq 2^O \setminus \emptyset$
- $\forall o_f \forall o'_f \in M.(o_f \neq o'_f \rightarrow o_f \cap o'_f = \emptyset)$
- $\forall o_f \forall o'_f \in N.(o_f \neq o'_f \rightarrow o_f \cap o'_f = \emptyset)$
- $\bigcup M = \bigcup N$, and

- $(|M \cap N| + 1 = |N|, |M| > |N|) \vee$
  $(|M \cap N| + 1 = |M|, |M| < |N|)$ $(*)$

The former of the last condition (signed by $(*)$) and the latter of the last condition correspond to a necessary condition for composition reaction and decomposition reaction respectively.

We give some examples of catalytic reactions. Given $C = \{c_1, c_3\}, O = \{a, b, s\}$, a catalytic reaction of Fig. 2.2 (i) and (iii) can be defined by
$(c_1, \{\{a\}, \{b\}, \{s\}\}, \{\{a, b\}, \{s\}\})$ and
$(c_3, \{\{a, b\}, \{s\}\}, \{\{a\}, \{b\}, \{s\}\})$ respectively.

Finally, a catalytic reaction network is defined as follows:

Definition 2.2.2 (Catalytic Reaction Network) A catalytic reaction network is a set of catalytic reactions.

2.3 Model Checking

A model checking is a method to verify a property of a state transition system. It was proposed by Clarke and Emerson [10]; separately Quielle and Sifakis proposed basically the same method [32]. It has been often used in various fields, which ranges from electronic-circuit-design verification [6] to secure-network-protocol (e.g., Secure Sockets Layer (SSL) protocol [13]) design verification [30]. In the model checking, it is typically assumed to use a Kripke structure as a state transition system. The property of a Kripke structure is described by a modal logic. There

are two kind of commonly used modal logics such as *computational tree logic* (CTL) and *linear temporal logic* (LTL).

2.3.1 Kripke Structure

Before we look on the detail of a model checking, we give the definition of a Kripke structure [24] which is necessary for a modal logic and a model checking.

Definition 2.3.1 (Kripke Structure) Let AP be a set of atomic propositions, *a Kripke structrue M* is a tuple (S, I, R, L), where

- S is a finite set of states,
- $I \subseteq S$ is a set of initial states,
- $R \subseteq S \times S$ is a set of transition relation such that R is left-total, i.e., $\forall s \in S,\ \exists s' \in S$ such that $(s, s') \in R$, and
- $L : S \to 2^{AP}$ is a labeling function.

2.3.2 Computational Tree Logic

CTL was proposed by Clarke et al. for the purpose of analysis and design of concurrent programs. [10] First, we give a definition of CTL syntax.

Definition 2.3.2（CTL Syntax） Let AP be a set of atomic propositions, a CTL formula ϕ is defined by the following syntax recursively.

$$\begin{aligned}\phi ::= &\top \mid \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \\ &\mathbf{EX}\phi \mid \mathbf{EG}\phi \mid \mathbf{EF}\phi \mid \mathbf{E}(\phi \mathbf{U} \phi) \mid \\ &\mathbf{AX}\phi \mid \mathbf{AG}\phi \mid \mathbf{AF}\phi \mid \mathbf{A}(\phi \mathbf{U} \phi)\end{aligned}$$

where $p \in AP$.

These right-hand terms denote basic logical terms i.e., true, false, p, negation and disjunction, quantifiers over paths i.e., all and exist, and path-specific operators i.e., next time, always, eventually and until respectively.

Next, we focus on the semantics of CTL. The semantics of CTL are defined as satisfaction relations (denoted by $\models$) of a pair $\langle M, s\rangle$ of a Kripke structure $M = (S, I, R, L)$ and its present state $s \in S$. We enumerate the semantics of CTL as follows:

- $M, s \models \top$ (true is always satisfied)
- $M, s \not\models \bot$ (false is never satisfied)
- $(M, s \models p)$ iff $(p \in L(s))$ (atomic propositions are satisfied when they are members of the present state's labels)

And there are two CTL semantics of boolean combinations as follows:

- $(M, s \models \neg\phi)$ iff $(M, s \not\models \phi)$
- $(M, s \models \phi \vee \psi)$ iff $[(M, s \models \phi) \vee (M, s \models \psi)]$

And there are eight CTL semantics of combinations of quantifiers over paths and path-specific operators as follows:

- $(M, s \models \mathbf{EX}\phi)$ iff $[\exists s' \in S.((s, s') \in R \wedge (M, s' \models \phi))]$.
- $(M, s \models \mathbf{AX}\phi)$ iff $[\forall s' \in S.((s, s') \in R \wedge (M, s' \models \phi))]$.
- $(M, s \models \mathbf{EG}\phi)$ iff there exists an infinite transition-path $s_0, s_1, s_2, \ldots$ generated from R such that $s_0 = s$ and for all $k \geq 0$, $M, s_k \models \phi$ is satisfied.
- $(M, s \models \mathbf{AG}\phi)$ iff $M, s' \models \phi$ is satisfied for all $s' \in S$ which is reachable from s by R.
- $(M, s \models \mathbf{EF}\phi)$ iff there exists $s' \in S$ which is reachable from s by R and $M, s' \models \phi$ is satisfied.
- $(M, s \models \mathbf{AF}\phi)$ iff for all infinite transition-path $s_0, s_1, s_2, \ldots$ generated from R such that $s_0 = s$, there exists $k \geq 0$ such that $M, s_k \models \phi$.
- $(M, s \models \mathbf{E}(\phi \mathbf{U} \psi))$ iff there exists an infinite transition-path $s_0, s_1, s_2, \ldots$ generated from R such that $s_0 = s$, it is satisfied that $\big[(\exists i.(M, s_i \models \psi)) \wedge (\forall j < i.(M, s_j \models \phi))\big]$.
- $(M, s \models \mathbf{A}(\phi \mathbf{U} \psi))$ iff for all infinite transition-paths $s_0, s_1, s_2, \ldots$ generated from R such that $s_0 = s$, it is satisfied that $\big[(\exists i.(M, s_i \models \psi)) \wedge (\forall j < i.(M, s_j \models \phi))\big]$.

2.3.3 Linear Temporal Logic

LTL was first proposed for the formal verification of computer programs by Amir Pnueil in 1977 [31]. First, we give a definition of LTL syntax.

Definition 2.3.3（LTL Syntax） Let AP be a set of atomic propositions, a LTL formula ϕ is defined by the following syntax recursively.

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \mathbf{X}\,\phi \mid \mathbf{G}\,\phi \mid \mathbf{F}\,\phi \mid \phi\,\mathbf{U}\,\phi$$

where $p \in AP$.

These right-hand terms denote true, false, p, negation, disjunction, next time, always, eventually and until respectively.

Next, we define a transition path π of a Kripke structure M.

Definition 2.3.4（Transition Path） Let M be a Kripke structure, $\pi = (\pi_0, \pi_1, \pi_2, \dots)$ is a transition path in M if it respects M's transition relation, i.e., $\forall i.(\pi_i, \pi_{i+1}) \in R$. π^i denotes π's ith suffix, i.e., $\pi^i = (\pi_i, \pi_{i+1}, \pi_{i+2}, \dots)$.

Also it can be shown that

$$\begin{aligned}(\pi^i)^j &= (\pi_i, \pi_{i+1}, \pi_{i+2}, \dots)^j \\ &= (\pi_{i+j}, \pi_{i+j+1}, \pi_{i+j+2}, \dots) \\ &= \pi^{i+j}.\end{aligned}$$

Now we focus on the semantics of LTL. First, we define the binary satisfaction relation, denoted by $\models$, for LTL formulae. This sat-

isfaction is with respect to a pair – $\langle M, \pi \rangle$, a Kripke structure and a transition path. Then we enumerate LTL semantics as follows:

- $M, \pi \models \top$ (true is always satisfied)
- $M, \pi \not\models \bot$ (false is never satisfied)
- $(M, \pi \models p)$ iff $(p \in L(\pi_0))$ (atomic propositions are satisfied when they are members of the path's first element's labels)

And there are two LTL semantics of boolean combinations as follows:

- $(M, \pi \models \neg\phi)$ iff $(M, \pi \not\models \phi)$
- $(M, \pi \models \phi \vee \psi)$ iff $[(M, \pi \models \phi) \vee (M, \pi \models \psi)]$

And there are four LTL semantics of temporal operators as follows:

- $(M, \pi \models \mathbf{X}\,\phi)$ iff $(M, \pi^1 \models \phi)$
- $(M, \pi \models \mathbf{F}\,\phi)$ iff $\left[\exists i.(M, \pi^i \models \phi)\right]$
- $(M, \pi \models \mathbf{G}\,\phi)$ iff $\left[\forall i.(M, \pi^i \models \phi)\right]$
- $(M, \pi \models \phi\ \mathbf{U}\ \psi)$ iff $\left[(\exists i.(M, \pi^i \models \psi)) \wedge (\forall j < i.(M, \pi^j \models \phi))\right]$

2.3.4 Model Checking Problem

Intuitively saying, a model checking problem is to judge whether a given Kripke structure M satisfies a given property described in a modal logic formula ϕ. Definitions of model checking problems depend on that they are given CTL or LTL as a property. Model checking problems of CTL and LTL are formally stated as follows respectively.

Definition 2.3.5（Model Checking Problem of CTL） Given a desired property decribed by a CTL formula ϕ and a Kripke structure $M = (S, I, R, L)$, a model checking problem of CTL is a decision problem whether the following formula

$$M, s \models \phi$$

is satisfied or not where $s \in I$ is a initial state.

Definition 2.3.6（Model Checking Problem of LTL） Given a desired property described by a LTL formula ϕ and a Kripke structure $M = (S, I, R, L)$, a model checking problem of LTL is a decision problem whether the following formula

$$\forall \pi.(M, \pi \models \phi)$$

is satisfied or not where $\pi_0 \in I$. Note that a set $\{\pi \mid (M, \pi \not\models \phi)\}$ is particularly called a set of *counterexamples*.

It is known that a model checking problem can be reduced to a graph search if M has finite states. Also it is known that a model checking

problem is a NP-complete problem if a given property is represented by a restricted LTL formula which contains temporal operators **G** and **F** only. If a given property is represented by a general LTL formula, this problem becomes a PSPACE-complete problem [33].

There are several implementations of the model checking verifier such as **S**imple **P**romela **IN**terpreter (SPIN) [20], **L**abel **T**ransition **S**ystem **A**nalyzer (LTSA) [27], **New S**ymbolic **M**odel **V**erifier version **2** (NuSMV2) [9] and so on. In this thesis, we use a model checking verifier NuSMV2.

2.3.5 Symbolic Model Checking

If the number of states in a given Kripke structure M becomes bigger, the cost of "a graph search" increases exponentially. To ease this problem, McMillan et al. proposed *symbolic model checking* [28][7]. Symbolic model checking does not hold explicitly states and transitions of a given Kripke structure. Instead, it holds symbolically them as Boolean formulae. It uses a binary decision diagram (BDD) [5] to store these Boolean formulae. By this, it can verify efficiently a very large Kripke structure such as 10^{20} states and more.

In symbolic model checking, a set of states S in a Kripke structure are represented as a following Boolean function $S(\boldsymbol{s})$ using a variable vector $\boldsymbol{s}$.

$$S(\boldsymbol{s}) = \begin{cases} \text{True} & \text{if } \boldsymbol{s} \in S \\ \text{False} & \text{otherwise} \end{cases} \tag{2.1}$$

A set of initial states $I \subseteq S$ also can be represented as the same way. To represent transition from $\boldsymbol{s} \in S$ to $\boldsymbol{s'} \in S$ and its relations $R \subseteq S \times S$, we

use following Boolean function $T(\boldsymbol{s}, \boldsymbol{s'})$.

$$T(\boldsymbol{s}, \boldsymbol{s}') = \begin{cases} \text{True} & \text{if } (\boldsymbol{s}, \boldsymbol{s}') \in R \\ \text{False} & \text{otherwise} \end{cases} \tag{2.2}$$

$S(\boldsymbol{s})$ and $T(\boldsymbol{s}, \boldsymbol{s'})$ are actually held as BDDs in symbolic model checking verifiers but we do not need give BDDs to them directly. Instead, we give a variable vector $\boldsymbol{s}$ representing states and Boolean functions of $S(\boldsymbol{s})$, $I(\boldsymbol{s})$ and $T(\boldsymbol{s}, \boldsymbol{s'})$ to symbolic model checking verifiers. One of famous implementations of symbolic model checking is **New** **S**ymbolic **M**odel **V**erifier version **2** (NuSMV2) [9].

2.3.6 Bounded Model Checking

Using above the variable vector $\boldsymbol{s}$, Boolean functions $I(\boldsymbol{s})$ and $T(\boldsymbol{s}, \boldsymbol{s'})$, we can introduce *bounded model checking* proposed by Biere et al. [4]. Bounded model checking is a kind of symbolic model checking. Most remarkable thing is that it reduces a model checking problem to a satisfiability problem (SAT) which can be solved by various SAT solvers. In recent days, a lot of SAT solvers are developed day by day and these SAT solvers' capability of solving SATs increases very rapidly [21].

Basically, to conduct bounded model checking, we judge whether following Boolean function

$$I(\boldsymbol{s}_0) \wedge \left(\bigwedge_{i=0}^{k-1} T(\boldsymbol{s}_i, \boldsymbol{s}_{i+1}) \right) \wedge \neg p(\boldsymbol{s}_0, \ldots, \boldsymbol{s}_k) \tag{2.3}$$

is satisfiable or not. Note that Boolean function $p(\boldsymbol{s}_0, \ldots, \boldsymbol{s}_k)$ is generated from a given LTL formula ϕ by a bounded model checking method, and k is the number of steps from initial states to verify a property ϕ.

This is a reason why this method is called "bounded" model checking. If above Boolean formula is satisfiable, the corresponding assignments $s_0, \ldots, s_k$ represents a counterexample of ϕ. Otherwise it means that there is no counterexample of ϕ at least k-steps from initial states. Most of modern SAT solvers are good at finding satisfiable assignments of a given Boolean function rather than proving non-existence of satisfiable assignments of the function. This means that bounded model checking is suitable for detecting counterexamples of LTL formulae. In fact, NuSMV2 can also conduct bounded model checking.

2.3.7 Probabilistic Model Checking

Probabilistic model checking is another method of reliability analysis for given Kripke structure. Probabilistic model checking is also called as stochastic model checking; and was introduced by Vardi [36]. Like conventional model checking, probabilistic model checking is also applied to industrial applications including the verification of IEEE 1394 (also known as Firewire) [1], which is a interface standard of a serial bus [26]. In original model checking, the reachability between two states of Kripke structure is defined as the existence of the directed edge between these states. Instead, in probabilistic model checking, the reachability between two states of Kripke structure is represented as a probability. This probability of the reachability is normalized with respect to each states as follows

$$\sum_{s',(s,s')\in R} P(s' \mid s) = 1 \text{ for all } s \in S. \tag{2.4}$$

Implementations of probabilistic model checkers include Prob-

Verus [16], E $\vdash$ MC [19] and Probabilistic Symbolic Model Checker (PRISM) [11][25]. In this thesis, we use PRISM to conduct a case study in chapter 5.

Probabilistic LTL

There are several logics such as pCTL and pCTL* [3] to express properties of probabilistic state transition models. In this thesis, we use probabilistic LTL which is used in PRISM.

Probabilistic LTL is extended property description language of LTL for probabilistic model checking. In probabilistic LTL, temporal operators **G** and **F** has an additional bound parameter k denoted by $\mathbf{G}_{\leq k}$ and $\mathbf{G}_{\leq k}$. These temporal operators have following semantics:

- $(M, \pi \models \mathbf{F}_{\leq k}\, \phi)$ iff $\left[\exists i \leq k.(M, \pi^i \models \phi)\right]$
- $(M, \pi \models \mathbf{G}_{\leq k}\, \phi)$ iff $\left[\forall i \leq k.(M, \pi^i \models \phi)\right]$

To discuss the probability of the transition path, probabilistic LTL also has quantitative operator $\mathbf{P}_{=?}$. Given a LTL property ϕ, $\mathbf{P}_{=?}\ \phi$ evaluates the existence probability of transition paths which satisfies the LTL property ϕ.

Probabilistic Model Checking Problem

Intuitively saying, a probabilistic model checking problem evaluates the existence probability of transition paths with length k which satisfies the property ϕ described by probabilistic LTL. This transition

assumes discrete-time Markov chain. A probabilistic model checking problem is defined as follows:

Definition 2.3.7 Given a Kripke structure M, $\tau_{s,s'} = P(s' \mid s)$, a probabilistic LTL ϕ and a length of bound k, a probabilistic model checking problem evaluates the following probability:

$$\sum_{\forall \pi.(M,\pi \models \phi \wedge |\pi|=k)} \prod_{t=1}^{k} P(\pi_t \mid \pi_{t-1}) \tag{2.5}$$

2.4 Summary

In this chapter, we introduced enabling technologies and concepts for ubiquitous computing scenario verification. Catalytic reaction network itself is originally from a bio-inspired framework. On the other hand, model checking is a formal method. From now on, we span a gap between ubiquitous computing related concepts and formal methods of model checking with our frameworks mentioned in the following chapters.

Chapter 3

Scenario Verification for Ubiquitous Computing using Model Checking

This chapter proposes a verification method of this model through model checking. Model checking is one of the most popular formal verification approach and it is often used in various fields of industry. Model checking is conducted using a Kripke structure which is a formal state transition model. We introduce a model called *Context Catalytic Reaction Network* (CCRN) to handle this federation model as a formal state transition model. We also give an algorithm to transform a CCRN into a Kripke structure and we conduct a case study of ubiquitous computing scenario verification, using this algorithm and the model checking. Finally, we discuss the ad-

vantages of our formal approach by showing the difficulties of our target problem experimentally.

3.1 Introduction

In this chapter, we propose a verification method of device-federation model based on catalytic reaction network. Basically we transform a scenario into the well-known state-transition model such as Kripke structure. This enables us to apply existing model checking verifiers. With this method, we can discuss the following things:

- Determining whether a property described in a temporal logic specification (e.g., a particular federation *finally* occured) is satisfied or not in the given scenario described by the catalytic reaction network model.
- Showing a counterexample if there is any case violating the property described above.

In a scenario using original catalytic reaction network model, there are so many proximity relations among SOs (n SOs would have 2^n proximity relations). This sometimes causes the state explosion problem in the model checking. We need to constrain the proximity relations in the original catalytic reaction network model. For this reason, we will first define the constrained model called "*Context Catalytic Reaction Network* (CCRN)." Then, we will propose the method to transform CCRN

into the well-known state transition model such as a Kripke structure that can apply existing model checking verifiers.

3.2 Context Catalytic Reaction Network

This section introduces a segment graph and a CCRN.

3.2.1 Segment Graph

As we discussed in previous section, a catalytic reaction is occurred when required SOs enter into the scope of the corresponding context. To analyze the property of a given catalytic reaction network as a state transition system, it is necessary to formalize the movement of SOs. For example, in Fig. 3.1 (i), there are contexts c_1 and c_2 and these scopes have an *overlap*. A user can walk around the path $\alpha\beta$ shown in Fig. 3.1 (i). This situation can be represented as a segment graph shown in Fig. 3.1 (ii). We consider that the user walk around this segment graph and the user is always located at one of the nodes of this segment graph. Each node of a segment graph has a corresponding set of scopes of contexts. In this way, the given situation like Fig. 3.1 (i) including overlaps of scopes of contexts can be represented as a discrete structure. Now we define a segment graph as follows.

Definition 3.2.1 (Segment Graph) Let C be a set of contexts, a segment graph G is a tuple (S, E, F), where

- S is a finite set of segments,

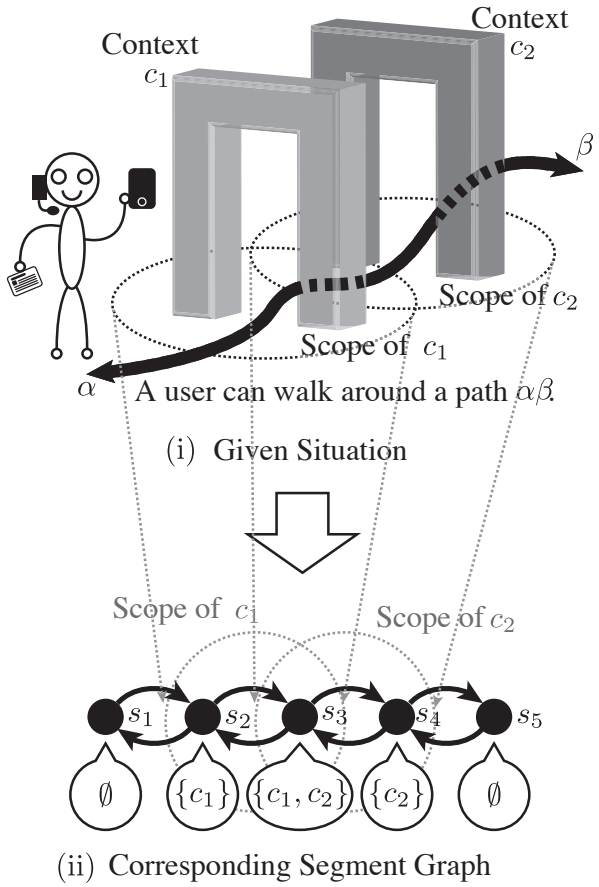


Figure 3.1 Example of Segment Graph

- $E \subseteq S \times S$ is a set of directed edges between two segments, and
- $F : S \to 2^C$ is a function returning scopes of contexts at corresponding segments.

3.2.2 Context Catalytic Reaction Network

A CCRN is a discrete structure of a situation involving SOs in a catalytic reaction network. A CCRN is defined as a conbination of a segment graph and a catalytic reaction network.

Definition 3.2.2（Context Catalytic Reaction Network） *A context catalytic reaction network* $(CCRN)$ is a tuple $(O, C, R, G, L_{\mathrm{FIX}}, l_0)$, where

- O is a set of SOs,
- C is a set of contexts,
- R is a set of catalytic reactions,
- G is a segment graph (S, E, F),
- $L_{\mathrm{FIX}} \subseteq O \times S$ is the locations of fixed SOs, and
- $l_0 \in S$ is the initial segment locating mobile SOs (mobile SOs can be represented as $O \setminus \{o \in O \mid \exists s \in S.((o, s) \in L_{\mathrm{FIX}})\}$).

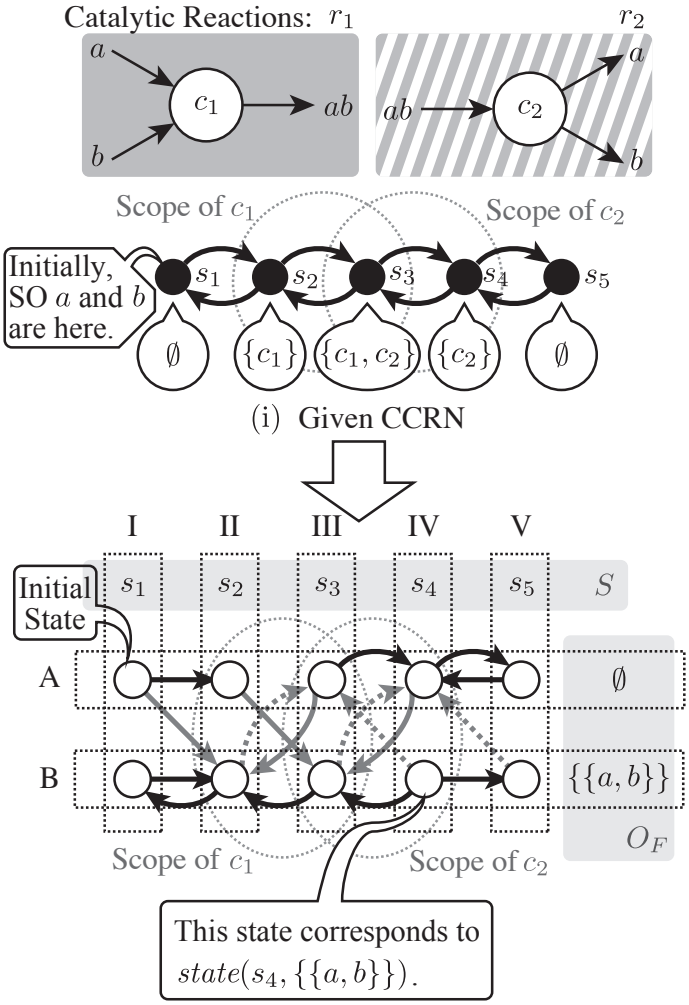
3.3 Verification Method of a CCRN

In this section, we propose a verification method of a CCRN. Before discussing the details of the method, we assume that all mobile SOs are carried together (by a single user). A state of a CCRN can be represented as a combination of the location of mobile SOs (e.g., mobile SOs are located at segment s) and the presence of federated SOs (e.g., federated SOs o_f and o'_f are existing) and we regard these two kind of facts as atomic propositions. We use the following atomic propositions (AP):

- $loc_{O_{MOB}}(s)$: mobile SOs are located at segment s
- $fed(o_f)$: federated SOs o_f is existing

While mobile SOs move around a segment graph, more than one federated SOs may appear. For example, federated SOs $\{a, b\}$ and $\{c, d\}$ may appear at the same time. For that reason, we define a single state of the presence of federated SOs as the subset of 2^O (e.g., $\{\{a, b\}, \{c, d\}\}$ is a subset of $2^{\{a,b,c,d\}}$). But each SO can not be a part of more than one federated SOs. For example, we do not permit federated SOs like $\{a, b\}$ and $\{b, c\}$ are presented at the same time because SO b is a part of both of these two federated SOs. Considering this constraint, a set of states of presence of federated SOs can be represented as $O_F = \{\emptyset\} \cup \{o_F \mid o_F \subseteq 2^O, \forall o_f, o'_f \in o_F.(o_f \neq o'_f \rightarrow o_f \cap o'_f = \emptyset), \forall o_f \in o_F.(|o_f| > 1)\}$. Finally, we represent a state of a CCRN as $state(s, o_F)$ where s is the segment at which mobile SOs are located and o_F is the set of federated SOs. For example, $state(s_0, \{\{a, b\}, \{c, d\}\})$ means mobile SOs are located at segment s_0 and federated SOs $\{a, b\}$ and $\{c, d\}$ are existing.

Using the above representation of a state of a CCRN and atomic propositions, we conduct verification of a CCRN by constructing a Kripke structure from a given CCRN. For example, let a set of SO O be $\{a, b\}$ and given a catalytic reaction network and a segment graph such as Fig. 3.2 (i). In this case, O_F is a set $\{\ \emptyset, \{\{a, b\}\}\ \}$ and a set of segments is $S = \{s_1, \ldots, s_5\}$. So we consider the product of O_F and S as the set of states in Kripke structure. White colored nodes in Fig. 3.2 (ii) are states in Kripke structure. States enclosed in dotted rectangle I ... V correspond to the element $s_1 \ldots s_5 \in S$ repectively. States enclosed in dotted rectangle A correspond to the element $\emptyset \in O_F$. Similarly, states enclosed in dotted rectangle B correspond to the element $\{\{a, b\}\} \in O_F$. If we consider the state of Kripke structure in this way, we can treat the movement of SOs without any catalytic reactions as transitions between



(ii) Constructed Kripke Structure by Proposed Method

Figure 3.2 Mechanism of the Algorithm

two states which both are in either group A or B. We can also treat the movement of SOs with any catalytic reactions as transition between two states which are in different groups. In this example, if no catalytic reaction is occured during SOs' movement, corresponding transition is defined as $(state(s, o_f), state(s', o_f))$ where $s, s' \in S$, $(s, s') \in E$ and $o_f \in O_F$ and if any catalytic reactions are occurred during SOs' movement, corresponding transition is defined as $(state(s, o_f), state(s', o'_f))$

where $s, s' \in S$, $(s, s') \in E$, $o_f, o'_f \in O_F$ and $o_f \neq o'_f$. In Fig. 3.2 (ii), black lined transitions and gray lined transitions and gray dotted lined transitions correspond to SOs' movement without any catalytic reactions, SOs' movement with catalytic reaction r_1 and SOs' movement with catalytic reaction r_2 respectively. Generalizing about this mechanism, here we give an algorithm in Fig. 3.3 to construct a Kripke structure from a given CCRN.

After constructing a Kripke structure from a CCRN, now we describe properties of a CCRN by LTL formulae. We enumerate examples of LTL formulae:

- $\mathbf{G}(\neg \mathit{fed}(o_f) \rightarrow \mathbf{F}(\mathit{fed}(o_f)))$
  Informally and intuitively saying, federated SOs o_f finally exists if o_f does not exist at the beginning and this always happens.

- $\mathbf{G}((\neg \mathit{fed}(o_f) \rightarrow \mathbf{F}(\mathit{fed}(o_f))) \vee (\neg \mathit{fed}(o'_f) \rightarrow \mathbf{F}(\mathit{fed}(o'_f))))$
  This means federated SOs o_f finally exists if o_f does not exist at the beginning. Similarly, federated SOs o'_f finally exists if o'_f does not exist at the beginning. At least one of these phenomena always happens.

Finally, we conduct the model checking, giving a Kripke structure and LTL formulae. This can be done by various implementations of model checking verifiers which we introduced in previous section.

Input: CCRN $(O, C, R, (S, E, F), L_{\text{FIX}}, l_0)$

Output: Kripke Structure $(\mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L})$

Initialization :

$O_{MOB} = O \setminus \{o \in O \mid \exists s \in S.((o, s) \in L_{FIX})\}$

$O_F = \{\emptyset\} \cup \{o_F \mid o_F \subseteq 2^O, \forall o_f, o'_f \in o_F.(o_f \neq o'_f \rightarrow o_f \cap o'_f = \emptyset), \forall o_f \in o_F.(|o_f| > 1)\}$

$AP = \{loc_{O_{MOB}}(s) \mid s \in S\} \cup \{fed(o_f) \mid o_f \in o_F, o_F \in O_F\}$

$\mathcal{S} = \{state(s, o_F) \mid s \in S, o_F \in O_F\}$

$\mathcal{I} = state(l_0, \emptyset)$

$\mathcal{R} = \emptyset$

Loop Process :

for each $o_F \in O_F$ **do**

  for each $s \in S$ **do**

    $\mathcal{L}(state(s, o_F)) = \{loc_{O_{MOB}}(s)\} \cup \{fed(o_f) \mid o_f \in o_F\}$

    $S' = \{s' \mid (s, s') \in E\}$

    for each $s' \in S'$ **do**

      $R' = \{(c, M, N) \in R \mid c \in F(s'), \{o_f \in M \setminus N \mid |o_f| > 1\} \subseteq o_F, O(c) \supseteq \bigcup M\}$

      where $O(c \in C) = O_{MOB} \cup \{o \in O \mid \exists s'' \in S.(c \in F(s''), (o, s'') \in L_{FIX})\}$

      if $R' \neq \emptyset$ **then**

        for each $(c, M, N) \in R'$ **do**

          choose $o'_F \in O_F$ s.t. $o_F \setminus o'_F = \{o_f \in M \setminus N \mid |o_f| > 1\}$, $o'_F \setminus o_F = \{o_f \in N \setminus M \mid |o_f| > 1\}$

          $\mathcal{R} = \mathcal{R} \cup \{(state(s, o_F), state(s', o'_F))\}$

        end for

      else

        $\mathcal{R} = \mathcal{R} \cup \{(state(s, o_F), state(s', o_F))\}$

      end if

    end for

  end for

end for

return $(\mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L})$

Figure 3.3 Algorithm for transforming CCRN into Kripke structure

3.4 Case Study of the Verification

We have conducted a case study of a verification of a given CCRN, using a model checking. We assume that a CCRN is given by the designer who intend to design applications of ubiquitous computing. Here we use an example of museum as shown in Fig. 3.4. A CCRN of this example is represented as a tuple $(O, C, R, (S, E, F), L_{FIX}, l_0)$ where

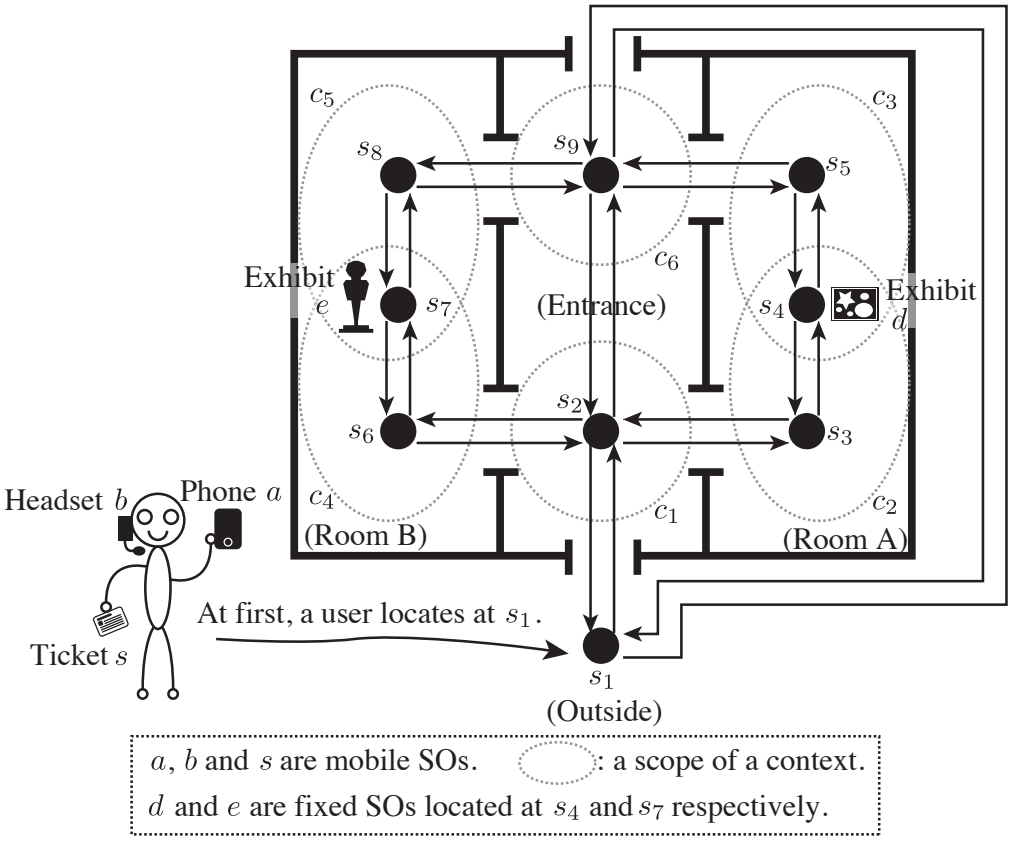
- $O = \{a, b, d, e, s\}$,
- $C = \{c_1, c_2, c_3, c_4, c_5, c_6\}$,
- $R = \{(c_1, \{\{a\}, \{b\}, \{s\}\}, \{\{a, b\}, \{s\}\}),$
  $(c_2, \{\{a, b\}, \{d\}\}, \{\{a, b, d\}\}),$
  $(c_3, \{\{a, b, d\}\}, \{\{a, b\}, \{d\}\}),$
  $(c_4, \{\{a, b\}, \{e\}\}, \{\{a, b, e\}\}),$
  $(c_5, \{\{a, b, e\}\}, \{\{a, b\}, \{e\}\}),$
  $(c_6, \{\{a, b\}, \{s\}\}, \{\{a\}, \{b\}, \{s\}\})\ \},$
- $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8, s_9\}$,
- $E = \{(s_1, s_2), (s_2, s_1), (s_2, s_3), (s_3, s_2), (s_3, s_4), (s_4, s_3),$
  $(s_4, s_5), (s_5, s_4), (s_5, s_9), (s_9, s_5), (s_2, s_6), (s_6, s_2),$
  $(s_6, s_7), (s_7, s_6), (s_7, s_8), (s_8, s_7), (s_8, s_9), (s_9, s_8),$
  $(s_9, s_1), (s_1, s_9)\}$,
- $F = \{(s_1, \emptyset), (s_2, \{c_1\}), (s_3, \{c_2\}), (s_4, \{c_2, c_3\}), (s_5, \{c_3\}),$
  $(s_6, \{c_4\}), (s_7, \{c_4, c_5\}), (s_8, \{c_5\}), (s_9, \{c_6\})\}$,

- $L_{FIX} = \{(d, s_4), (e, s_7)\}$, and
- $l_0 = s_1$.

In this example, a user enters the entrance of a museum, carrying a phone a, a headset b and a ticket s. Once the user entered the entrance, the phone a and the headset b are federated by a reaction associated with the scope of c_1, which is triggered by the ticket s. Then, the federated SOs ab are worked as a voice guide of the museum. Next, if the user enters into room A, the federated SO ab and an exhibit d are federated by a reaction associated with the scope of c_2. By the federated SO abd, an explanation of the exhibit d can be provided to the user. After this, the user leaves the room A and the federated SO abd is decomposed and becomes ab again by a reaction associated with the scope of c_3. The similar reactions occur in the room B, which is for an explanation of an exhibit e. If the user leaves one of the exhibition rooms and returns to the entrance, the federated SO ab is decomposed before leaving the museum.

Now we verify a CCRN of this example. Using an algorithm shown in Fig. 3.3, we can obtain a Kripke structure M. Then, the designer may give desired properties of the given CCRN by LTL formulae such as:

- $\phi_1 = \mathbf{G}(\neg(fed(\{a, b, d\}) \wedge fed(\{a, b, e\})))$,
- $\phi_2 = \mathbf{G}(\ (\neg fed(\{a, b, d\}) \rightarrow \mathbf{F}(fed(\{a, b, d\}))) \vee$
  $(\neg fed(\{a, b, e\}) \rightarrow \mathbf{F}(fed(\{a, b, e\})))\ )$, and
- $\phi_3 = \mathbf{G}(\ (loc_{O_{MOB}}(s_3) \vee loc_{O_{MOB}}(s_6)) \wedge fed(\{a, b\})\ )$.



Catalytic Reactions:

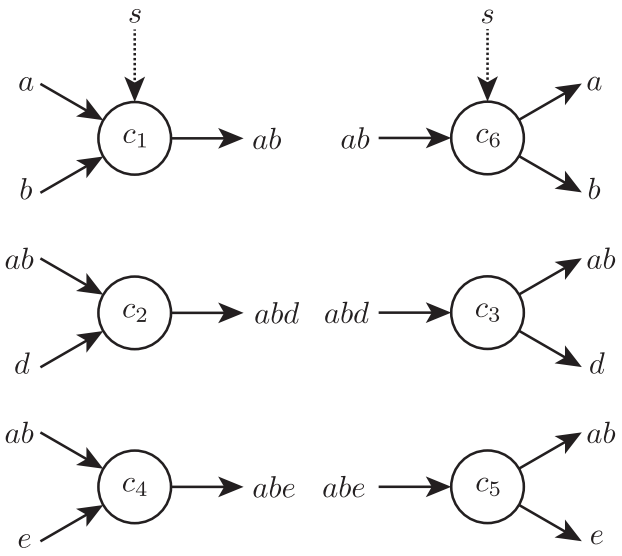


Figure 3.4 Example of Museum

Intuitively saying, ϕ_1 means that no more than one federation for the explanation of exhibits exists at the same time and ϕ_2 means that if a user enters into one of the exhibition rooms, an explanation of each exhibit is always provided to a user and ϕ_3 means that when a user enters into one of the exhibition rooms, the federation for a voice guide of the museum is always ready.

Now we verify a CCRN using a generated Kripke structure M, ϕ_1, ϕ_2 and ϕ_3. To conduct model checking, we used NuSMV2 as a model checking verifier.

In NuSMV2, SMV language is used to represent Kripke structures. In this chapter, we enumerated all possible states, their corresponding atomic propositions and transitions explicitly. For example, if we are given a set of atomic propositions $AP = \{p, q\}$ and a Kripke structure (S, I, R, L) where

- $S = \{s_0, s_1, s_2, s_3\}$,
- $I = \{s_0\}$,
- $R = \{(s_0, s_1), (s_0, s_2),$
  $(s_1, s_2),$
  $(s_2, s_3),$
  $(s_3, s_3)\}$ and
- $L(s_0) = \emptyset, L(s_1) = \{p\}, L(s_2) = \{q\}, L(s_3) = \{p, q\}$;

we generate a SMV language file such as Fig. 3.5 straightforwardly. The size of this SMV language file generated by like this way is in proportion

```
MODULE main
VAR
  state : {s0, s1, s2, s3};
ASSIGN
  init(state) := s0;
  next(state) :=
    case
      state = s0 : {s1, s2};
      state = s1 : {s2};
      state = s2 : {s3};
      state = s3 : {s3};
    esac;
DEFINE
  p := state = s1 | state = s3;
  q := state = s2 | state = s3;
```

Figure 3.5 Example of a SMV Language File

to the number of states, atomic propositions and transitions of given Kripke structure. NuSMV2 manual [8] describes details of SMV language format.

We have confirmed that $\forall\pi.(M, \pi \models \phi_1)$ is satisfied. However, $\forall\pi.(M, \pi \models \phi_2)$ and $\forall\pi.(M, \pi \models \phi_3)$ are *not* satisfied. A model checking verifier also gives a counterexample π_{c_2} and π_{c_3} corresponding to ϕ_2 and ϕ_3 respectively such as

$$\begin{aligned}\pi_{c_2} = (&state(s_1, \emptyset), state(s_2, \{\{a, b\}\}), state(s_3, \{\{a, b, d\}\}),\\ &state(s_4, \{\{a, b\}\}), state(s_5, \{\{a, b\}\}), state(s_9, \emptyset),\\ &state(s_5, \emptyset), state(s_4, \emptyset), state(s_5, \emptyset), state(s_4, \emptyset) \dots),\end{aligned}$$

and

$$\begin{aligned}\pi_{c_3} = (&state(s_1, \emptyset), state(s_2, \{\{a, b\}\}), state(s_3, \{\{a, b, d\}\}),\\ &state(s_4, \{\{a, b\}\}), state(s_5, \{\{a, b\}\}), state(s_9, \emptyset),\\ &state(s_8, \emptyset), state(s_7, \emptyset), state(s_6, \emptyset)\ ).\end{aligned}$$

Bold lines in Fig. 3.6 and Fig. 3.7 are the visualization of π_{c_2} and π_{c_3} respectively.

In the case of π_{c_2}, first, the user enters the entrance of the museum, then, the user goes to the room A and goes away from room A. But the user enters the room A again from which the user goes away. Finally, the user stays there. In this situation, we never obtain the federated SO *abd* again since the user stays in the room A. In the case of π_{c_3}, first, similarly to the case of π_{c_2}, the user enters the entrance of the museum, then, the user goes to the room A and goes away from room A. But the user enters the room B from which is intended for an exit of room B. And then, the user goes to the entrance of room B reversely. In this situation, the user don't have the federated SO *ab* when the user intend to receive the explanation of exhibit *e*, so the user can not receive

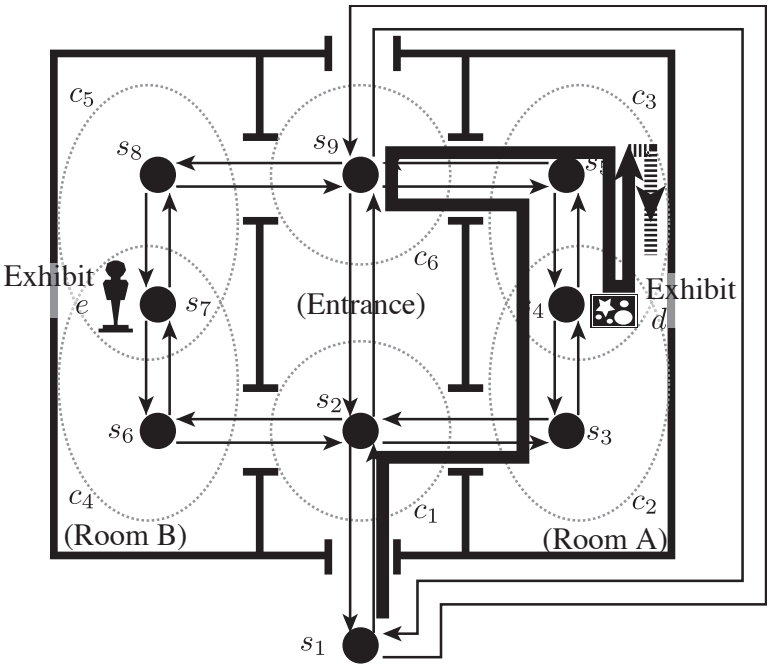


Figure 3.6 Counterexample corresponding to ϕ_2 of Museum Example

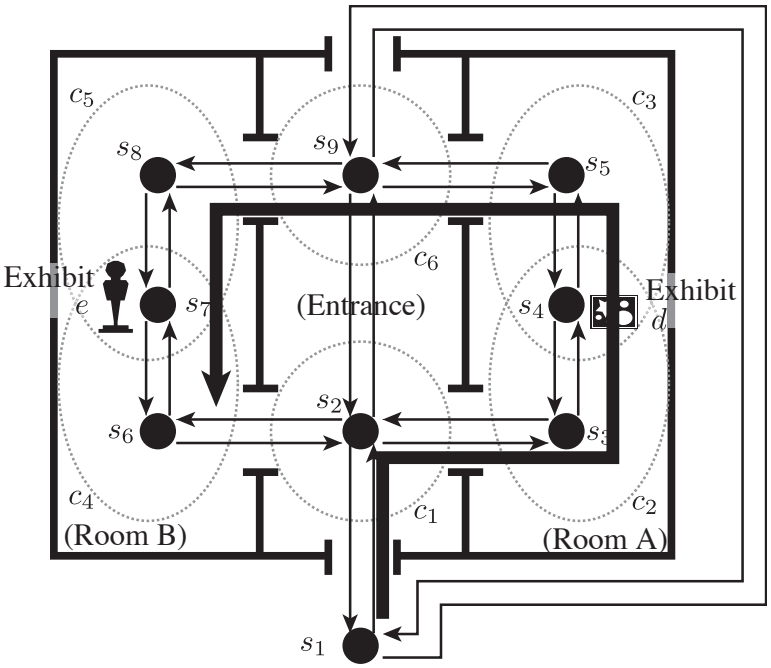


Figure 3.7 Counterexample corresponding to ϕ_3 of Museum Example

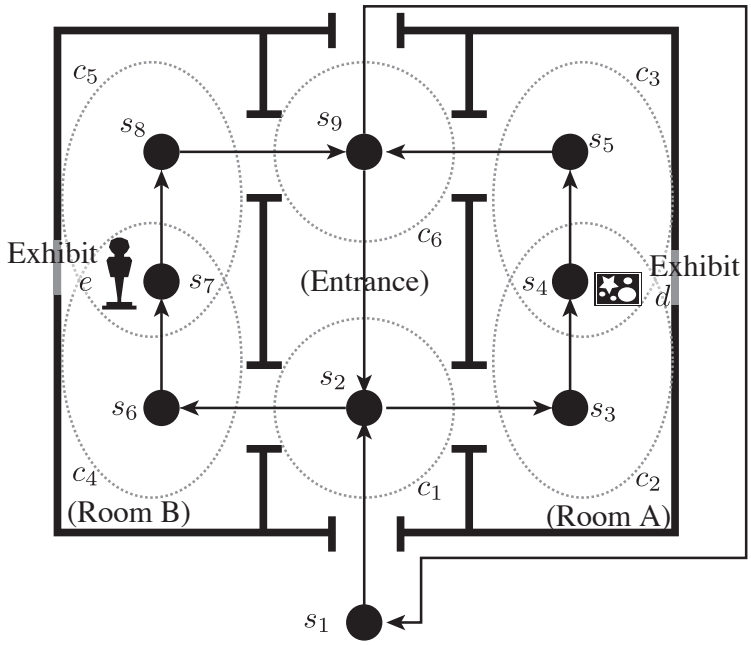


Figure 3.8 Revised Museum Example

the explanation ob exhibit *e*.

From these counterexamples, we learned that which some appropriate constraint on the segment graph is necessary.

Now we *debug* the system to make all properties of a given CCRN given by LTL formulae satisfied. To do so, we need to revise the segment graph of a given CCRN of this example. We have rewritten E of the given CCRN as follows (Fig. 3.8 is the visualization of this revision):

$$E = \{(s_1, s_2), (s_2, s_3), (s_3, s_4), (s_4, s_5), (s_5, s_9), (s_2, s_6), (s_6, s_7), (s_7, s_8), (s_8, s_9), (s_9, s_1)\}.$$

This revision indicates that the user should follow the *regular route* of the museum.

Then, we have conducted the model checking again using the revised Kripke structure M, ϕ_1, ϕ_2 and ϕ_3. Finally, we have confirmed

that all of $\forall\pi.(M, \pi \models \phi_1)$, $\forall\pi.(M, \pi \models \phi_2)$ and $\forall\pi.(M, \pi \models \phi_3)$ are satisfied. If all of these LTL formulae are satisfied, this museum meets all of requirements defined by the designer of this museum. Of course, the designer can try other properties within range of LTL, using a combination of two kind of atomic propositions.

In this case study, we show that our method actually helps designers of applications to find exceptions of the design of applications and to debug these exceptions using counterexamples provided by model checking verifiers through trial and error. Using our method, we can discuss the property such as the validity and the safety of applications consisting of mutually related multiple federations among SOs *without exhaustive hand simulation*. As we mentioned in previous section, model checking problems of Kripke structure with finite states can be reduced to a graph search. If we conduct this graph search on the this example by hand, we must test very large number of patterns of user's walk. Table 3.1 indicates the number of 1–20 steps graph walk patterns starting from segment s_1. For example, to detect the counterexample π_{c_3} which has 9 steps by hand, we must conduct the exhaustive test from 1 step to 9 steps cases to make sure whether ϕ_3 are satisfied or not. In this case, we must check 14,022 patterns of user's walk. This is very expensive. It is important to reduce to a model checking problem which is able to be solved by various implementations of model checking verifiers. Formal approaches such as this kind of verification liberates the designers from conducting exhaustive checking. Formal approaches such as this kind of verification is also important because it can avoid specifications errors of ubiquitous computing applications in advance of actual implementations of these

Table 3.1 Number of Graph Walk Patterns of Museum Example

# of Steps	# of Patterns	# of Steps	# of Patterns
1	2	11	66,332
2	8	12	182,300
3	20	13	495,900
4	60	14	1,359,132
5	156	15	3,704,604
6	444	16	10,138,396
7	1,180	17	27,664,156
8	3,292	18	75,648,796
9	8,860	19	206,538,524
10	24,476	20	564,549,404

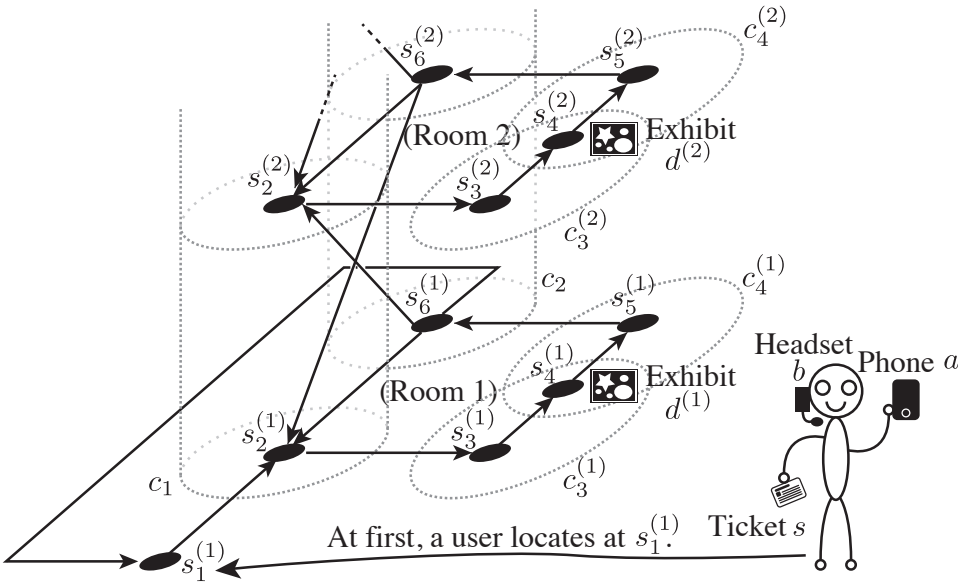
Remark: These results are the case of Fig. 3.4. In the revised case (Fig. 3.8), the number of graph walk patterns will be decreased.

applications, which may incur additional costs.

3.5 Scalability of Our Method

We evaluated the scalability of our method. To evaluate, we used a generalized example of the museum such as Fig. 3.9. In this example, there are n rooms and each room i has a exhibit $d^{(i)}$ and we defined reactions to provide an explanation of exhibit $d^{(i)}$ to the user in corresponding room i. Directed edges $(s_6^{(i)}, s_2^{(i+1)})$ and $(s_6^{(i+1)}, s_2^{(i)})$ of the segment graph represent stairs connected between room i and room $i+1$. We verified this example through the cases from $n = 1$ to $n = 6$. We set properties of these cases by a LTL formula $\mathbf{G}(loc_{O_{MOB}}(s_1^{(1)}) \rightarrow \mathbf{F}(loc_{O_{MOB}}(s_4^{(n)}) \rightarrow fed(\{a, b, d^{(n)}\}))$. This formula means that if the user once enters the museum, the exhibit explanation of the highest floor is always provided to the user.

We conducted an experiment by using a Core i7 3820QM machine with 16GB memory. In this experiment, we use NuSMV2 version 2.6.0 as a model checking verifier. Table. 3.2 indicates the experiment results of these cases. The left-hand side and the right-hand side of this table indicate the size of model checking problems and the cost needed for solving them by NuSMV2 respectively. The more the number of rooms, the more cost needed for solving increases exponentially. This is because of the size of $|O_F|$ which is defined by a power set of O. But note that, as we mentioned in sect. 1, if we consider this kind of verification problem by using *original* catalytic reaction model, we would be forced to consider 2^n states of n SOs' proximity relations on each segment. In that situation, the state space would be exploded more rapidly and ex-



a, b and s are mobile SOs. $d^{(i)}$ is fixed SOs located at $s_4^{(i)}$.
: a scope of a context.

Catalytic Reactions:

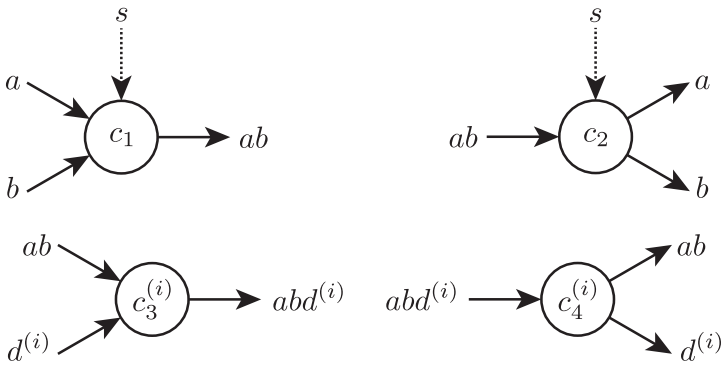


Figure 3.9 Example of Museum Containing Multiple Rooms

Table 3.2 Results of the Scalability Experiment

n	$\|O\|$	$\|C\|$	$\|S\|$	$\|\mathcal{S}\|$	CPU Time	MEM. Usage
1	4	4	6	30	0.01 s	13.81 MB
2	5	6	11	165	0.04 s	16.50 MB
3	6	8	16	832	0.41 s	48.46 MB
4	7	10	21	4,263	8.69 s	656.75 MB
5	8	12	26	22,802	273.56 s	13,088.76 MB
6	9	14	31	128,340	N/A	MEM. Out

Remark: "MEM. Out" means that we abort the calculation due to the lack of memory space.

ponentially and we would not be able to verify even a small case such as shown in Table. 3.2 in realistic time. For this reason, our framework is important as the first step of the formularization to verify scenarios of federations of SOs.

3.6 Summary

In this chapter, we proposed a verification method of applications which is described by a CCRN using model checking. Using our framework, various properties of application scenarios of ubiquitous computing can be discussed by logic such as LTL. Our framework actually helps the designers to debug ubiquitous computing application scenarios. With our framework, the cost of detecting any counterexamples is much reduced compared to hand simulation. These contributions are important as the first step of the formularization to verify ubiquitous computing scenarios.

To verify more practical ubiquitous computing scenarios, the scalability of the verification method is important. In next chapter, we will consider the efficient way of representing all possible states of given CCRN by introducing techniques such as *symbolic* approaches instead of the naive approach.

Chapter 4

Improving the Scalability of Scenario Verification Using Symbolic Model Checking

Verification of ubiquitous computing scenarios enables us to detect design-related faults of ubiquitous computing applications before we actually implement them. In this chapter, we propose a verification framework of CCRN, which is a description model of ubiquitous computing scenarios, as a new application field of symbolic model checking. To do so, we illustrate a method how to transform a scenario written in CCRN into a symbolic model checking problem. We also show experimentally that our framework makes it possible

to verify large scale ubiquitous computing scenarios which could not be verified in realistic time by our previous method. Additionally, we show experimentally the usefulness of fault detections using bounded model checking in the same framework.

4.1 Introduction

In the previous chapter, we extended a catalytic reaction network model to CCRN by adding a discrete structure called "segment graph". Using a CCRN model, we formalized a ubiquitous computing scenario verification problem and proposed a method to reduce ubiquitous computing scenario verification problems to model checking problems. This method enabled us to discuss the properties of ubiquitous computing scenarios. These properties are included as mentioned in the previous chapter.

Most important aspect of this kind of verification is that these discussions about properties of a given ubiquitous computing scenario can be done in the step of the *design* before the implementation steps which may incur additional costs.

However, this reduction method was a kind of *naive* approach so there were challenges related to the scalability during the verification. In this chapter, we propose a verification method using symbolic model checking to improve the scalability. Symbolic model checking is a one of model checking technique to verify very large scale of state transition

systems [7]. We show the method how to take advantage of symbolic model checking techniques when we reduce ubiquitous computing scenario verification problems to model checking problems. We also show experimentally that our method can verify larger ubiquitous computing scenario verification problems compared to our previous naive approach. As a result, this method enables us to discuss more practical ubiquitous computing scenario verification problems.

Additionally, if we establish a reduction method using a symbolic model checking approach, we can also take advantage of bounded model checking in the same framework [4]. Bounded model checking can detect counterexamples from a given model checking problem faster than symbolic model checking in most cases (Note that bounded model checking is not good at proving that there is no counterexample of a given model checking problem). Bounded model checking is widely used for the counterexample detection from model checking problems. In this chapter, we also evaluate experimentally how practical is bounded model checking for ubiquitous computing scenario verification problems.

4.2 CCRN Verification through Symbolic Model Checking

In this section, we propose a method of reducing a CCRN verification to a symbolic model checking problem. Concretely, we propose a way to transform a CCRN $(O, C, R, (S, E, F), L_{\mathrm{FIX}}, l_0)$ into a variable vector $\boldsymbol{s}$ and Boolean functions $S(\boldsymbol{s})$, $I(\boldsymbol{s})$ and $T(\boldsymbol{s})$.

There are two types of states in a CCRN $(O, C, R, (S, E, F), L_{\mathrm{FIX}}, l_0)$. One is a state of mobile SOs' location. A set of mobile SOs O_{MOB} are defined as $O \setminus \{o \in O \mid \exists s \in S.((o, s) \in L_{\mathrm{FIX}}\}$. O_{MOB} are carried together by a user and are located at segment $s \in S$, so this kind of states has $|S|$ states. The other type of states is a state of existence of federated SOs o_f. Federated SOs o_f are defined as $o_f \in 2^O \setminus \{\}$, so $2^{|O|} - 1$ kinds of o_f can be considered. As a result, there are $2^{2^{|O|}-1}$ states of $2^{|O|} - 1$ kinds of o_fs' existence. Therefore, there are $|S| \times 2^{2^{|O|}-1}$ states of a given CCRN if we count its states explicitly. Now we deal with this large states by introducing a symbolic approach for efficient verification.

At first, we define a variable vector $\boldsymbol{s}$ representing states. Considering the above discussion, we represent a state that mobile SOs O_{MOB} are located at segment $s \in S$ as "$segment = s$", and a state that federated SOs o_f are existing as "$fed(o_f) = \mathrm{True}$" respectively. Using those, we define a variable vector $\boldsymbol{s}$ as follows.

$$\boldsymbol{s} = (segment, \underbrace{fed(o_f), fed(o_f{}'), \cdots}_{2^{|O|}-1}) \tag{4.1}$$

In this chapter, Boolean function $S(\boldsymbol{s})$ representing all possible states are defined as a Boolean function that returns True for all $\boldsymbol{s}$. Instead, we define appropriate $T(\boldsymbol{s}, \boldsymbol{s}')$ in the rest of this section. Boolean function

$I(\boldsymbol{s})$ representing initial states are defined as follows.

$$\begin{aligned} I(\boldsymbol{s}) =& (segment = l_0) \\ & \wedge \left(\bigwedge_{o_f \in 2^O \setminus \{\}, |o_f|=1} fed(o_f) = \text{True} \right) \\ & \wedge \left(\bigwedge_{o_f \in 2^O \setminus \{\}, |o_f|>1} fed(o_f) = \text{False} \right) \end{aligned} \tag{4.2}$$

Next, we define $T(\boldsymbol{s}, \boldsymbol{s}')$. To make the representation of $T(\boldsymbol{s}, \boldsymbol{s}')$ simple, we define auxiliary variables. We define transitions S_e that mobile SOs S_{MOB} move from segment s to segment s' along the edge e of a given segment graph as follows.

$$S_e \triangleq (segment = s) \wedge (segment' = s') \tag{4.3}$$

We also consider transitions of changing the state of federated SOs' existence. In the case of CCRN, this kind of state changes are occurred when catalytic reactions are occurred. We define transitions $R_\emptyset$ that no catalytic reaction was occurred as follows.

$$R_\emptyset \triangleq \bigwedge_{o_f \in 2^O \setminus \{\}} \Big(fed(o_f) = fed'(o_f) \Big) \tag{4.4}$$

Furthermore, we define transitions R_r that catalytic reaction $r = (c, M, N)$ was occurred as follows.

$$\begin{aligned} R_r \triangleq & \bigwedge_{o_f \in N} \Big(fed'(o_f) = \text{True} \Big) \wedge \bigwedge_{o_f \in M} \Big(fed'(o_f) = \text{False} \Big) \\ & \wedge \bigwedge_{o_f \in (2^O \setminus \{\}) \setminus (M \cup N)} \Big(fed(o_f) = fed'(o_f) \Big) \end{aligned} \tag{4.5}$$

And we define the condition RC_r which is necessary for catalytic reaction r as follows.

$$RC_r \triangleq \bigwedge_{o_f \in M} \Big(fed(o_f) = \text{True} \Big) \tag{4.6}$$

A set of catalytic reactions $R_{\mathrm{app}}(e)$ that are occurred when mobile SOs O_{MOB} move from segment $s \in S$ to segment $s' \in S$ along the edge e in a given segment graph, is defined as follows.

$$\begin{aligned} &R_{\mathrm{app}}(e) \triangleq \{(c, M, N) \in R \mid c \in F(s'), O(c) \supseteq \bigcup M\} \text{ where} \\ &O(c \in C) = O_{\mathrm{MOB}} \cup \{o \in O \mid \exists s'' \in S.(c \in F(s''), (o, s'') \in L_{\mathrm{FIX}}\} \end{aligned} \tag{4.7}$$

Then, we define transition relation T_e of each edge $e \in E$ of a given segment graph as follows.

$$T_e \triangleq \begin{cases} S_e \wedge R_\emptyset & \text{if } R_{\mathrm{app}}(e) = \emptyset \\ \bigvee\limits_{r \in R_{\mathrm{app}(e)}} \left(S_e \wedge RC_r \wedge R_r \right) \vee & \\ \left(S_e \wedge \neg \left(\bigvee\limits_{r \in R_{\mathrm{app}(e)}} RC_r \right) \wedge R_\emptyset \right) & \text{otherwise} \end{cases} \tag{4.8}$$

Finally, using the definition of T_e, we define $T(\boldsymbol{s}, \boldsymbol{s}')$ as follows.

$$T(\boldsymbol{s}, \boldsymbol{s}') = \bigvee_{e \in E} T_e \tag{4.9}$$

4.3 Experiments and Discussion

In this section, we report results of two kinds of experiments. First one is intended to show the scalability of the proposed method. Second one is aimed to show the usefulness of bounded model checking.

4.3.1 Scalability of CCRN Verification through Symbolic Model Checking

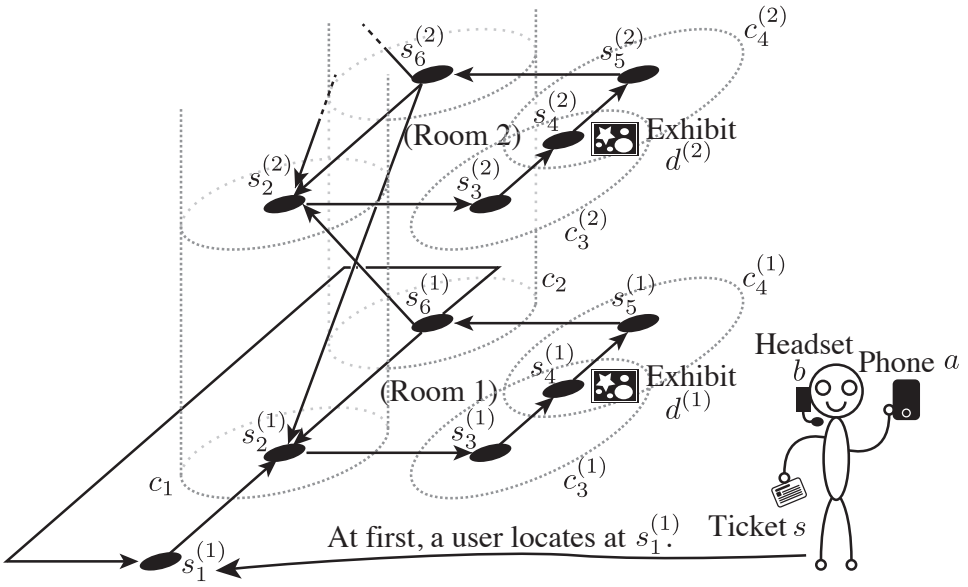
In this experiment, we evaluated the scalability of our method. To conduct the experiment, we used a CCRN such as Fig.4.1. Left hand side of Fig.4.1 represents the corresponding segment graph of the CCRN. This CCRN assumes an ubiquitous computing scenario of a museum that has n rooms and each room i has a exhibit $d^{(i)}$.

We define reactions c_1 and c_2 to make a phone a and a headset b federated for a museum guide service and we also define reactions $c_3^{(i)}$ and $c_4^{(i)}$ to provide an explanation of exhibit $d^{(i)}$ to the user in corresponding room i. Directed edges $(s_6^{(i)}, s_2^{(i+1)})$ and $(s_6^{(i+1)}, s_2^{(i)})$ of the segment graph represent stairs connected between room i and room $i+1$. We set properties of these cases by following CTL formula.

$$\begin{aligned}&\mathbf{AG}(segment = s_1^{(1)} \\ &\quad \rightarrow \mathbf{AF}(segment = s_4^{(n)} \rightarrow fed(\{a, b, d^{(n)}\}) = \text{True}))\end{aligned} \tag{4.10}$$

This formula means that if the user once enters the museum, the exhibit explanation of the highest floor is always provided to the user. And this museum example satisfies this formula. So, in this case, we verified the CCRN to confirm this formula actually does satisfy.

We conducted this experiment by using a Core i7 3820QM machine with 16GB memory. In this experiment, we use NuSMV2 version 2.6.0 as a model checking verifier. We also compared with our naive method which is in previous chapter. The naive method enumerates all possible states of a given CCRN explicitly. So even if we use NuSMV2,



a, b and s are mobile SOs. $d^{(i)}$ is fixed SOs located at $s_4^{(i)}$.
: a scope of a context.

Catalytic Reactions:

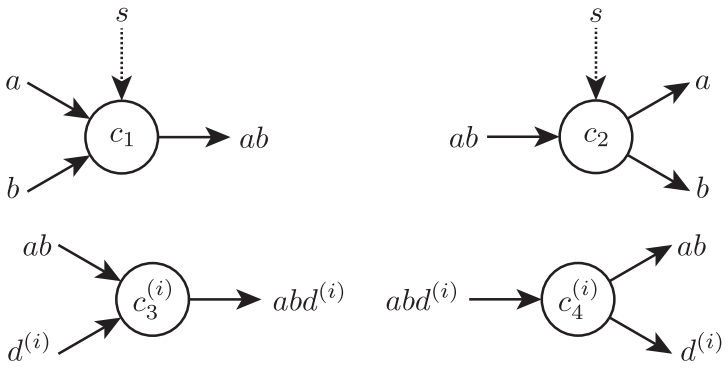


Figure 4.1 Museum Example of CCRN

this naive method do not take any advantage of symbolic model checking. Table. 4.1 indicates the experiment results through the cases from $n = 1$ to $n = 11$. The left-hand side, the middle and the right-hand side of this table indicate the size of model checking problems, the cost needed for solving them by our previous naive method and the cost needed for solving them by our proposal method respectively. On this machine used for this experiment, the instance of $n = 5$ is a limit of our previous naive method. However, our proposal method extended the limit to $n = 10$. In the case of $n = 10$, there would be about 2.78×10^{2467} states if we enumerate those states explicitly. It also can be said that our proposed method reduced the double exponential scale of problem into the single exponential scale of problem. From these results, we illustrate that using symbolic model checking enables us to verify more large scale of ubiquitous computing scenarios in realistic time and costs.

Table 4.1 Experiment Results of The Scalability Evaluation

Problem Instance				Naive Method (chap. 3)		Proposal Method	
n	$\|O\|$	$\|C\|$	$\|S\|$	CPU (s)	MEM (MB)	CPU (s)	MEM (MB)
1	4	4	6	0.01	12.06	0.01	12.48
2	5	6	11	0.01	13.50	0.01	13.67
3	6	8	16	0.24	37.05	0.01	16.35
4	7	10	21	5.45	566.57	0.02	24.11
5	8	12	26	172.30	12,528.02	0.08	47.68
6	9	14	31	N/A	MEM. Out	0.37	127.97
7	10	16	36	—	—	1.47	425.63
8	11	18	41	—	—	5.75	1,546.30
9	12	20	46	—	—	57.30	6,113.32
10	13	22	51	—	—	264.26	11,309.29
11	14	24	56	—	—	N/A	MEM. Out

Remarks: "MEM. Out" means that we abort the experiment due to the lack of memory.

4.3.2 Detecting Faults of CCRN through Bounded Model Checking

We conducted another kind of experiment to show the usefulness of bounded model checking. In this experiment, we used the same example of the CCRN as shown in Fig.4.1. However, to detect faults of the given CCRN, we omitted randomly reactions $c_3^{(i)}$ and $c_4^{(i)}$ on purpose. By doing this, we generated different 10 problem instances from each example of museum with n rooms through the cases from $n = 1$ to $n = 10$. So, totally, we verified 100 problem instances of "degraded" museum examples by bounded model checking. We set properties of these cases by a LTL formula as follows.

$$\mathbf{G} \bigwedge_{i=1}^{n} \Big((\neg(segment = s_4^{(i)}) \wedge fed(\{a, b, d^{(i)}\}) = \text{False}) \\ \vee (segment = s_4^{(i)} \wedge fed(\{a, b, d^{(i)}\}) = \text{True}) \Big) \tag{4.11}$$

This means that all reactions for corresponding explanation of exhibits in each room occurs properly. We conducted this experiment by using the same machine and same version of NuSMV2 as the scalability experiment and we set the bound of verification $k = 50$. Experimental results are represented in Fig. 4.2 and Fig. 4.3. In Fig.4.2, each dot's color corresponds to the size of the problem instance (i.e., the number of rooms) and each dot's coordinate indicates the computational costs needed for the problem instance verification to detect the faults [*1]. From

[*1] For all of problem instances of this experiment, it takes less than 0.1 sec to make an counterexample of each instances. This means that time costs spent on detections of faults are dominant. This tendency is also observed in the case of Fig. 4.3.

this results, we can conclude that most of cases are faster and less memory usage than the cases of symbolic model checking when we are intended to detect faults of a given CCRN. For example, there is a case of detecting a fault in less than 20 sec from one of problem instances of $n = 10$. Figure 4.3 shows detailed experimental results of fault detection time comparison between symbolic model checking and bounded model checking. When we conduct this fault detection using symbolic model checking, we use following a CTL formula which is equivalent to the LTL formula above.

$$\mathbf{AG} \bigwedge_{i=1}^{n} \Big((\neg(segment = s_4^{(i)}) \wedge fed(\{a, b, d^{(i)}\}) = \text{False}) \\ \vee (segment = s_4^{(i)} \wedge fed(\{a, b, d^{(i)}\}) = \text{True}) \Big) \tag{4.12}$$

In Fig. 4.3, each of plots represents a problem instance and its color corresponds to the size of the problem instance just like Fig. 4.2. In this figure, a horizontal axis and a vertical axis are computational time needed for the fault detection of the problem instance by using bounded model checking and symbolic model checking respectively. If a dot is plotted above the dotted diagonal line, it means bounded model checking is faster than symbolic model checking in a corresponding problem instance of the dot. From this figure, it can be said that bounded model checking is faster than symbolic model checking to conduct fault detections in most of the cases of problem instances. In the fastest case, bounded model checking is about 10 times faster than symbolic model checking.

Note that some cases of $n = 1$ enclosed with a circle in Fig.4.2 take more time and more memory space because no reactions of these cases are omitted by chance. An usual bounded model checking verifier

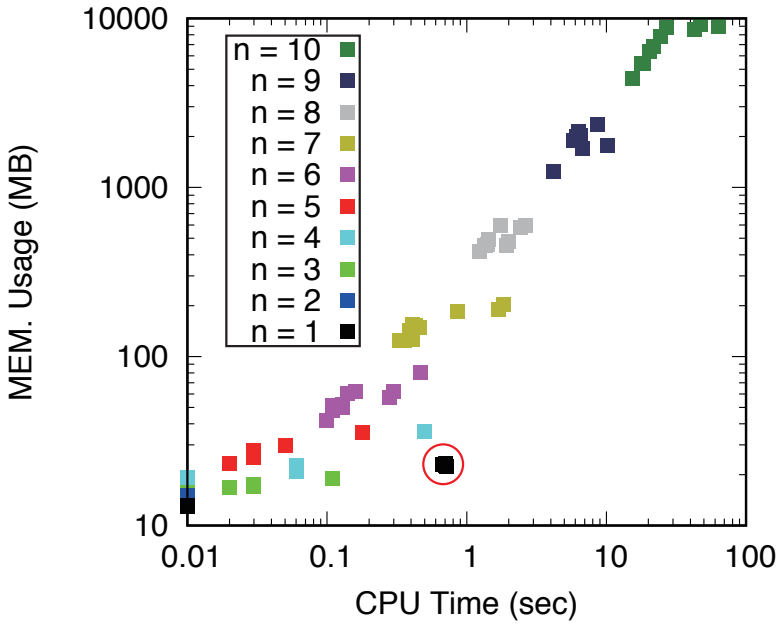


Figure 4.2 Computational Costs for CCRN Fault Detection

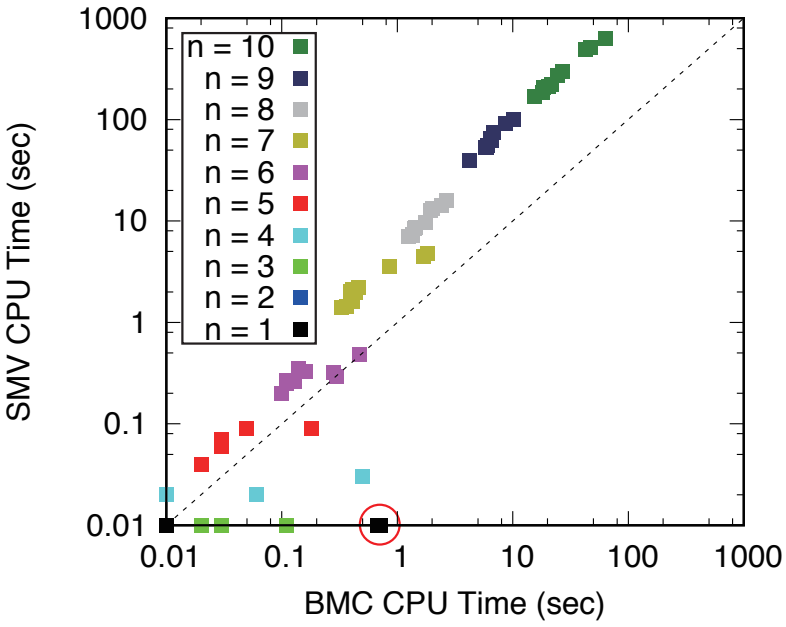


Figure 4.3 Fault Detection Time Comparison between Symbolic Model Checking (SMV) and Bounded Model Checking (BMC)

confirms the property inductively by changing the bound through $k = 1$ to $k = 50$. The verifier aborts the verification as soon as it detects any faults. However if there is no faults in cases which we remark above, the verifier is forced to verify until $k = 50$ (in this case of this experiment setting) and this causes to take more time and more memory spaces. In other words, we can expect that a bounded model checking method detects faults very fast if they exists.

4.4 Summary

We proposed a method to reduce a CCRN verification problem to a symbolic model checking problem. Our proposal method enables us to verify more large scale ubiquitous computing scenarios in realistic time and memory space. To show that symbolic model checking is useful approach to verify ubiquitous computing scenarios, we conducted experiments using a museum example of ubiquitous computing scenario as a case study. Additionally, we also show that bounded model checking is also useful approach especially to detect faults of ubiquitous computing scenario.

Chapter 5

Reliability Analysis of Ubiquitous Computing Scenario Using Probabilistic Model Checking

This chapter proposes the method of reliability analysis of ubiquitous computing scenarios. In ubiquitous computing scenarios, various devices communicate with each other through wireless network, and this kind of communications sometimes break due to external interferences. To discuss the reliability in such situation, we introduce the notion of probability into CCRN, which is a description model of

ubiquitous computing scenarios. This enables us to conduct quantitative analyses such as considerations of a trade-off between the reliability of ubiquitous computing scenarios and the costs which may be necessary for their implementations. To conduct a reliability analysis of ubiquitous computing scenarios, we use the technique of probabilistic model checking. We also evaluate our method experimentally by conducting a case study using a practical example assuming a museum.

5.1 Introduction

In previous chapters, we showed an approach to verify ubiquitous computing scenarios by proposing CCRN and its verification through model checking. We also proposed more efficient method of this kind of verifications through symbolic model checking. These contributions enabled us to verify the property of ubiquitous computing scenario, which is described in formal logic formulation and these verifications are conducted systematically thanks to various model checking verifiers, such as NuSMV2 [9].

However, there are still challenges of our approach. For example, ubiquitous computing scenarios are assumed that SOs typically communicate with each other by the wireless communication. This means that we need to consider these communications sometimes break due to various external causes. For this reason, it is important to discuss this

kind of interference formally. In this chapter, we show an approach to reliability analysis of ubiquitous computing scenarios by introducing a notion of probability to CCRN, which is a description model of ubiquitous computing scenarios. To analyze this kind of reliability, we use the technique of probabilistic model checking [25].

5.2 Probabilistic CCRN

In this section, we propose probabilistic CCRN (P-CCRN), which is the extended model of CCRN by adding a notion of probability. To introduce a notion of probability, we consider two kinds of probability in CCRN. One is the probability of a user behavior, and the other one is the probability of each of catalytic reactions. We denote the probability of users' moving from segment i to segment j by $\tau_{i,j}$ and for all segments $i \in \mathrm{S}$ of a given segment graph, it is satisfied that

$$\sum_{j,(i,j)\in E} \tau_{i,j} = 1 \tag{5.1}$$

where E is a set of edges in a given segment graph. We denote the probability of the occurrence of catalytic reaction r by $\theta_r = [0,1]$. By getting them together, we define a probabilistic function P as follows.

Definition 5.2.1 (Probabilistic Component) A probabilistic component **P** is a tuple $\langle \mathrm{T}, \Theta \rangle$ where $\mathrm{T} = \{\tau_{i,j} \in [0,1] \mid (i,j) \in \mathrm{E}\}$, $\Theta = \{\theta_r \in [0,1] \mid r \in \mathrm{R}\}$, E is a set of edges in a segment graph and R is a set of catalytic reactions.

Definition 5.2.2 (Probabilictic CCRN) Let **C** and **P** be a CCRN and a probabilistic component respectively. A probabilistic CCRN (P-CCRN)

is a tuple of $\langle \mathbf{C}, \mathbf{P} \rangle$.

5.3 Formulation of P-CCRN Reliability Analysis

This section shows a reliability analysis method by using P-CCRN. To do so, we define states of P-CCRN and represent transitions between two states as a probability function. Finally, we propositionize states of P-CCRN to conduct probabilistic model checking.

5.3.1 State Representation

Let U be a set of states included in CCRN. Each of states of given CCRN u can be represented as a pair of states of the user's location S_{seg} and states of SOs' federation S_{fed} denoted by $\langle S_{\text{seg}}, S_{\text{fed}} \rangle$ where $S_{\text{seg}} \in \text{S}$ and $S_{\text{fed}} \in \mathfrak{P}(\text{O})$ (i.e., S_{fed} is a partition set of O). We also assume the independence between two events a user behavior and catalytic reactions' success or failure.

5.3.2 Transition Representation

A transition (u, u^{next}) between two states are occurred when a user who has SOs moves along with a directed edge in given segment graph. S_{seg} is changed directly when the user moves and S_{fed} is changed by catalytic reactions of corresponding contexts located at the segment that the user aims to go. We define the function of $r_i(S_{\text{fed}})$ to represent applications of catalytic reactions.

Definition 5.3.1 (Catalytic Reaction Application) Let $r_i = (c_i, M_i, N_i)$ and S_{fed} be a catalytic reaction and a state of SOs' federation respectively. $r_i(S_{\text{fed}}) : \mathfrak{P}(\mathrm{O}) \to \mathfrak{P}(\mathrm{O})$ is a function of catalytic reaction application which is a procedure of following update of given S_{fed}:

$$r_i(S_{\text{fed}}) = \begin{cases} S_{\text{fed}} \setminus del(r_i) \cup add(r_i) & \text{if } pre(r_i) \subseteq S_{\text{fed}} \\ S_{\text{fed}} & \text{otherwise} \end{cases} \tag{5.2}$$

where $pre(r_i) \triangleq M$, $add(r_i) \triangleq N \setminus M$ and $del(r_i) \triangleq M \setminus N$.

When the state of CCRN is $\langle S_{\text{seg}}, S_{\text{fed}} \rangle$ and a transition $(\langle S_{\text{seg}}, S_{\text{fed}} \rangle, \left\langle S_{\text{seg}}^{\text{next}}, S_{\text{fed}}^{\text{next}} \right\rangle)$ occurs, $\exists S_{\text{seg}}^{\text{next}}.(S_{\text{seg}}, S_{\text{seg}}^{\text{next}}) \in E$ and $\exists r.(r \in R(S_{\text{seg}}^{\text{next}}, S_{\text{fed}}))$ are selected probabilistically where $R(s, \mathrm{A}) = \{r_i \mid c_i \in F(s) \wedge pre(r_i) \subseteq \mathrm{A}\}$.

5.3.3 Assigning the probability of a transition between two states

Now we assign the probability of a transition $(\langle S_{\text{seg}}, S_{\text{fed}} \rangle, \left\langle S_{\text{seg}}^{\text{next}}, S_{\text{fed}}^{\text{next}} \right\rangle)$ between two states of given probabilistic CCRN. This probability is represented as $P(\left\langle S_{\text{seg}}^{\text{next}}, S_{\text{fed}}^{\text{next}} \right\rangle \mid \langle S_{\text{seg}}, S_{\text{fed}} \rangle)$. Using the assumption of the independence between two events a user behavior and catalytic reactions' success or failure, we can rewrite this probability as follows:

$$P(\left\langle S_{\text{seg}}^{\text{next}}, S_{\text{fed}}^{\text{next}} \right\rangle \mid \langle S_{\text{seg}}, S_{\text{fed}} \rangle) = P(S_{\text{seg}}^{\text{next}} \mid S_{\text{seg}}) P(S_{\text{fed}}^{\text{next}} \mid S_{\text{seg}}^{\text{next}}, S_{\text{fed}}) \tag{5.3}$$

$P(S_{\text{seg}}^{\text{next}} \mid S_{\text{seg}})$ can be defined from T directly, namely,

$$P(S_{\text{seg}}^{\text{next}} \mid S_{\text{seg}}) \triangleq \tau_{S_{\text{seg}}, S_{\text{seg}}^{\text{next}}}. \tag{5.4}$$

On the other hand, $P(S_{\text{fed}}^{\text{next}} \mid S_{\text{seg}}^{\text{next}}, S_{\text{fed}})$ can be defined by several ways. For example, if there are more than one catalytic reaction that can be applied when a user enters into a segment $S_{\text{seg}}^{\text{next}}$ with federated devices S_{fed}, at first, we evaluate the probability of all catalytic reactions independently like we try a coin flip with the number of these reactions of coins and assume that heads are reactions that are applied. In this chapter, we give three strategy to deal with this kind of situation.

1. If there are more than one head, we choose one of them uniformly. This represents *mutual exclusion* of concurrent processes among multiple devices.

2. If there are more than one head, we do *not* choose any of these. This assumes that the mutual exclusion does not work and of course this kind of situation should be avoided properly.

3. Let all the catalytic reactions be indexed in order of reaction rate and if there are more than one head, we choose the catalytic reaction with lowest number from them. This represents that fastest catalytic reaction is applied at the highest priority.

In this chapter, we use the first strategy to conduct case studies in latter section. This strategy can be represented as follows:

$$
\begin{aligned}
&P(S_{\text{fed}}^{\text{next}} \mid S_{\text{seg}}^{\text{next}}, S_{\text{fed}}) = \\
&\quad \begin{cases} \prod_{i \in \mathrm{R}'} (1 - \theta_i) & \text{if } S_{\text{fed}}^{\text{next}} = S_{\text{fed}} \\ \sum_{j=1}^{|\mathrm{R}'|} \sum_{\chi \in X_{ij}} \frac{1}{j} \prod_{k \in \chi} \theta_k \prod_{\ell \in \mathrm{R}' \setminus \chi} (1 - \theta_\ell) & \text{otherwise} \end{cases} \\
&\qquad \text{such that } S_{\text{fed}}^{\text{next}} \setminus S_{\text{fed}} = \mathit{add}(r_i) \text{and } r_i \in \mathrm{R}', \\
&\qquad \text{where } \mathrm{X}_{ij} = \{i\} \cup \binom{\mathrm{R}' \setminus \{i\}}{j-1} \text{ and } \mathrm{R}' = R(S_{\text{seg}}^{\text{next}}, S_{\text{fed}}). \quad (5.5)
\end{aligned}
$$

5.3.4 Propositionizing

To conduct probabilistic model checking, we assign two kinds of propositions $\mathrm{fed}(\mathrm{O}' \subseteq \mathrm{O})$ and $\mathrm{seg}(s \in \mathrm{S})$ to each states of given P-CCRN. Given a state $\langle S_{\mathrm{seg}}, S_{\mathrm{fed}} \rangle$, semantics of these propositions are defined as follows:

- $\mathrm{seg}(s \in \mathrm{S}) \models \top$ iff $s = S_{\mathrm{seg}}$ (a user locates at segment s)
- $\mathrm{fed}(\mathrm{O}' \subseteq \mathrm{O}) \models \top$ iff $\mathrm{O}' \in S_{\mathrm{fed}}$ (a federation O' exists)

5.4 Case Study of Reliability Analysis

We have conducted a case study of a reliability analysis of a given P-CCRN, using probabilistic model checking. We assume that a CCRN is given by the designer who intend to design applications of ubiquitous computing. Here we use a CCRN of a museum example as shown in Figure 5.1. Left hand side and right hand side of this figure represent the segment graph **G** and the catalytic reaction network R of this CCRN respectively. In this example, a user enters the entrance of a museum, carrying a phone a, a headset b and a ticket s. Once the user entered the entrance, the phone a and the headset b are federated by a reaction associated with the scope of c_1, which is triggered by the ticket s. Then, the federated SOs ab are worked as a voice guide of the museum. Next, if the user enters into room A, the federated SO ab and an exhibit d are federated by a reaction associated with the scope of c_2. By the federated

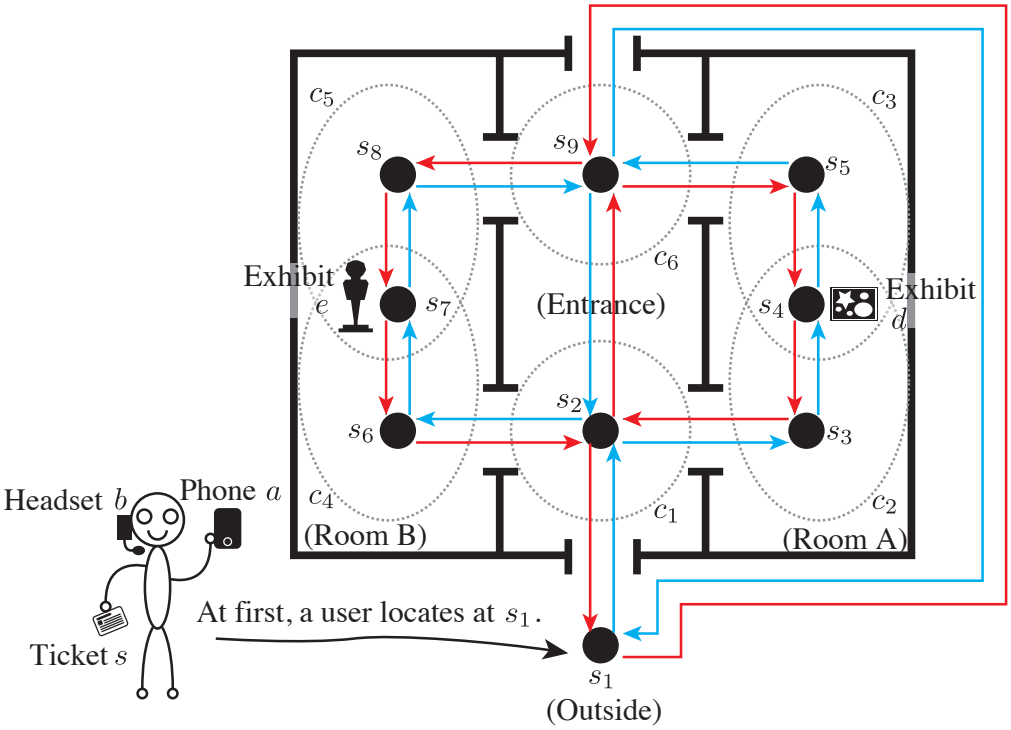
SO *abd*, an explanation of the exhibit *d* can be provided to the user. After this, the user leaves the room A and the federated SO *abd* is decomposed and becomes *ab* again by a reaction associated with the scope of c_3. The similar reactions occur in the room B, which is for an explanation of an exhibit *e*. If the user leaves one of the exhibition rooms and returns to the entrance, the federated SO *ab* is decomposed before leaving the museum.

Next we assign the probability to the user movement and catalytic reactions. In Figure 5.1, every directed edges of the segment graph is colored with blue or red. Blue edges assume the regular route of the museum to tour and red edges assume the opposite (i.e., wrong) way. The user can move along with these edges but here we use parameter $\alpha \in [0,1]$ to decide how frequent he or she tends to go along with the regular route. More precisely, in every segments, the user chooses blue edges with a probability of α, otherwise, he or she chooses red edges with a probability of $1-\alpha$. Then, if there are more than one edge after he or she chooses color of edge, he or she chooses an one edge uniformly from them. For all $(i,j) \in \mathrm{E}$, we can set $\tau_{i,j}$ as follows:

$$\tau_{i,j} = \begin{cases} \alpha / |\mathrm{BLUE}_i| & \text{if } \tau_{i,j} \text{ is a blue edge} \\ (1-\alpha) / |\mathrm{RED}_i| & \text{if } \tau_{i,j} \text{ is a red edge} \end{cases} \tag{5.6}$$

where BLUE_i is a set of $\{(i,j) \in \mathrm{E} \mid (i,j) \text{ is a blue edge}\}$ and RED_i is a set of $\{(i,j) \in \mathrm{E} \mid (i,j) \text{ is a red edge}\}$. In regards to catalytic reactions, we assign the same probability $\beta \in [0,1]$ to occurrences of all catalytic reactions. Namely, we set $\theta_r = \beta$ for all $r \in \mathrm{R}$.

In this configuration, we conducted an experiment of reliability analysis of P-CCRN. We use PRISM to evaluate the probability of fol-



a, b and s are mobile SOs. : a scope of a context.
d and e are fixed SOs located at s_4 and s_7 respectively.

Catalytic Reactions:

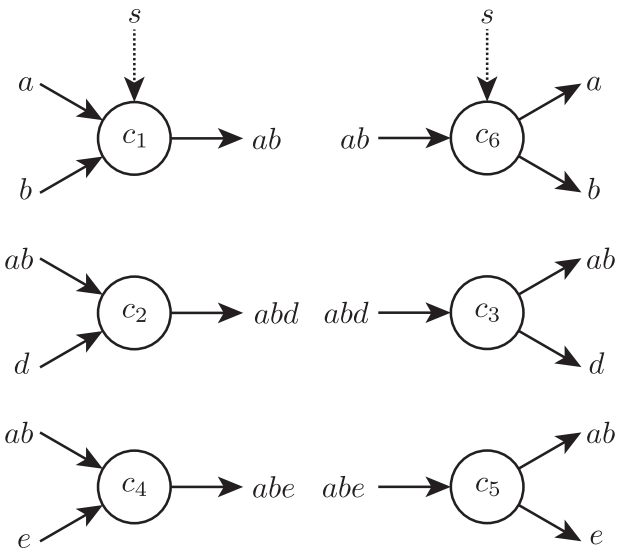


Figure 5.1 A CCRN assuming a museum

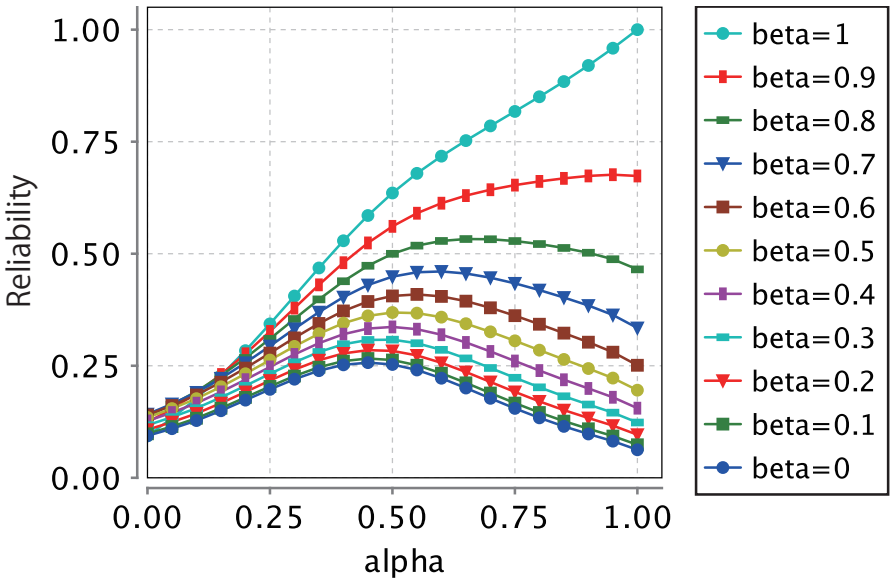
lowing properties with the bound parameter $k = 20$.

$$\phi_1 = \mathbf{P}_{=?}\left[\mathbf{G}_{\leq k}\left(\neg \mathrm{seg}(s_3) \vee \mathrm{fed}(\{a, b, d\})\right)\right] \tag{5.7}$$

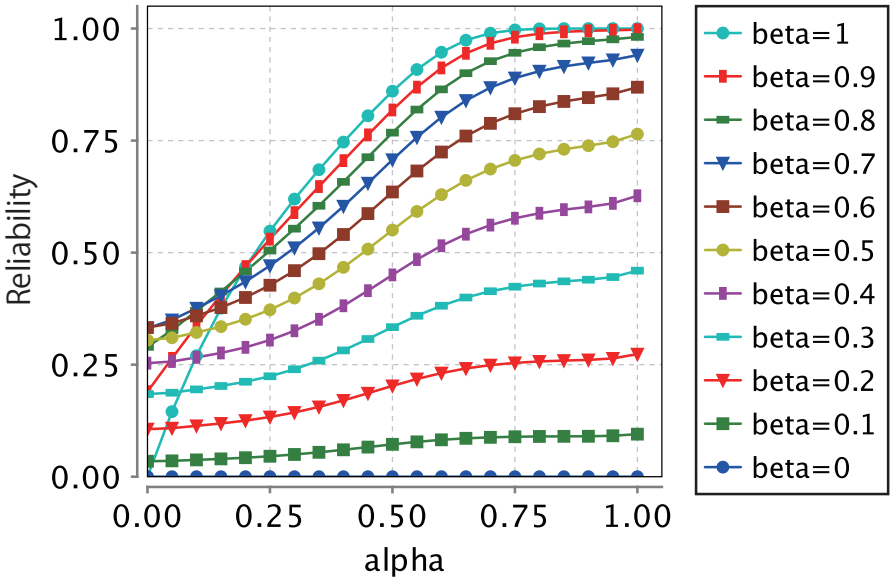
$$\phi_2 = \mathbf{P}_{=?}[\mathbf{F}_{\leq k}((\mathrm{seg}(s_3) \wedge \mathrm{fed}(\{a, b, d\})) \vee (\mathrm{seg}(s_6) \wedge \mathrm{fed}(\{a, b, e\}))\,)] \tag{5.8}$$

Intuitively, ϕ_1 means that how frequent the user can be always provided the explanation of exhibition d when he or she is at segment s_2 and ϕ_2 means that how frequent the user can be provided the explanation of exhibition d or e even just once when he or she enters the corresponding room of the exhibition.

Figure 5.2 shows results of these probability evaluation of property ϕ_1 and ϕ_2 by changing parameters α and β from 0 to 1. When α and β are 1, this is the most ideal case. In other words, the user always moves along with the regular route only and catalytic reactions always react when the conditions of them satisfy. In this case, both of properties ϕ_1 and ϕ_2 are satisfied with a probability of 1. However, if α and β are decreased (i.e., the user behaves unpleasantly and catalytic reactions do not fire even if the conditions of them satisfy), probabilities of properties ϕ_1 and ϕ_2 are also decreased. The most important aspect of this reliability analysis is that we can evaluate precisely and quantitatively how reliable this kind of ubiquitous computing scenarios are. For a particular example, quantitative evaluation of ubiquitous computing scenarios help us to consider trade-offs between the reliability of ubiquitous computing scenarios and the cost of implementation for the satisfaction of the reliability by changing parameters of probabilities, such as α and β in this case study. In this case, if β is closer to 1, this means we may need more costs for the implementations.



(i) Property ϕ_1



(ii) Property ϕ_2

Figure 5.2 Results of Experiments

5.5 Summary

In this chapter, we proposed the method of reliability analysis for ubiquitous computing scenarios described by P-CCRN. By our method, we can discuss the reliability of ubiquitous computing scenarios even if these scenarios are in rather practical situation than ideal cases. Reliability analyses are important because these analyses are quantitative, and this means we can discuss about trade-offs between the reliability and the cost for the satisfaction of the reliability. Once we design a ubiquitous computing scenario by P-CCRN, we may actually implement this which usually takes the cost. In that sense, our approach for reliability analysis is not only theoretical but also practical.

Chapter 6

Conclusions and Open Problems

In this chapter, we conclude our contributions presented in this thesis.

6.1 Concluding Remarks

In chapter 3, we proposed a verification method of applications which is described by a CCRN using model checking. Using our framework, various properties of application scenarios of ubiquitous computing can be discussed by logic such as LTL. Our framework actually helps the designers to debug ubiquitous computing application scenarios. With our framework, the cost of detecting any counterexamples is much reduced compared to hand simulation. These contributions are

important as the first step of the formularization to verify ubiquitous computing scenarios.

In chapter 4, we proposed a method to reduce a CCRN verification problem to a symbolic model checking problem. Our proposal method enables us to verify more large scale ubiquitous computing scenarios in realistic time and memory space. To show that symbolic model checking is useful approach to verify ubiquitous computing scenarios, we conducted experiments using a museum example of ubiquitous computing scenario as a case study. Additionally, we also show that bounded model checking is also useful approach especially to detect faults of ubiquitous computing scenario.

In chapter 5, we proposed the method of reliability analysis for ubiquitous computing scenarios described by P-CCRN. By our method, we can discuss the reliability of ubiquitous computing scenarios even if these scenarios are in rather practical situation than ideal cases. Reliability analyses are important because these analyses are quantitative, and this means we can discuss about trade-offs between the reliability and the cost for the satisfaction of the reliability. Once we design a ubiquitous computing scenario by P-CCRN, we may actually implement this which usually takes the cost. In that sense, our approach for reliability analysis is not only theoretical but also practical.

With above these three propositions, we established a formal verification framework for verifying appropriateness of ubiquitous computing scenario design. Formal verification can find design-related faults of objects before actual implementations of them. In particular, objects which take roles of social infrastructures such as ubiquitous

computing scenarios typically need many of costs for their implementations. In this sense, it is important to conduct comprehensive fault detections of these ubiquitous computing scenarios during a step of their design by using formal verification. In addition to these formal verification, we also showed the method to evaluate the reliability of ubiquitous computing scenarios with considerations of uncertainties including external interferences. This method extends our theoretical contributions to the more practical applications in the sense that we can discuss trade-offs between the reliability of ubiquitous computing scenarios and their implementation costs by using our method. From these facts about our contributions, we conclude that our contributions are worthwhile on both theoretically and practically.

6.2 Open Problems and Future Directions

As future directions, there are several challenges.

In chapter 3, we have considered the case of a single user and we believe this is enough to verify the connectivity of mutual related multiple federations among SOs. We assumed that the designers has already understood the notion of a catalytic reaction network. But we need to develop more designer-friendly tools such as graphical user interfaces to generate a CCRN in future work. To consider more practical situations, we will also consider the case of multiple users. Namely, more than one user move around, carrying SOs simultaneously. This will enable us to consider more complex applications of ubiquitous computing.

In chapter 4, we improved the scalability of scenario verification

of CCRN. However, we still continue to improve the scalability of our method. To do so, we consider to reduce variables in variable vector $\boldsymbol{s}$.

In chapter 5, we considered interferences of ubiquitous computing scenario. However, we still need to analyze more various kinds of ubiquitous computing scenarios assuming more various interferences including possible situations in real places.

Acknowledgements

I would like to thank my supervisor Prof. Shin-ichi Minato. He not only advised me many of technical things for this research, but also encouraged me all the time. I also would like to thank Prof. Hiroki Arimura and Assoc. Prof. Ichigaku Takigawa for their valuable comments and supports to write this thesis. Also I would like to thank Prof. Yuzuru Tanaka. He has not explicitly shown me how to conduct research, but I have learned much from him what a researcher should be like, and what kind of a philosophy I should have. I would like to thank my co-authors, especially, Masakazu Ishihata for his meaningful discussion and advice. I would like to thank the secretary of Large-Scale Knowledge Processing Laboratory Sachiko Soma and the secretary of JSPS KAKENHI KIBAN (S) Discrete Structure Manipulation System Project Yukie Watanabe. They supported me in many complicated office tasks of the university. I also would like to thank all colleagues and alumni of Large-Scale Knowledge Processing Laboratory (formerly called Knowledge Media Laboratory) and Meme Media Laboratory. Particularly, I would like to thank Hajime Imura. He has supported my mind for my entire laboratory life of almost a decade.

Finally, I would like to express my special thanks to my mother Satomi Minoda and my father Mitsuhiko Minoda for being patient and having understanding during my very long student life of a dozen years.

Related Publications

[a] 蓑田 玲緒奈，湊 真一．記号モデル検査によるスマートオブジェクトの近接連携シナリオの効率的な検証．電子情報通信学会論文誌（D），J101.D(3), 2018. (to appear, written in Japanese text)

[b] Reona Minoda and Shin-ichi Minato. Verifying Scenarios of Proximity-Based Federations among Smart Objects through Model Checking and Its Advantages. *IEICE Transactions on Information and Systems*, E100.D(6):1172–1181, 2017.

[c] Reona Minoda, Masakazu Ishihata, and Shin-ichi Minato. Probabilistic CCRN: Reliability Analysis of Ubiquitous Computing Scenarios Using Probabilistic Model Checking. In *Proceedings of The Eleventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2017)*, pages 85–91, 2017.

[d] Reona Minoda and Shin-ichi Minato. Efficient Scenario Verification of Proximity-based Federations among Smart Objects through Symbolic Model Checking. In *Proceedings of the 7th International Joint Conference on Pervasive and Embedded Computing*

and Communication Systems (PECCS/PEC 2017), pages 13–21, 2017.

[e] Reona Minoda, Yuzuru Tanaka, and Shin-ichi Minato. Verifying Scenarios of Proximity-based Federation among Smart Objects through Model Checking. In *Proceedings of The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016)*, pages 65–71, 2016.

Copyright Notice

Some materials such as figures and tables in this thesis are used with permissions issued from publishers of articles above.

Bibliography

[1] IEEE Standard for a High-Performance Serial Bus. *IEEE Standard 1394-2008*, pages i–954, 2008.

[2] Rajeev Alur and David L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[3] Andrea Bianco and Luca de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS '95)*, pages 499–513, 1995.

[4] Armin Biere, Alessandro Cimatti, Edmund Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *Proceedings of 5th International Conference on Tools and Algorithms for the Analysis and Constructions of Systems (TACAS '99)*, number 97, pages 193–207, 1999.

[5] Randal E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, 1986.

[6] Jerry R. Burch, Edmund M. Clarke, David E. Long, Kenneth L. McMillan, and David L. Dill. Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th ACM/IEEE De-*

sign Automation Conference (DAC '90), pages 46–51, New York, NY, USA, 1990. ACM.

[7] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and Lain-Jinn Hwang. Symbolic model checking: 10^{20} States and beyond. *Information and Computation*, 98(2):142–170, 1992.

[8] Roberto Cavada, Alessandro Cimatti, Charles Arthur Jochim, Gavin Keighren, Emanuele Olivetti, Marco Pistore, Marco Roveri, and Andrei Tchaltsev. NuSMV 2.6 User Manual. `http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf`, 2016 (accessed December 14, 2017).

[9] Alessandro Cimatti, Edmund Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2: An opensource tool for symbolic model checking. *Computer Aided Verification*, 2404:359–364, 2002.

[10] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of Logics of Programs Workshop*, pages 52–71, Berlin, Heidelberg, 1982. Springer Berlin Heidelberg.

[11] Luca de Alfaro, Marta Kwiatkowska, Gethin Norman, David Parker, and Roberto Segala. Symbolic Model Checking of Probabilistic Processes Using MTBDDs and the Kronecker Representation. In *Proceedings of Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, pages 395–410, 2000.

[12] Rolf Drechsler and Ulrich Kühne, editors. *Formal Modeling and Verification of Cyber-Physical Systems*. Springer Fachmedien Wiesbaden, Wiesbaden, 2015.

[13] Alan Freier, Phil Karlton, and Paul Kocher. RFC6101: The Secure Sockets Layer (SSL) Protocol Version 3.0. `https://www.rfc-editor.org/rfc/rfc6101.txt`, 2011 (accessed January 13th, 2018).

[14] Safa Guellouz, Adel Benzina, Mohamed Khalgui, and Georg Frey. ZiZo : A Complete Tool Chain for the Modeling and Verification of Reconfigurable Function Blocks. In *Proceedings of 10th International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM 2016)*, pages 144–151, 2016.

[15] Arnd Hartmanns, Sean Sedwards, and Pedro R. D'Argenio. Efficient Simulation-Based Verification of Probabilistic Timed Automata. In *Proceedings of the 2017 Winter Simulation Conference (WSC 2017)*, pages 1419–1430, 2017.

[16] Vasiliki Hartonas-Garmhausen, Sergio Campos, and Edmund M. Clarke. ProbVerus: Probabilistic Symbolic Model Checking. In *Proceedings of 5th International AMAST Workshop on Real-Time and Probabilistic Systems (ARTS '99)*, pages LNCS 1601, 96–110, 1999.

[17] Ichiro Hasuo. Metamathematics for Systems Design. *New Generation Computing*, 35(3):271–305, 2017.

[18] Paula Herber, Marcel Pockrandt, and Sabine Glesner. STATE – A SystemC to Timed Automata Transformation Engine. In *Proceedings of IEEE 17th International Conference on High Performance Computing and Communications, IEEE 7th International Symposium on Cyberspace Safety and Security, and IEEE 12th International Conference on Embedded Software and Systems (HPCC/CSS/ICESS 2015)*, pages 1074–1077, 2015.

[19] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser,

and Siegle Markus. A Markov chain model checker. *Proceedings of International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2000)*, pages 347–362, 2000.

[20] Gerard J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, 1997.

[21] Matti Jarvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The International SAT Solver Competitions. *Ai Magazine*, 33(1):89–94, 2012.

[22] Jérémie Julia and Yuzuru Tanaka. Proximity-based federation of smart objects. *Journal of Intelligent Information Systems*, 46(1):147–178, 2016.

[23] Stuart Kauffman. *Investigations*. Oxford University Press, Oxford New York, 2002.

[24] Saul A. Kripke. Semantical Analysis of Modal Logic I Normal Modal Propositional Calculi. *Zeitschrift für Mathematische Logik und Grundlagen der Mathematik*, 9(5-6):67–96, 1963.

[25] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In G Gopalakrishnan and S Qadeer, editors, *Proceedings of 23rd International Conference on Computer Aided Verification (CAV 2011)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[26] Marta Kwiatkowska, Gethin Norman, and Jeremy Sproston. Probabilistic model checking of deadline properties in the IEEE 1394 FireWire root contention protocol. *Formal Aspects of Computing*, 14(3):295–318, 2003.

[27] Jeff Magee and Jeff Kramer. *Concurrency State Models and Java Programs*. John Wiley and Sons, 1999.

[28] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, 1993.

[29] Robin Milner. Theories for the global ubiquitous computer. In *In Proceedings of 7th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2014)*, pages 5–11, 2004.

[30] John C. Mitchell, Vitaly Shmatikov, and Ulrich Stern. Finite-state Analysis of SSL 3.0. In *Proceedings of the 7th Conference on USENIX Security Symposium (SSYM '98)*, page 16, Berkeley, CA, USA, 1998. USENIX Association.

[31] Amir Pnueli. The temporal logic of programs. *Proceedings of 18th Annual Symposium on Foundations of Computer Science (SFCS 1977)*, pages 46–57, 1977.

[32] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in CESAR. In *Proceedings of 5th International Symposium on Programming*, pages 337–351, 1982.

[33] A. Prasad Sistla and Edmund M. Clarke. The complexity of propositional linear temporal logics. *Journal of the ACM*, 32(3):733–749, 1985.

[34] Yan Sun, Tin-Yu Wu, Xinming Li, and Mohsen Guizani. A Rule Verification System for Smart Buildings. *IEEE Transactions on Emerging Topics in Computing*, 5(3):367–379, 2017.

[35] Yuzuru Tanaka. Proximity-based federation of smart objects: liberating ubiquitous computing from stereotyped application scenarios. In *In Proceedings of 14th International Conference on Knowledge-Based and Intelligent Information and Engineering Systems (KES 2010)*, pages 14–30. Springer, 2010.

[36] Moshe Y. Vardi. Automatic Verification of Probabilistic Concurrent Finite-State Systems. In *26th Annual Symposium on Foundations of Computer Science (FOCS '85)*, pages 327–338, 1985.

[37] Mark Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, 1991.

[38] Chang Xu and Shing-Chi Cheung. Inconsistency Detection and Resolution for Context-aware Middleware Support. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE 2005)*, pages 336–345, 2005.