Efficient Scenario Verification of Proximity-based Federations among Smart Objects through Symbolic Model Checking

Reona Minoda and Shin-ichi Minato
Graduate School of Information Science and Technology, Hokkaido University, Sapporo, Japan

Keywords: Ubiquitous Computing, Catalytic Reaction Network, Formal Verification, Symbolic Model Checking, Smart Object.

Abstract: Verification of ubiquitous computing (UC) scenarios enables us to detect design-related faults of UC applications before we actually implement them. In this paper, we propose a verification framework of context catalytic reaction network (CCRN), which is a description model of UC 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 UC 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.

1 INTRODUCTION

Today, we are surrounded with various devices with computation and communication capabilities. In this paper, we call these devices *"smart objects (SO)"*. SOs include personal computers (PCs), mobile phones, sensor devices, embedded computers and radio frequency identifier (RFID) tags. But we can also treat physical things like foods, medicine bottles and cups as SOs by embedding RFID tags in those. Here we use the term *federation* to denote the definition and execution of interoperation among resources that 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 other, a phone rings to inform a user to *warn not eat them together*. This phenomenon is a federation. Indeed, we can also consider federations related to other SOs and these federation may be involved in each other. We call these federation *"ubiquitous computing application scenarios (UC scenario)"* (see Fig.1).

Yuzuru Tanaka pointed out that the importance of UC scenario analyses and discussions because this liberates us from stereotyped UC scenarios such as *"location transparent service continuance"* (i.e., a user can use a service wherever the user goes) and *"context-aware service provision"* (i.e., a user can use different kinds of services depending on where the user is) (Tanaka, 2010). Julia and Tanaka proposed catalytic reaction network as a description model of UC scenario (Julia and Tanaka, 2016).

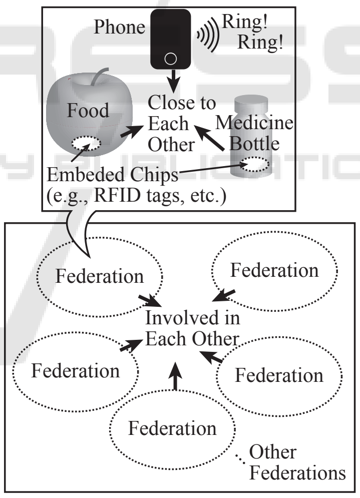

Figure 1: Example of Ubiquitous Computing Application Scenario.

In our previous work, we extended a catalytic reaction network model to *context catalytic reaction network* (CCRN) by adding a discrete structure called "segment graph". Using a CCRN model, we formalized a UC scenario verification problem and proposed a method to reduce UC scenario verification problems to model checking problems (Minoda et al., 2016). This method enabled us to discuss the properties of UC scenarios. These properties are included as follows.

- Determining whether a property described in a linear temporal logic (LTL) specification (e.g., a particular federation *finally* occurs after some conditions) is satisfied or not in the given UC scenario described by CCRN.
- Showing a counterexample if there are any cases violating the properties described above.

Most important aspect of this kind of verification is that these discussions about properties of a given UC 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 paper, 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 (Burch et al., 1992). We show the method how to take advantage of symbolic model checking techniques when we reduce UC scenario verification problems to model checking problems. We also show experimentally that our method can verify larger UC scenario verification problems compared to our previous naive approach. As a result, this method enables us to discuss more practical UC 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 (Biere et al., 1999). 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 paper, we also evaluate experimentally how practical is bounded model checking for UC scenario verification problems.

The rest of this paper is organized as follows. The rest of this section introduces related works of our research. Section 2 provides preliminaries of this paper, such as basic definitions and notations including CCRN, symbolic model checking and bounded model checking. Using them, we propose the verification method of CCRN using symbolic model checking in section 3. In section 4, we evaluate our verification method experimentally by using both symbolic model checking and bounded model checking. Finally, we conclude our contributions in section 5.

1.1 Related Works

In this section, we enumerate related researches in the field of ubiquitous computing.

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* (Alur and Dill, 1994) as a state transition model to conduct formal verifications of given systems' properties (Drechsler and Kühne, 2015).

Context Inconsistency Detection. In the field of ambient computing, Xu and Cheung propose a method of context inconsistency detection (Xu and Cheung, 2005). 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*.

2 PRELIMINARIES

In this section, we give definitions and notations which is necessary for this paper.

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 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.

2.2 Catalytic Reaction Network

A catalytic reaction network is originally proposed by Stuart Kauffman in the field of biology to analyze protein metabolism (Kauffman, 2002). Based on

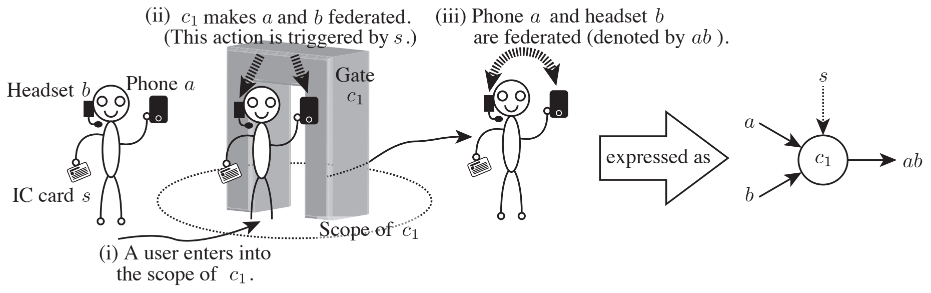
(ii) c_1 makes a and b federated.
(This action is triggered by s.)
(iii) Phone a and headset b are federated (denoted by ab).
Headset b
Phone a
Gate c_1
IC card s
Scope of c_1
(i) A user enters into the scope of c_1.
expressed as
s
a
b
c_1
ab

Figure 2: Example of a Catalytic Reaction.

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 (Tanaka, 2010). In this paper, 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. We assume that only the scopes of contexts are considered instead. In other words, we consider that the catalytic reaction is occurred if all required SOs just enter into the scope of the corresponding context.

Figure 2 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.

In catalytic reaction networks, there are four types of catalytic reactions as we show in Fig. 3. We categorize these four types of reactions into two groups. One group is the *composition* reaction group (Fig. 3 (i) and (ii)), the other group is the *decomposition* reaction group (i.e., Fig. 3 (iii) and (iv)). A catalytic reaction of Fig. 2 is a type (i) catalytic reaction. We also consider the catalytic reaction without a stimulus such as Fig. 3 (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,

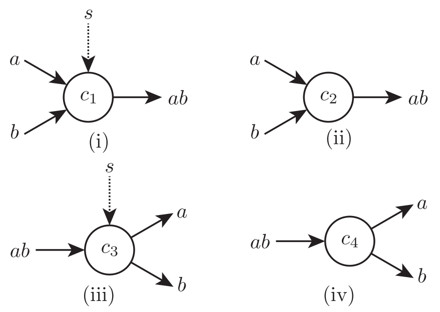

Figure 3: Four Types of a Catalytic Reactions.

we consider the decomposition reactions such as Fig. 3 (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 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*
- $(|M \cap N| + 1 = |N|, |M| > |N|) \vee$
  $(|M \cap N| + 1 = |M|, |M| < |N|) \qquad (*)$

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 cat-

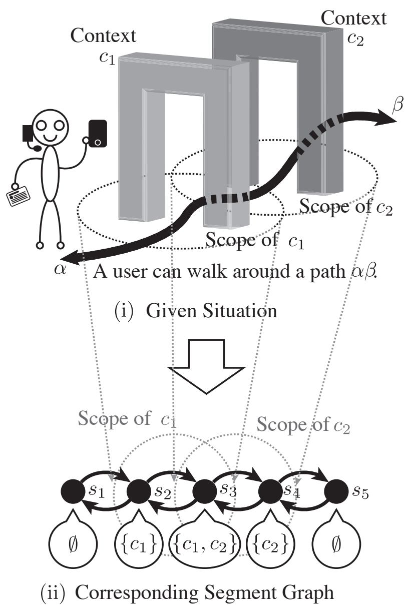

Figure 4: Example of Segment Graph.

alytic reaction of Fig. 3 (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 (Catalytic Reaction Network)**.** *A catalytic reaction network is a set of catalytic reactions.*

2.3 Context Catalytic Reaction Network

This section describes a segment graph and a CCRN.

2.3.1 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. 4 (i), there are contexts c_1 and c_2 and these scopes have an *overlap*. A user can walk around the path αβ shown in Fig. 4 (i). This situation can be represented as a segment graph shown in Fig. 4 (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. 4 (i) including overlaps of scopes of contexts can be represented as a discrete structure.

Now we define a segment graph as follows.

Definition 3 (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,*
- *$E \subseteq S \times S$ 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.*

2.3.2 Context Catalytic Reaction Network

A context catalytic reaction network (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 4 (Context Catalytic Reaction Network)**.** *A context catalytic reaction network (CCRN) is a tuple $(O, C, R, G, L_{FIX}, l_0)$, where*

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

2.4 Model Checking

A model checking is a method to verify a property of a state transition system. It has been often used in various fields, which ranges from electronic-circuit-design verification (Burch et al., 1990) to secure-network-protocol (e.g., Secure Sockets Layer (SSL) protocol) design verification (Mitchell et al., 1998). 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 *linear temporal logic (LTL)* and *computational tree logic (CTL)*. In this paper, we use LTL to describe the property of the Kripke structure.

2.4.1 Kripke Structure

Before we look on the detail of a model checking, we give the definition of a Kripke structure (Kripke, 1963) which is necessary for a modal logic and a model checking.

Definition 5 (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.*

2.4.2 Linear Temporal Logic

Linear temporal logic (LTL) is one of the most well-known modal logic. LTL was first proposed for the formal verification of computer programs by Amir Pnueil in 1977 (Pnueli, 1977). First, we give a definition of LTL syntax.

Definition 6 (Linear Temporal Logic Syntax)**.** *Let AP be a set of atomic propositions, a linear temporal logic formula ϕ is defined by the following syntax recursively.*

$$\phi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi \vee \phi \mid \boldsymbol{X}\,\phi \mid \boldsymbol{G}\,\phi \mid \boldsymbol{F}\,\phi \mid \phi\,\boldsymbol{U}\,\phi \quad (1)$$

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 7 (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

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

Now we focus on the semantics of linear temporal logic. First, we define the binary satisfaction relation, denoted by $\models$, for LTL formulae. This satisfaction 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 $\left[(\exists i.(M, \pi^i \models \psi)) \wedge (\forall j < i.(M, \pi^j \models \phi))\right]$

2.4.3 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. A model checking problem is formally stated as follows.

Definition 8 (Model Checking Problem)**.** *Given a desired property described by a modal logic formula ϕ (in this paper, we use LTL) and a Kripke structure M, a model checking problem is a decision problem whether the following formula*

$$\forall \pi.(M, \pi \models \phi) \quad (3)$$

is satisfied or not. 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.

There are several implementations of the model checking verifier such as **S**imple **P**romela **IN**terpreter (SPIN) (Holzmann, 1997) and **L**abel **T**ransition **S**ystem **A**nalyzer (LTSA) (Magee and Kramer, 1999).

2.4.4 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* (Burch et al., 1992). 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) (Bryant, 1986) 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 $\boldsymbol{s}$.

$$S(\boldsymbol{s}) = \begin{cases} \text{True} & \text{if } \boldsymbol{s} \in S \\ \text{False} & \text{otherwise} \end{cases} \quad (4)$$

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 $T(\boldsymbol{s}, \boldsymbol{s'})$.

$$T(\boldsymbol{s}, \boldsymbol{s}') = \begin{cases} \text{True} & \text{if } (\boldsymbol{s}, \boldsymbol{s}') \in R \\ \text{False} & \text{otherwise} \end{cases} \tag{5}$$

$S(\boldsymbol{s})$ 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) (Cimatti et al., 2002).

2.4.5 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. (Biere et al., 1999). 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 (Jarvisalo et al., 2012). Basically, to conduct bounded model checking, we judge whether following Boolean function

$$I(\boldsymbol{s}_0) \wedge \left(\bigwedge_{i=0}^{k-1} T(\boldsymbol{s}_i, \boldsymbol{s}_{i+1}) \right) \wedge \neg p(\boldsymbol{s}_0, \dots, \boldsymbol{s}_k) \tag{6}$$

is satisfiable or not. Note that Boolean function $p(\boldsymbol{s}_0, \dots, \boldsymbol{s}_k)$ 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 $\boldsymbol{s}_0, \dots, \boldsymbol{s}_k$ 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.

3 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 $(O, C, R, (S, E, F), L_{\text{FIX}}, l_0)$ 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 $(O, C, R, (S, E, F), L_{\text{FIX}}, l_0)$. One is a state of mobile SOs' location. A set of mobile SOs O_{MOB} are defined as $O \setminus \{o \in O \mid \exists s \in S.((o, s) \in L_{\text{FIX}}\}$. 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 "$\mathit{segment} = s$", and a state that federated SOs o_f are existing as "$\mathit{fed}(o_f) = \text{True}$" respectively. Using those, we define a variable vector $\boldsymbol{s}$ as follows.

$$\boldsymbol{s} = (\mathit{segment}, \underbrace{\mathit{fed}(o_f), \mathit{fed}(o_f'), \cdots}_{2^{|O|}-1}) \tag{7}$$

In this paper, 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.

$$\begin{aligned} I(\boldsymbol{s}) = & (\mathit{segment} = l_0) \\ & \wedge \left(\bigwedge_{o_f \in 2^O \setminus \{\}, |o_f| = 1} \mathit{fed}(o_f) = \text{True} \right) \\ & \wedge \left(\bigwedge_{o_f \in 2^O \setminus \{\}, |o_f| > 1} \mathit{fed}(o_f) = \text{False} \right) \end{aligned} \tag{8}$$

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.

$$S_e \triangleq (\mathit{segment} = s) \wedge (\mathit{segment}' = s') \tag{9}$$

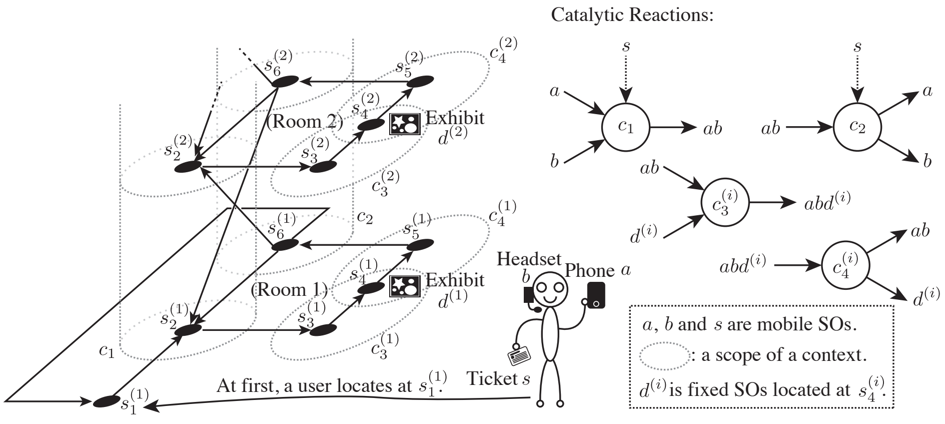

Figure 5: Museum Example of CCRN.

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.

$$R_\emptyset \triangleq \bigwedge_{o_f \in 2^O \setminus \{\}} \left(fed(o_f) = fed'(o_f) \right) \tag{10}$$

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

$$R_r \triangleq \bigwedge_{o_f \in N} \left(fed'(o_f) = \text{True} \right) \wedge \bigwedge_{o_f \in M} \left(fed'(o_f) = \text{False} \right) \wedge \bigwedge_{o_f \in (2^O \setminus \{\}) \setminus (M \cup N)} \left(fed(o_f) = fed'(o_f) \right) \tag{11}$$

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

$$RC_r \triangleq \bigwedge_{o_f \in M} \left(fed(o_f) = \text{True} \right) \tag{12}$$

A set of catalytic reactions $R_{\text{app}}(e)$ 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.

$$R_{\text{app}}(e) \triangleq \{(c, M, N) \in R \mid c \in F(s'), O(c) \supseteq \bigcup M\} \text{ where}$$
$$O(c \in C) = O_{\text{MOB}} \cup \{o \in O \mid \exists s'' \in S.(c \in F(s'')), (o, s'') \in L_{\text{FIX}}\} \tag{13}$$

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

$$T_e \triangleq \begin{cases} S_e \wedge R_\emptyset & \text{if } R_{\text{app}}(e) = \emptyset \\ \bigvee_{r \in R_{\text{app}(e)}} \left(S_e \wedge RC_r \wedge R_r \right) \vee & \\ \left(S_e \wedge \neg \left(\bigvee_{r \in R_{\text{app}(e)}} RC_r \right) \wedge R_\emptyset \right) & \text{otherwise} \end{cases} \tag{14}$$

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

$$T(\boldsymbol{s}, \boldsymbol{s}') = \bigvee_{e \in E} T_e \tag{15}$$

4 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.

4.1 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.5. Left hand side of Fig.5 represents the corresponding segment graph of the CCRN. This CCRN assumes an UC 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 LTL formula.

$$\mathbf{G}(segment = s_1^{(1)} \rightarrow \mathbf{F}(segment = s_4^{(n)} \rightarrow fed(\{a, b, d^{(n)}\}) = \text{True})) \tag{16}$$

This formula means that if the user once enters the museum, the exhibit explanation of the highest floor

Table 1: Experiment Results of The Scalability Evaluation.

Problem Instance				Naive Method (Minoda et al., 2016)		Proposal Method	
n	$\|O\|$	$\|C\|$	$\|S\|$	CPU (s)	MEM (MB)	CPU (s)	MEM (MB)
1	4	4	6	0.01	13.81	0.01	13.41
2	5	6	11	0.04	16.50	0.02	15.24
3	6	8	16	0.41	48.46	0.04	19.61
4	7	10	21	8.69	656.75	0.09	31.64
5	8	12	26	273.56	13,088.76	0.24	67.85
6	9	14	31	N/A	MEM. Out	0.78	188.54
7	10	16	36	—	—	2.85	636.56
8	11	18	41	—	—	10.93	2,349.27
9	12	20	46	—	—	78.12	9,368.13
10	13	22	51	—	—	455.73	13,075.79
11	14	24	56	—	—	N/A	MEM. Out

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

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 our previous work (Minoda et al., 2016). The naive method enumerates all possible states of a given CCRN explicitly. So even if we use NuSMV2, this naive method do not take any advantage of symbolic model checking. Table. 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 UC scenarios in realistic time and costs.

4.2 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.5. However, to detect faults of the given CCRN, we omitted randomly reactions $c_3^{(i)}$ and

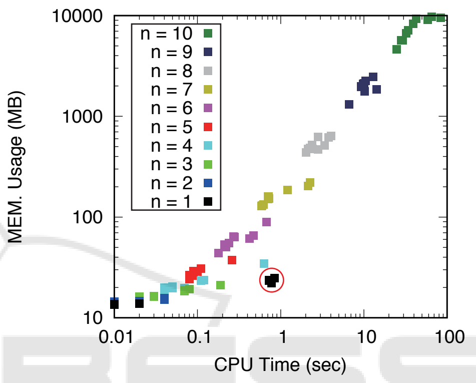

Figure 6: Computational Costs for CCRN Fault Detection.

$c_4^{(i)}$ 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.

$$\mathbf{G} \bigwedge_{i=1}^{n} \Big((\neg(segment = s_4^{(i)}) \wedge fed(\{a,b,d^{(i)}\}) = \mathrm{False}) \vee (segment = s_4^{(i)} \wedge fed(\{a,b,d^{(i)}\}) = \mathrm{True}) \Big) \quad (17)$$

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 $k = 50$.

In Fig.6, 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. From 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 24.73 sec from one of problem instances of $n = 10$. If we detect a fault from this problem instance by using symbolic model checking, it takes 318.87 sec.

Note that some cases of $n = 1$ enclosed with a circle in Fig.6 take more time and more memory space because no reactions of these cases are omitted by chance. An usual bounded model checking verifier 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.

5 CONCLUSIONS

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 UC scenarios in realistic time and memory space. To show that symbolic model checking is useful approach to verify UC scenarios, we conducted experiments using a museum example of UC scenario as a case study. Additionally, we also show that bounded model checking is also useful approach especially to detect faults of UC scenario. As our future work, we continue to improve the scalability of our method. To do so, we consider to reduce variables in variable vector $\boldsymbol{s}$.

ACKNOWLEDGEMENTS

This work was partly supported by JSPS KAKENHI (S) Grant Number 15H05711.

REFERENCES

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

Biere, A., Cimatti, A., Clarke, E., and Zhu, Y. (1999). Symbolic model checking without BDDs. In *Tools and Algorithms for the Analysis and Constructions of Systems*, number 97, pages 193–207.

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

Burch, J., Clarke, E., McMillan, K., Dill, D., and Hwang, L. (1992). Symbolic model checking: 10^{20} States and beyond. *Information and Computation*, 98(2):142–170.

Burch, J. R., Clarke, E. M., McMillan, K. L., and Dill, D. L. (1990). Sequential circuit verification using symbolic model checking. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, DAC '90, pages 46–51, New York, NY, USA. ACM.

Cimatti, A., Clarke, E., and Giunchiglia, E. (2002). Nusmv 2: An opensource tool for symbolic model checking. *Computer Aided Verification*, 2404:359–364.

Drechsler, R. and Kühne, U., editors (2015). *Formal Modeling and Verification of Cyber-Physical Systems*. Springer Fachmedien Wiesbaden, Wiesbaden.

Holzmann, G. (1997). The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295.

Jarvisalo, M., Le Berre, D., Roussel, O., and Simon, L. (2012). The International SAT Solver Competitions. *Ai Magazine*, 33(1):89–94.

Julia, J. and Tanaka, Y. (2016). Proximity-based federation of smart objects. *Journal of Intelligent Information Systems*, 46(1):147–178.

Kauffman, S. (2002). *Investigations*. Oxford University Press, Oxford New York.

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

Magee, J. and Kramer, J. (1999). *Concurrency State Models and Java Programs*. John Wiley and Sons, New York, New York, USA.

Minoda, R., Tanaka, Y., and Minato, S.-i. (2016). Verifying Scenarios of Proximity-based Federation among Smart Objects through Model Checking. In *Proceedings of UBICOMM 2016 The Tenth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, number c, pages 65–71.

Mitchell, J. C., Shmatikov, V., and Stern, U. (1998). Finite-state Analysis of SSL 3.0. In *Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7*, SSYM'98, page 16, Berkeley, CA, USA. USENIX Association.

Pnueli, A. (1977). The temporal logic of programs. *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57.

Tanaka, Y. (2010). Proximity-based federation of smart objects: liberating ubiquitous computing from stereotyped application scenarios. In *Knowledge-Based and Intelligent Information and Engineering Systems*, pages 14–30. Springer.

Xu, C. and Cheung, S. C. (2005). Inconsistency Detection and Resolution for Context-aware Middleware Support. *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 336–345.