IEICE TRANS. INF. & SYST., VOL.E100–D, NO.6 JUNE 2017

PAPER Special Section on Formal Approach

Verifying Scenarios of Proximity-Based Federations among Smart Objects through Model Checking and Its Advantages*

Reona MINODA[†a)], ***Student Member and*** **Shin-ichi MINATO**[†b)], ***Senior Member***

SUMMARY This paper proposes a formal approach of verifying ubiquitous computing application scenarios. Ubiquitous computing application scenarios assume that there are a lot of devices and physical things with computation and communication capabilities, which are called smart objects, and these are interacted with each other. Each of these interactions among smart objects is called "federation", and these federations form a ubiquitous computing application scenario. Previously, Yuzuru Tanaka proposed "a proximity-based federation model among smart objects", which is intended for liberating ubiquitous computing from stereotyped application scenarios. However, there are still challenges to establish the verification method of this model. This paper 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 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.

key words: *ubiquitous computing, catalytic reaction network, formal verification, model checking, smart object*

1. Introduction

Today, 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 embeded 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 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). 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 [2].

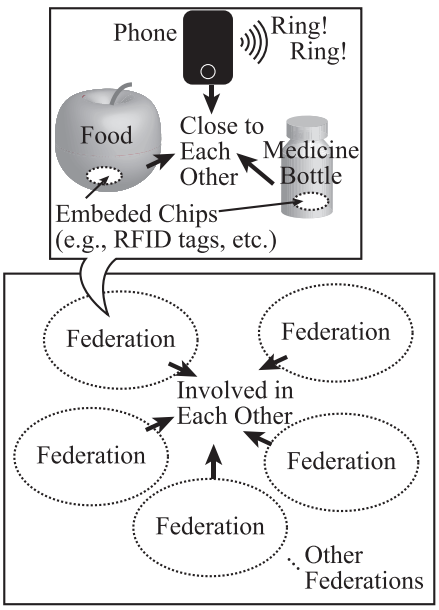


Fig. 1 Example of ubiquitous computing application scenario

Since Weiser proposed the notion of ubiquitous computing, it has been almost quarter of century. In the meantime, a lot of different 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 [3]. One is "*location transparent service continuance*" (i.e., a user can use a service wherever the user goes). The other one is "*context-aware service provision*" (i.e., a user can use different kinds of services depending on where the user is). Robin Milner thought that the lack of models for describing ubiquitous computing application scenarios caused to prevent from considering various types of application scenarios [4]. Besides, ac-

Manuscript received August 29, 2016.
Manuscript revised December 18, 2016.
Manuscript publicized March 7, 2017.

[†]The authors are with the Graduate School of Information Science and Technology, Hokkaido University, Sapporo-shi, 060–0814 Japan.

*This paper is the extended version of our work which was published in proceedings of UBICOMM 2016 [1].

a) E-mail: minoda@meme.hokudai.ac.jp
b) E-mail: minato@ist.hokudai.ac.jp

DOI: 10.1587/transinf.2016FOP0009

cording to Milner, it is not possible to describe all concepts of ubiquitous computing by using a single model [4]. 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 [3]. 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 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 [5]. 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.

In this paper, 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 linear temporal logic (LTL) 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.

This paper is extended version of our work which was published in proceedings of UBICOMM 2016 [1]. In addition to the content of the proceeding, this paper also contains more detailed explanations about our verification method, additional considerations of case study and experimental analyses of difficulties of our target problem.

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. Using them, we define a CCRN in Sect. 3. Then, we propose the verification method of a CCRN in Sect. 4. Section 5 introduces the case study of the verification. We also evaluate the scalability of our method in Sect. 6. Finally, we summarize the results of this paper in Sect. 7.

1.1 Releated Works

1.1.1 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* [6] as a state transition model to conduct formal verifications of given systems' properties [7].

1.1.2 Context Inconsistency Detection

In the field of ambient computing, Xu and Cheung propose a method of context inconsistency detection [8]. 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.,

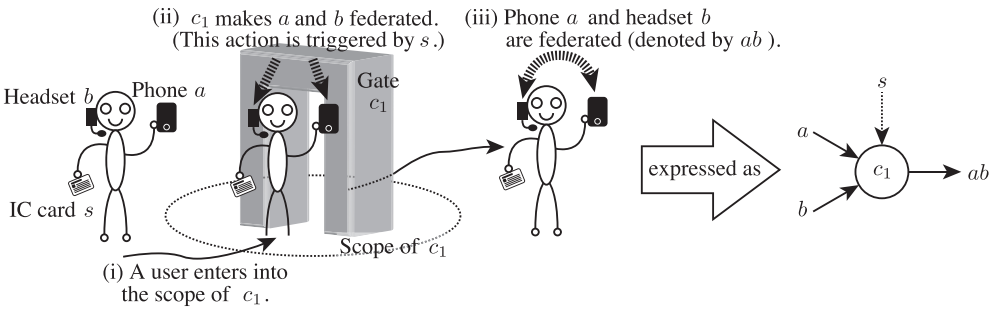


Fig. 2 Example of a catalytic reaction

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 [9]. 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 [3]. 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. Tanaka assumed that all scopes of the context and SOs involved in a catalytic reaction are considered [3]. However, as we mentioned in previous section, this causes the state explosion problem during the model checking. For this reason, in this paper, 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

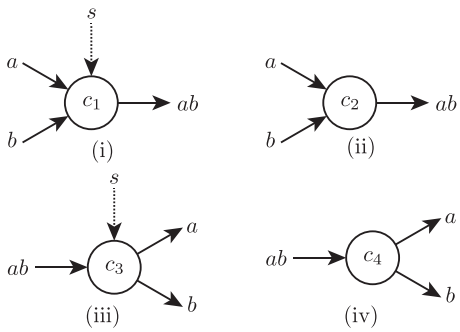


Fig. 3 Four types of a catalytic reactions

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

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. 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 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 [10] to secure-network-protocol (e.g., Secure Sockets Layer (SSL) protocol) design verification [11]. 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.

2.3.1 Kripke Structure

Before we look on the detail of a model checking, we give the definition of a Kripke structure [12] which is necessary for a modal logic and a model checking.

Definition 3 (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 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 [13]. First, we give a definition of LTL syntax.

Definition 4 (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.

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

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 5 (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 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 $\left[(M, \pi \models \phi) \vee (M, \pi \models \psi)\right]$

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 $\left[(\exists i.(M, \pi^i \models \psi)) \wedge (\forall j < i.(M, \pi^j \models \phi))\right]$

2.3.3 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 6 (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

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

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) [14], **L**abel **T**ransition **S**ystem **A**nalyzer (LTSA) [15], **New** **S**ymbolic **M**odel **V**erifier version **2** (NuSMV2) [16] and so on. In this paper, we use a model checking verifier NuSMV2.

3. Context Catalytic Reaction Network

This section introduces a segment graph and a CCRN.

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 $\alpha\beta$ 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 7 (Segment Graph): Let C be a set of contexts, a segment graph G is a tuple (S, E, F), where

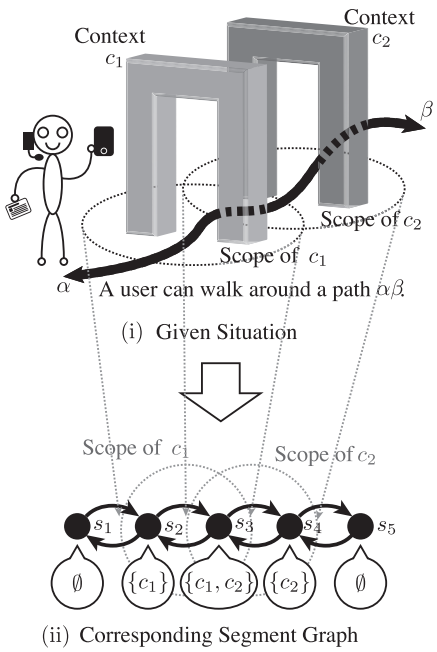


Fig. 4 Example of segment graph

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

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 8 (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 smart objects,
- 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 $O \setminus \{o \in O \mid \exists s \in S.((o, s) \in L_{\mathrm{FIX}})\}$).

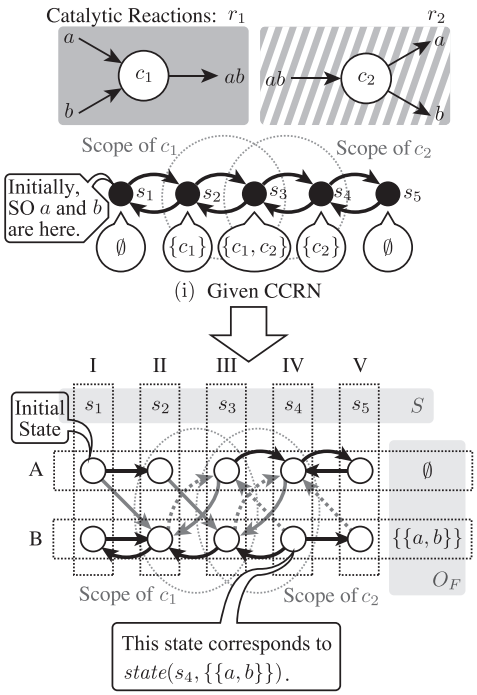
4. Verification Method of a CCRN

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



(ii) Constructed kripke structure by proposed method

Fig. 5 Mechanism of the Algorithm

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. 5 (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. 5 (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 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. 5 (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. 6 to construct a Kripke structure

Input: CCRN $(O, C, R, (S, E, F), L_{\mathrm{FIX}}, l_0)$
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$(\mathcal{S}, \mathcal{I}, \mathcal{R}, \mathcal{L})$

Fig. 6 Algorithm for transforming CCRN into 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:

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

5. 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. 7. A CCRN of this example is represented

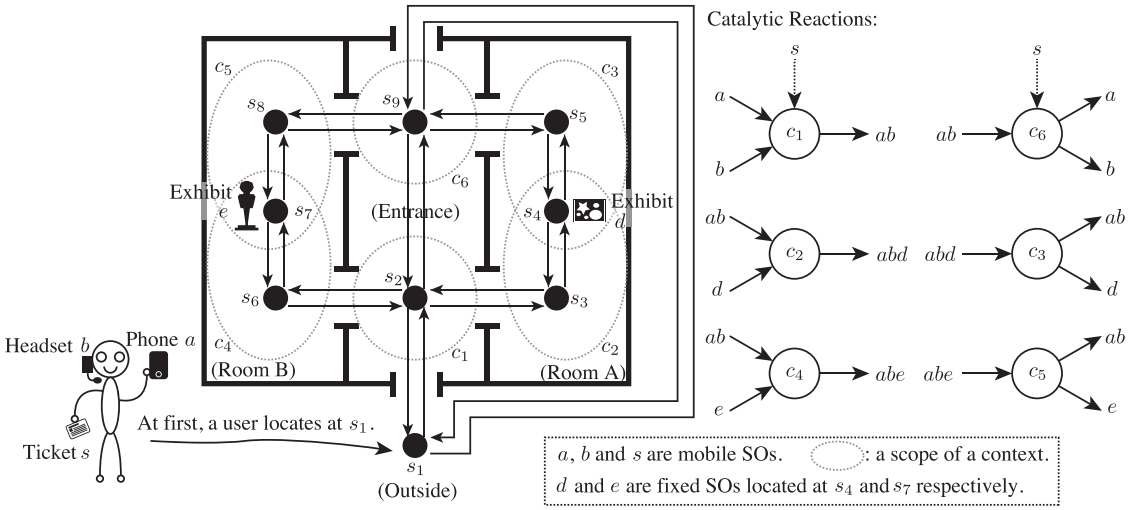


Fig. 7 Example of museum

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. 6, 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
- $\phi_3 = \mathbf{G}(\ (loc_{O_{MOB}}(s_3) \vee loc_{O_{MOB}}(s_6)) \wedge fed(\{a, b\})\ )$.

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 paper, 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. 8 straightforwardly. The size of this SMV language file generated by like this way is in proportion to the number of states, atomic propositions and transitions of given Kripke structure. NuSMV2 manual [17] 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

$\pi_{c_2} = (state(s_1, \emptyset), state(s_2, \{\{a, b\}\}), state(s_3, \{\{a, b, d\}\}),$

```
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;
```

Fig. 8 Example of a SMV language file

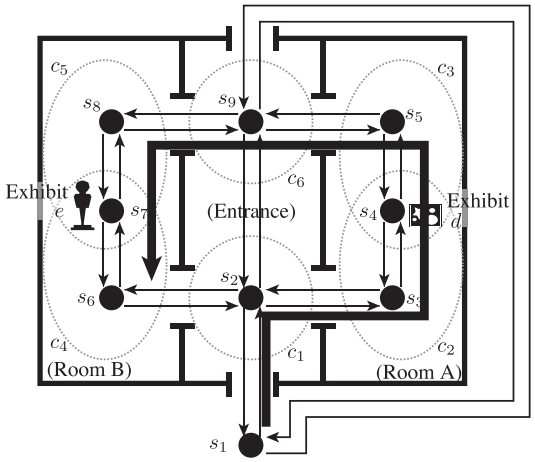


Fig. 10 Counterexample corresponding to ϕ_3 of museum example

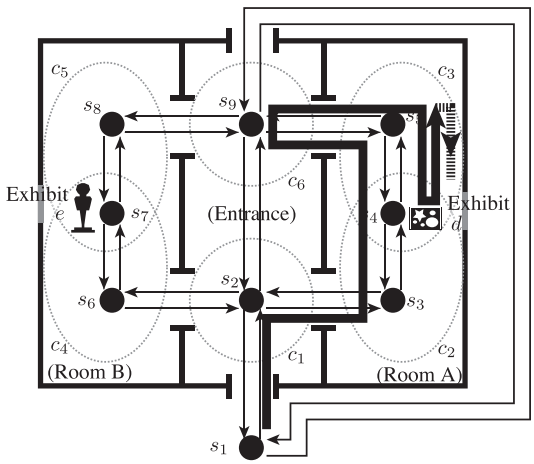


Fig. 9 Counterexample corresponding to ϕ_2 of museum example

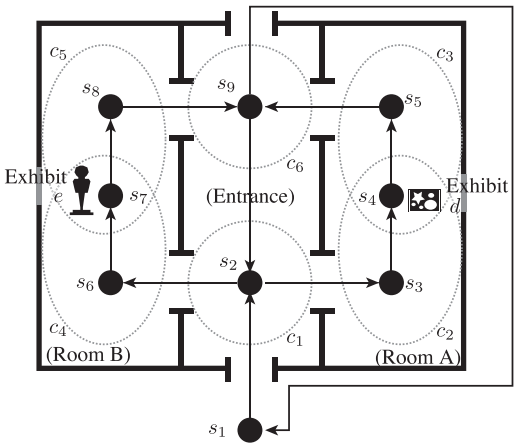


Fig. 11 Revised museum example

$$state(s_4, \{\{a, b\}\}), state(s_5, \{\{a, b\}\}), state(s_9, \emptyset),$$
$$state(s_5, \emptyset), state(s_4, \emptyset), state(s_5, \emptyset), state(s_4, \emptyset) \dots),$$

and

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

Bold lines in Fig. 9 and Fig. 10 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 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. 11 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

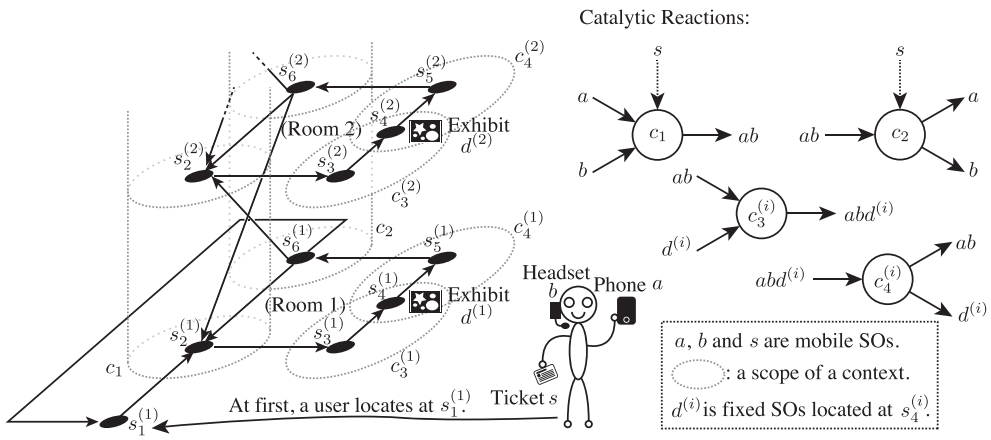


Fig. 12 Example of museum containing multiple rooms

Table 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. 7. In the revised case (Fig. 11), the number of graph walk patterns will be decreased.

Table 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.

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 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 applications, which may incur additional costs.

6. Scalability of Our Method

We evaluated the scalability of our method. To evaluate, we used a generalized example of the museum such as Fig. 12. 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 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 exponentially and we would not be able to

verify even a small case such as shown in Table 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.

7. Conclusion and Future Work

In this paper, 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. 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. These contributions are important as the first step of the formularization to verify ubiquitous computing scenarios. 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, there are two challenges. First, 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. Another challenge is about the scalability of our framework. 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.

Acknowledgements

We would like to thank Prof. Yuzuru Tanaka for advice on our work. This work was partly supported by JSPS KAKENHI (S) Grant Number 15H05711.

References

[1] R. Minoda, Y. Tanaka, and S. Minato, "Verifying Scenarios of Proximity-based Federation among Smart Objects through Model Checking," Proc. Tenth Intl. Conf. on Mobile Ubiquitous Computing, Systems, Services and Technologies, pp.65–71, 2016.
[2] M. Weiser, "The Computer for the 21st Century," Scientific American, vol.265, no.3, pp.94–104, Sept. 1991.
[3] Y. Tanaka, "Proximity-based federation of smart objects: liberating ubiquitous computing from stereotyped application scenarios," Knowledge-Based and Intelligent Information and Engineering Systems, vol.6276, pp.14–30, Springer, 2010.
[4] R. Milner, "Theories for the global ubiquitous computer," Foundations of Software Science and Computation Structures, vol.2987, pp.5–11, Springer, 2004.
[5] J. Julia and Y. Tanaka, "Proximity-based federation of smart objects," Journal of Intelligent Information Systems, vol.46, no.1, pp.147–178, Feb. 2016.
[6] R. Alur and D.L. Dill, "A theory of timed automata," Theoretical Computer Science, vol.126, no.2, pp.183–235, April 1994.
[7] R. Drechsler and U. Kühne, eds., Formal Modeling and Verification of Cyber-Physical Systems, Springer Fachmedien Wiesbaden, Wiesbaden, 2015.
[8] C. Xu and S.C. Cheung, "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, pp.336–345, 2005.
[9] S. Kauffman, Investigations, Oxford University Press, Oxford New York, 2002.
[10] J.R. Burch, E.M. Clarke, K.L. McMillan, and D.L. Dill, "Sequential circuit verification using symbolic model checking," Proceedings of the 27th ACM/IEEE Design Automation Conference, DAC '90, New York, NY, USA, pp.46–51, ACM, 1990.
[11] J.C. Mitchell, V. Shmatikov, and U. Stern, "Finite-state Analysis of SSL 3.0," Proceedings of the 7th Conference on USENIX Security Symposium - Volume 7, SSYM'98, Berkeley, CA, USA, p.16, USENIX Association, 1998.
[12] S.A. Kripke, "Semantical Analysis of Modal Logic I Normal Modal Propositional Calculi," Zeitschrift für Mathematische Logik und Grundlagen der Mathematik, vol.9, no.5-6, pp.67–96, 1963.
[13] A. Pnueli, "The temporal logic of programs," 18th Annual Symposium on Foundations of Computer Science (sfcs 1977), pp.46–57, 1977.
[14] G. Holzmann, "The model checker SPIN," IEEE Trans. Softw. Eng., vol.23, no.5, pp.279–295, May 1997.
[15] J. Magee and J. Kramer, Concurrency State Models and Java Programs, John Wiley & Sons, New York, USA, 1999.
[16] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An opensource tool for symbolic model checking," Computer Aided Verification, vol.2404, pp.359–364, 2002.
[17] R. Cavada, A. Cimatti, C.A. Jochim, G. Keighren, E. Olivetti, M. Pistore, M. Roveri, and A. Tchaltsev, "NuSMV 2.6 User Manual," FBK-irst, http://nusmv.fbk.eu/NuSMV/userman/v26/nusmv.pdf, accessed Dec. 14. 2016.

Reona Minoda received B.E. and M.S. degrees in Information Science from Hokkaido University in 2010 and 2012, respectively. He joined JST ERATO MINATO Discrete Structure Manipulation System Project as a research assistant from 2014 to 2016. He has been a research assistant at JSPS KAKENHI (S) Discrete Structure Manipulation System Project since 2016. He is a student member of IEICE.

Shin-ichi Minato received the B.E., M.E. and D.E. degrees in Information Science from Kyoto University in 1988, 1990, and 1995, respectively. He worked for NTT Laboratories from 1990 until 2004. He was a Visiting Scholar at the Computer Science Department of Stanford University in 1997. He joined Hokkaido University as an Associate Professor in 2004, and has been a Professor since October 2010. He also serves a Visiting Professor at National Institute of Informatics from 2015. He served a Research Director of JST ERATO MINATO Discrete Structure Manipulation System Project from 2009 to 2016, and now he is leading JSPS KAKENHI (S) Project until 2020. He is a senior member of IEICE, a senior member of IPSJ, and a member of IEEE and JSAI.