Situated Sensor Composition for Event-based System

Junta Koyama
Department of Social Informatics
Kyoto University
Kyoto, Japan
koyama@ai.soc.i.kyoto-u.ac.jp

Yohei Murakami
Unit of Design,
Kyoto University
Kyoto, Japan
yohei@i.kyoto-u.ac.jp

Donghui Lin
Department of Social Informatics
Kyoto University
Kyoto, Japan
lindh@i.kyoto-u.ac.jp

Abstract—With the development of IoT and the linking of sensors to the cloud, the demand for event-driven service invocation continues to increase. Moreover, as sensors will be wrapped as Web services, sensor service composition is required in order to detect complex events cost effectively. As the observed environment targeted here changes continuously, iterated sensor service composition will be required to handle all situations. In this research, we first formalize sensor service composition as the constraint-based optimization problem (COP). Moreover, we introduce temporal constraints in the detection of the temporal relationships of complex events and thus realize enhanced sensor service composition.

Keywords- ***Internet of Things (IoT), Sensor Services, Service Composition***

I. INTRODUCTION

Recently, with the development of Internet of Things (IoT), demand is rising for event-driven service invocation where the detection of an event by processing sensor outputs triggers the invocation of Web services on the cloud such as systems that detect and respond to anomalies in factories [1]. If the observed environment is fixed, such as a factory, we can realize service invocation by tightly coupling sensors to the cloud. However, loose coupling is required when the observed environment is mobile such as a vehicle. In this situation, sensors will be accessed/utilized by services and sensor services [7] are used to detect atomic events and thus complex events which consist of atomic events linked by *AND*/*OR* logical operators. If the observed environment moves continuously, sensor service composition must be iterated to support each situation because the sensor services available and the target complex events change over time. Moreover, we can avoid the cascading failure among several sensor services by repetitively composing them [2].

In the Internet of Services (IoS) domain, two approaches to service composition, satisfying the complex demands of users, exist: vertical service composition and horizontal service composition. The former creates workflows, fully integrated service processes. The latter, on the other hand, selects and links appropriate service instances from a set of services (service cluster) so as to satisfy functional requirements by using nonfunctional requirements such as Quality of Service (QoS) [3]. Each sensor service type can be several service instances and horizontal service composition for sensor services is seen as increasingly important because each instance has different nonfunctional requirements such as attributes or specifications. However, sensor services are different from conventional Web services. Service clusters cannot be defined from just functional requirements consisting of input/output as we need to consider spatio-temporal constraints on which sensor services are able to detect target events. Moreover, sensor services require service composition in Complex Event Processing (CEP) engine because they are composed based on not workflows but ECA rules [4]. Sensor services should be formed by logical operators including temporal relations in ECA rules of complex events, not rigid control structures common in workflows. In this research, we propose site-specific sensor service composition for detecting complex events by considering these characteristics and applying an existing horizontal service composition method that uses constraint-based optimization.

This paper is organized as follows. Section 2 introduces an example to explain the problem addressed by this research. In Section 3, we describe our proposed method and we apply our method to a real world scenario in Section 4. Also we implement our proposed method in Section 5. Finally, in Section 6, we introduce some works related to event-driven Service-Oriented Architecture (SOA), sensor services and sensor selection, and composition and conclude this paper in this Section 7.

II. MOTIVATING EXAMPLE

This research takes the self-driving car as its motivating example. The self-driving car will have various in-vehicle sensors, however, they are not sufficient to detect long-range events and we need to enhance service reliability by using sensors embedded in the environment. Here, we consider the scenario that a car is to stop if another vehicle approaches along a street that is 30 meters in front of the current location. The target event is "another vehicle approaches from the left side" and it can be represented as the complex event *IF Sound >= 120 db DURING Movement == True THEN Stop(self)* by using two atomic events *Sound* and *Movement*. This ECA rule represents the complex event becoming true when atomic event *Sound* is more than 120 decibels during atomic event *Movement* is True and the two atomic events can be detected by Mic sensor service and Motion sensor service. We assume that the system can judge the situation by only using *Sound* and *Movement* events. Mic sensor service is assumed that detects *Sound* events and record its sound at the same time. In this case, the recorded sound data will be sent to the sound recognition service on cloud for judging whether the event is really true or not after Mic sensor service finishes detecting the event. In the environment, there are several sensor service instances and each will have different specifications (Figure 1). We assume that the delay in detecting events is given as one of the specifications. This delay means the maximum temporal delay from when the event happens to when the sensor service detects it. In this situation, the problem

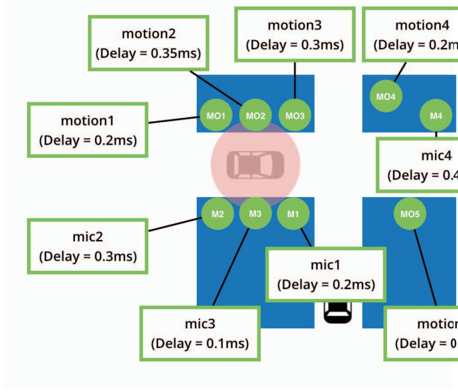


Figure 1. Example of Complex Event Detection for Self-Driving Car

we investigate in this research is which combination of sensor services should be used to realize the complex event targeted. If we consider the services in isolation, the most appropriate combination appears to be *motion3* and *mic1*. However, the complex event includes *DURING* as a temporal relation. Therefore, the target complex event may not be detected if *Movement* detection does not coincide with *Sound* detection. Since detection faults can cause critical problems in applications such as the self-driving car, false negatives must be avoided. Given the structure of the complex event, the Motion sensor service should have smaller delay than the Mic sensor service, so the combination of *motion1* and *mic2,* which satisfies this requirement, is the most appropriate. Thus, sensor services selection must satisfy the structure of the complex event.

III. Constraint-based Sensor Composition

The conventional approach to the Web service composition problem is horizontal service composition with constraint-based optimization. COP is the problem of deciding the assignment of values that maximizes or minimizes a given objective function while satisfying given constraints. Hassine et al. defined abstract Web services (components of workflows) as variables, concrete Web services (including input and output of each abstract Web service) as domains of variables, and formalized the Web service composition problem as the COP problem of selecting concrete Web services that maximize the objective function [13]. Hard constraints and soft constraints are set based on the user's QoS preferences. Moreover, hard constraints are defined to execute appropriate processing flows for each control structure of workflows because Web services are composed based on workflows. Here the objective function is a function that maximizes the user's preferences and minimize penalties on soft constraints.

In this paper, we apply horizontal service composition to the sensor service composition problem of detecting complex events. In 4.1, we formalize the sensor service composition problem as COP and extend it for detecting complex events with temporal relations in 4.2.

A. *Definition of the Problem and Events*

As we mention in Section 1, Sensor service composition will be repeated as the detected environment moves continuously and thus desired sensor services and target events will also change. Therefore, the sensor service composition problem is defined as the sequence of *k* COP_i *(1≤i≤K)* (Eq. 1).

$$COP = \{COP_1, ..., COP_i, ..., COP_K\} \quad \text{(Eq. 1)}$$

COP_i has CE_i that consists of *N* atomic events, AE_j, linked by *AND*/*OR* logical operators (Eq. 2).

$$CE_i = \{AE_1, ..., AE_j, ..., AE_N\} \quad \text{(Eq. 2)}$$

Each atomic event is defined in absolute or relative time and space (Figure 2). Absolute time and space is time and space defined with absolute geographical coordinates, absolute time and space values, and event range (Eq. 3).

$$\{type_i, coordination_i, time_i, range_i\} \quad \text{(Eq. 3)}$$
$$\{type_i, distance_i, azimuth_i, time_i, range_i\} \quad \text{(Eq. 4)}$$

Relative time and space is time and space defined by distance and azimuth from the user, relative time from the current time and event range (Eq. 4).

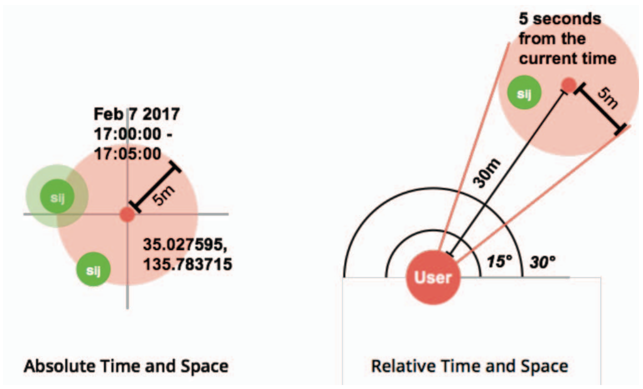


Figure 2. Absolute Time and Space and Relative Time and Space

B. *Formalization of Sensor Service Composition Problem*

Next, we define COP_i. We define the sensor service types that satisfy functional requirements for detecting atomic event AE_j as variables X_j and its set as X (Eq. 5). Then the domain of each variables is defined with the set of sensor service instances D_i being able to detect atomic event AE_j in terms of time and space and its set as D (Eq. 6).

$$X = \{X_1, ..., X_n\} \quad \text{(Eq. 5)}$$
$$D = \{D_1, ..., D_n\} \quad \text{(Eq. 6)}$$

The definition of domain D_i of each variable differs with the definition of target atomic events. The definition for absolute time and space is shown in Eq. 7 and the definition for relative time and space is shown in Eq. 8.

$$\begin{aligned} D_i = \{s_{ij}(s_{ij}.Location, s_{ij}.AvailableTime, s_{ij}.Coverage)| \\ (s_{ij}.Location \subseteq AE_i.range\ OR \\ s_{ij}.Coverage \cap AE_i.range > 0)\ AND \\ AE_i.time \subseteq s_{ij}.AvailableTime\} \end{aligned} \quad \text{(Eq. 7)}$$

$$\begin{aligned} D_i = \{s_{ij}(s_{ij}.Distance, s_{ij}.Azimuth, s_{ij}.AvailableTime, s_{ij}.Coverage)| \\ AE_i.distance - AE_i.range \leq s_{ij}.Distance \pm s_{ij}.Coverage \\ \leq AE_i.distance + AE_i.range\ AND \\ s_{ij}.Azimuth \subseteq AE_i.azimuth\ AND \\ AE_i.time \subseteq s_{ij}.AvailableTime\} \end{aligned} \quad \text{(Eq. 8)}$$

TABLE I. FOUR CONSTRAINT TYPES

Spatial Constraints	Location of Sensor Service
	Coverage (Sensing Area) of Sensor Service
	View Angle of Sensor Service
	Direction of View Angle of Sensor Service
Temporal Constraints	Delay of Sensor Service
	Response Time of Sensor Service
	Available Time of Sensor Service
Environmental Constraints	Working Temperature of Sensor Service
	Working Humidity of Sensor Service
QoS Constraints	Cost of Sensor Service
	Reputation of Sensor Service
	Data Range of Sensor Service
	Accuracy of Sensor Service

Constraint are either hard constraints (C_H) or soft constraints (C_S). Hard constraints are mainly given by service providers while soft constraints are given by the user (Eq. 9).

$$C = C_S \cup C_H \quad \text{(Eq. 9)}$$

This papers assigns constraints into four types; spatial constraints, temporal constraints, environmental constraints and QoS constraints (Table 1).

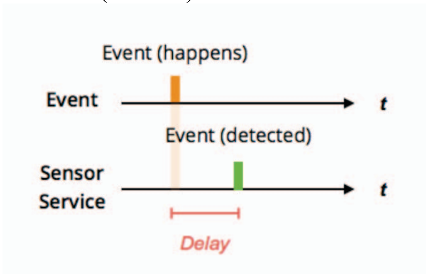


Figure 3. Maximum Delay in Event Detection, *Delay*

As shown in Section 2, the problem of the target complex event not being detected can occur due to mismatch in the detection delay of events and failure to evaluate temporal relations in complex events. However, if event detection is critical, detection faults are not allowed. Therefore, we consider how to minimize the detection delay, *Delay,* as a key goal target. *Delay* represents how much the event detection of an atomic event is offset from its occurrence (Figure 3). Although we cannot identify when an event happens in reality, it can be calculated from *Frequency* which represents the sampling interval and *Accuracy* which represents the sensor service precision as follows (Eq. 10).

$$Delay = \frac{Frequency}{Accuracy} \quad \text{(Eq. 10)}$$

In this research, we assume that *Delay* has been already specified along with other QoS attributes. We consider the minimization of this *Delay* and the summation of penalties on soft constraints. We also define aggregation functions of *Delay* for each logical operator because sensor services should be composed based on logical operators in complex events. In the case of *AND* operator, all sensor services should fire. The worst case is when atomic events happen in ascending order of *Delay* of the sensor service detecting them and the detection of atomic events is not overlapped (Figure 4). In this research, we assume this worst case and the target is to optimize the summation of *Delay*. In the case of *OR* operator, either of atomic events happens and either of sensor services fires. The worst case is when the sensor service with the biggest *Delay* fires, therefore, the target to be optimized is *Delay* of the sensor service with the biggest *Delay*. Even if other sensor service fires, its *Delay* is smaller than the biggest *Delay*. We define an objective function that takes account of the aggregation function of *Delay* and the summation of penalties on soft constraints (Eq. 11).

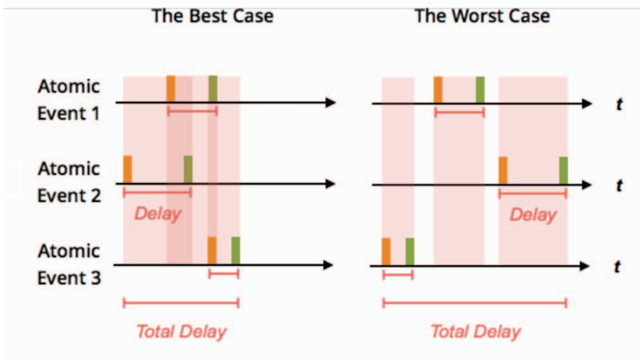


Figure 4. The Best Case and The Worst Case of *Delay* in AND relation

$$f(comb_i) = \underset{comb_i \in Cmb}{\arg\min} \left(\begin{Bmatrix} \sum_{s_{ij} \in comb_i} Delay_{s_{ij}} & \text{(AND)} \\ \max_{s_{ij} \in comb_i} Delay_{s_{ij}} & \text{(OR)} \end{Bmatrix} + \sum_{C_{Si} \in C_S} \rho_{C_{Si}} \right) \quad \text{(Eq. 11)}$$

C. *Extension of Formalization for Complex Events with Temporal Relations*

In addition to *AND*/*OR* logical operators, we consider temporal operators that represent which and when atomic events happens as operators in complex events. Therefore, we extend the COP formalization to realize sensor service composition for detecting complex events with temporal relations. Here, we take five of Allen's temporal operators [12] as temporal relations, see Figure 5 and Table 2.

As an example, in the *BEFORE* relation between two atomic events AE_1 and AE_2, AE_1 happens at first and then AE2 happens. Therefore, if *Delay* of the sensor service detecting AE_1 is greater than that one of the sensor service detecting AE_2, the detection of AE_1 will be delayed and the temporal relation is possible to be reversed (Figure 6). Therefore, *Delay* of sensor service type detecting atomic event AE_1 should set to be shorter.

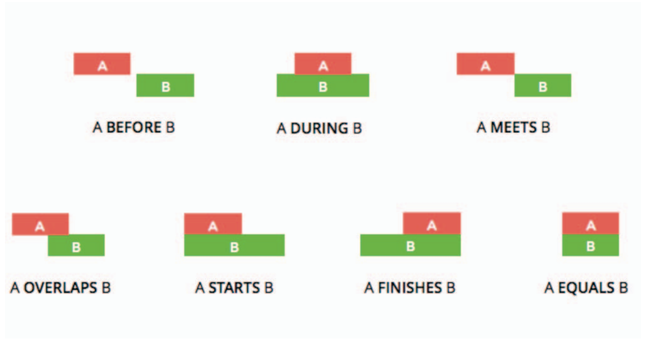


Figure 5. Allen's temporal operators

Also, we set the objective function with different aggregation function of *Delay* for each temporal relation. Accordingly, temporal orders of each atomic event can be considered. For example, in *BEFORE*, *MEET* and *OVERLAP* relation, atomic event AE_1 happens and then atomic event AE_2 happens. The end of complex event detection is after AE_2 is detected and *Delay* of sensor service detecting AE_1 is included in *Delay* of sensor service detecting AE_2. Therefore, *Delay* of sensor service detecting the atomic event happening last is set as the target to be optimized (Eq. 12). When the ECA rule of a complex event

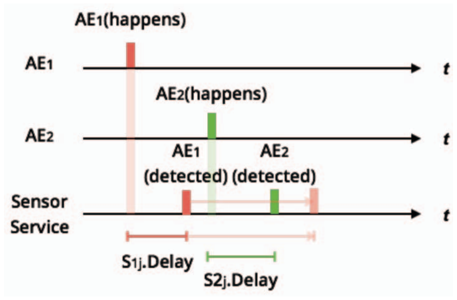


Figure 6. *Delay* in *BEFORE* relation

$$f(comb_i) = \underset{comb_i \in Cmb}{\arg\min} \left(\begin{cases} Delay_{s_{1j} \in comb_i} & \text{(before, meet, overlaps)} \\ Delay_{s_{nj} \in comb_i} & \text{(during)} \\ \sum_{s_{ij} \in comb_i} Delay_{s_{ij}} & \text{(equal)} \end{cases} \right\} + \sum_{C_{Si} \in C_S} \rho_{C_{Si}}) \qquad \text{(Eq. 12)}$$

has three or more atomic events and two or more temporal relations between them, temporal constraints and aggregation function are compiled in two ways. The first is when all temporal relations are the same between atomic events such as AE_1 *BEFORE* AE_2 *BEFORE* ... *BEFORE* AE_n. In this case, temporal constraints and aggregation function corresponding to the temporal relation can be applied. The second type is when some temporal relations are different between atomic events such as AE_1 *BEFORE* AE_2 *BEFORE* ... *DURING* AE_n. In this case, atomic events on the rule form a time line from the head of the rule to the tail of the rule. Therefore, we obtain an appropriate temporal constraint by compiling the rule backwards and rewrite the rule for atomic events that have priority. For example, if we compile the rule AE_1 *BEFORE* AE_2 *DURING* AE_2 into AE_1 *BEFORE* AE_3 backward, the sub rule AE_2 *DURING* AE_3 corresponds to the aggregation function that triggers minimization of the delay of the latter sensor service. Therefore, the priority in this rule will be given to AE_3; we can rewrite AE_2 *DURING* AE_3 simply as AE_3 and obtain the rule AE_1 *BEFORE* AE_3 so that temporal constraints and aggregation function of the *BEFORE* relation can be applied.

TABLE II. TEMPORAL CONSTRAINTS FOR EACH TEMPORAL RELATION

AE_1 ***BEFORE*** AE_2	*Delay(Sensor Service for AE_1) < Delay (Sensor Service B for AE_2)*
AE_1 ***DURING*** AE_2	*Delay(Sensor Service for AE_2) ≤ Delay (Sensor Service for AE_1)*
AE_1 ***MEETS*** AE_2	*Delay(Sensor Service A for AE_1) ≤ Delay (Sensor Service for AE_2)*
AE_1 ***OVERLAPS*** AE_2	*Delay(Sensor Service for AE_1) ≤ Delay (Sensor Service for AE_2)*
AE_1 ***EQUALS*** AE_2	*Delay(Sensor Service for AE_1) ≤ Delay (Sensor Service for AE_2)* (as C_S) *Delay(Sensor Service for AE_2) ≤ Delay (Sensor Service for AE_1)* (as C_S)

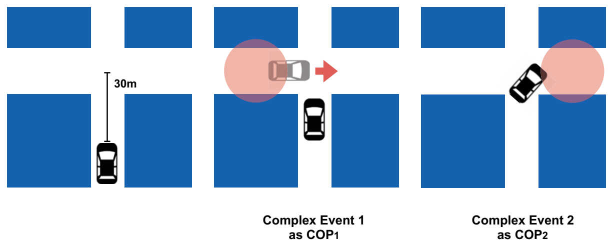


Figure 7. *Real World Scenario*

IV. Real World Scenario

In this section, we present an example using a concrete real world scenario and illustrate our proposed method by solving the problem (introduced in Section 2). First of all, we set the following scenario.

At noon, a vehicle equipped with self-driving system needs to start driving and turn right at the street 30 meters ahead. The vehicle needs to detect an event whether any other vehicle is coming from the left side and stop the vehicle for that. And the vehicle turns right if there is no vehicle and

TABLE III. SENSOR SERVICE INSTANCES FOR EACH SENSOR SERVICE TYPE IN THE SCENARIO

	Distance (m)	*Azimuth (°)*	*Coverage (m)*	*View Angle (°)*	*Direction (°)*	*Delay (ns)*	*Response Time (ns)*	*Min Available Time*	*Max Available Time*	*Cost ($/h)*	*Reputation*	*Accuracy (%)*
$motion_1$	48	326	8	55	225	0.2	0.5	100000	150000	0.4	4	87
$motion_2$	45	335	9	65	180	0.35	0.6	0	120000	0.3	3	85
$motion_3$	42	345	8	50	202	0.3	0.5	90000	170000	0.6	3	80
$motion_4$	20	30	5	60	337	0.2	0.6	80000	150000	0.3	3	80
$motion_5$	40	15	5	55	180	0.3	0.4	110000	180000	0.5	2	75
mic_1	19	329	5	-	-	0.2	0.6	100000	170000	0.3	4	70
mic_2	32	305	5	-	-	0.3	0.7	110000	190000	0.4	2	80
mic_3	22	320	5	-	-	0.1	0.5	90000	170000	0.5	3	75

pedestrian on the left. Here, we do not consider competitive relationships between other vehicles, that is, other vehicles which is the target of detection for this vehicle does not try to detect this vehicle in this scenario. We assume that all actions of other vehicles are given priority. In this case, this composite service can be formed by combining a motion sensor service and mic sensor service for recording and detecting events, and a Web service that classifies the kind of recorded sounds. This service gives the data to the Web service if the event is detected and from the result returned judges whether it is vehicle or not.

In this scenario, the following two complex events are targets for detection (Figure 7).

1. Pause if any other vehicle is coming from the left street.
2. Turn right after confirming that there is no vehicle or pedestrian on the right street and no vehicle is coming from the left side.

These two complex events are continuous in terms of time and space, however, the problems for one sequence with two *COP*s; COP_1 for complex event 1 and COP_2 for complex event 2 because domains and constraints of the two COPs are different. COP_1 is described by the following ECA rule.

$$CARSOUND_1 \geqslant 120\ db\ DURING\ LEFTMOVEMENT_1 == True\ THEN\ Stop(self)$$

This rule represents the complex event that only becomes true when atomic event CARSOUND1 is more than 120decibels while atomic event $LEFTMOVEMENT_1$ is true. The definitions of each atomic event involve relative time and space attributes (Table 4). The event type of atomic event $LEFTMOVEMENT_1$ is *MOVEMENT* and Motion sensor service can detect it. $CARSOUND_1$ is *SOUND* and Mic sensor service can detect it.

$CE = \{LEFTMOVEMENT_1, CARSOUND_1\}$
$X = \{Motion_1, Mic_1\}$

Domains of each sensor service type based on the spatio-temporal attributes of atomic events are as follows. In this case, each atomic event is defined in relative time and space and the domains are generated according to (Eq. 13).

$Motion_1 = \{motion_1, motion_2, motion_3\}$
$Mic_1 = \{mic_1, mic_2, mic_3\}$

Twelve attributes for $Motion_1$ and nine attributes that are same attributes without two attributes, *ViewAngle* and *Direction* for Mic_1, are set. The sensor service instances are represented in Table 3.

First, hard constraints set by the service provider and domains that satisfy them are described as follows.

C_{H1}: $Motion_1.ViewAngle > 50$
C_{H2}: $Motion_1.Delay \leq 0.6ns$
C_{H3}: $Mic_1.Delay \leq 0.6ns$
C_{H4}: $Motion_1.ResponseTime \leq 0.8ns$
C_{H5}: $Mic_1.ResponseTime \leq 0.8ns$
C_{H6}: $Motion_1.Reputation \geq 2$
C_{H7}: $Mic_1.Reputation \geq 2$

$Motion_1 = \{motion_1, motion_2\}$
$Mic_1 = \{mic_1, mic_2\}$

Constraints C_{H8}, C_{S1} and C_{S2} set by the users and hard constraint C_{H9} defined for the *DURING* relation are as follows.

C_{H8}: $Motion_1.Cost + Mic_1.Cost < \$1/h$
C_{H9}: $Motion_1.Delay \leq Mic_1.Delay$
C_{S1}: $Motion_1.Cost + Mic_1.Cost < \$0.8/h$
C_{S2}: $0.5(Motion_1.Reputation) + 0.5(Mic_1.Reputation) \geq 3$

The first constraint, C_{H8}, is hard and the combinations that satisfy it are all combinations with four instances as follows.

$$Cmb = \{\{motion_1, mic_2\}, \{motion_1, mic_2\}, \{motion_2, mic_1\}, \{motion_2, mic_2\}\}$$

Among the combinations $comb_i \in Cmb$ that satisfy C_{H9}; $comb_2$ and $comb_4$ are as follows with satisfaction status on soft constraints ρ_{1i} and ρ_{2i}.

$comb_1 = \{motion_1, mic_1\}, Delay=0.2, \rho_{11}=0, \rho_{21}=0$
$comb_2 = \{motion_1, mic_2\}, Delay=0.2, \rho_{12}=1, \rho_{22}=0$

Next, we set the objective function for *DURING* relation as the minimization of *Delay* of $Motion_1$ and penalties on soft constraints. The best combination is $comb_1 = \{motion_1, mic_1\}$.

Second, COP_2 can be represented as the following ECA rule that is a complex event with three atomic events.

$$RIGHTMOVEMENT_1 == False\ BEFORE\ LEFTMOVEMENT_2 == FALSE\ DURING\ CARSOUND_2 \leq 60\ db\ THEN\ TurnRight(self)$$

In this case, $RIGHTMOVEMENT_1$ represents the movement in the right side of the street. The other two are equivalent to $LEFTMOVEMENT_1$ and $CARSOUND_1$ in COP_1. The rule represents the complex event that only becomes true when atomic event $CARSOUND_2$ exceeds 60 decibels while atomic event $LEFTMOVEMENT_2$ is true. The definitions of atomic events involve relative time and space values (Table 4). $RIGHTMOVEMENT_1$ is *MOVEMENT* and can be detected by Motion sensor service type.

$CE = \{LEFTMOVEMENT_2, CARSOUND_2, RIGHTMOVEMENT_1\}$
$X = \{Motion_2, Mic_2, Motion_3\}$

The domains that satisfy each condition are shown below.

TABLE IV. ATOMIC EVENTS

	$LEFTMOVEMENT_1$, $CARSOUND_1$	$RIGHTMOVEMENT_1$
distance	32m	32m
azimuth	303°-349°	15°-30°
time	Within 5 seconds from the current time	Within 5 seconds from the current time
range	8m	8m

$Motion_3 = \{motion_4, motion_5\}$

Hard constraints set by the service provider are the same as those in COP_1. Some constraints C_{H12}, C_{S1} and C_{S2} set by the user are as follows. In the rule, we have two temporal relations; *BEFORE* and *DURING*. By using the rule compiling approach shown in Section 3, we can obtain the rewritten rule and we can set constraints C_{H13} for *BEFORE* relation.

C_{H12}: $Motion_2.Cost + Mic_2.Cost + Motion_3.Cost < \$1.3/h$
C_{H13}: $Motion_3.Delay < Mic_2.Delay$
C_{S1}: $0.3(Motion_2.Accuracy)+0.3(Mic_2.Accuracy) +0.3(Motion_3.Accuracy) \geq 80$
C_{S2}: $Motion_2.Reputation \geq 4$

The first constraint, C_{H12} is a hard constraint and combinations that satisfy this constraint are generated.

$Cmb = \{\{motion_1, mic_1, motion_4\}, \{motion_1, mic_1, motion_5\}, \{motion_1, mic_2, motion_4\}, \{motion_2, mic_1, motion_4\}, \{motion_2, mic_1, motion_5\}, \{motion_2, mic_2, motion_4\}, \{motion_2, mic_2, motion_5\}\}$

Among the combinations $comb_i \in Cmb$ that satisfy constraints C_{H13}, $comb_3$ and $comb_5$ are shown below.

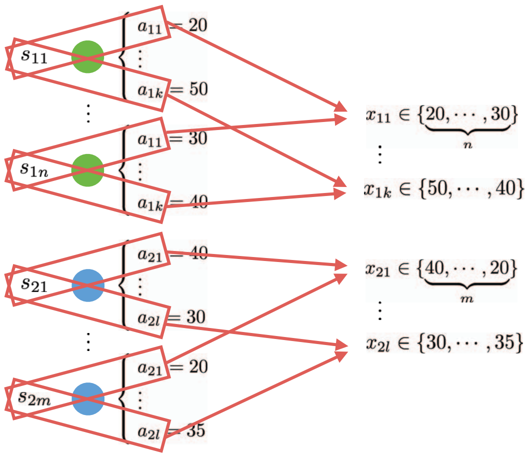
$comb_3 = \{motion_1, mic_2, motion_4\}, Delay=0.2, \rho_{13}=0, \rho_{23}=0$
$comb_5 = \{motion_2, mic_2, motion_4\}, Delay=0.2, \rho_{15}=0, \rho_{25}=1$

Finally, the objective function for *BEFORE* relation is set as minimization of *Delay* of $Motion_1$ and penalties on soft constraints. The best combination is $comb_3 = \{motion_1, motion_2, motion_4\}$.

V. IMPLEMENTATION

We implemented our proposed method by using the SAT-based constraint solver, Sugar [13]. The sensor service composition problem studied here can be described as an object-oriented expression that selects the most appropriate sensor service instances for each sensor service type as classes.

We first implemented an encoder that encodes the problem into Sugar syntax because most constraint solvers do not support object-oriented expressions. First, the encoder defines each QoS attribute of each sensor service type as a variable in Sugar syntax. Second, the combination of QoS values is fixed as instances by setting the support points (Figure 8). A support point is a combination of values which the solver (Sugar) can only take as solutions. Accordingly,



s_{11}
$a_{11} = 20$
$a_{1k} = 50$
s_{1n}
$a_{11} = 30$
$a_{1k} = 40$
s_{21}
$a_{21} = 40$
$a_{2l} = 30$
s_{2m}
$a_{21} = 20$
$a_{2l} = 35$
$x_{11} \in \underbrace{\{20, \cdots, 30\}}_{n}$
$x_{1k} \in \{50, \cdots, 40\}$
$x_{21} \in \underbrace{\{40, \cdots, 20\}}_{m}$
$x_{2l} \in \{30, \cdots, 35\}$

Figure 8. Definition of variables and Instances as support points

we can solve the problem, originally described using object-oriented expressions, on Sugar. For example, green sensor service type has three attributes; *Cost*, *Reputation* and *Accuracy* in Figure 9. Each attribute of green sensor service type is defined as a variable with domain values (x_{11} with 3 and 4, x_{12} with 2 and 3, and x_{13} with 50 and 80). In order to fix each instance, support points are defined. For example, values 3, 2 and 50 of variables x_{11}, x_{12} and x_{13} represent the instance s_{11}. Therefore, we set 3, 2 and 50 as the support point (specifically, set as *(relation r1 3 (supports (3 2 50)))* and *(r1 x11 x12 x13)* in Sugar syntax). Consequently, combinations which are not defined as support points cannot be selected as solutions. Setting a hard constraint for each

QoS is achieved by setting it as a constraint for each variable. In Figure 9, for example, if we set the hard constraint of *Cost*

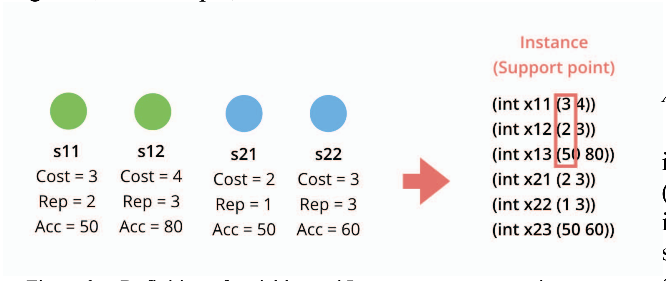


Figure 9. Definition of variables and Instances as support points

of green sensor service type is more than 3, we need to describe it as *(> x11 3)* in Sugar. Here each instance is fixed as support points and the only instance whose cost is more than 3 is s_{12} (QoS value; *4*, *3*, *80*). As a result, s_{12} is selected as the instance satisfying the hard constraint. On the other hand, a soft constraint can set as a domain of penalty values *{0,1}*, a constraint, and a clause that connects the constraint to the domain of penalty values. In Figure 9, for example, if we set a soft constraint of *Reputation* of blue sensor service type must be more than or equal to 3, we need to describe it as *(int p1 0 1)* and *(imp (= p1 0) (>= x22 3))* in Sugar. The satisfaction status of each instance is that s_{21} *does not* satisfy and s_{22} satisfies. Soft constraints are considered in the optimization process shown in Section 4. As a first step, we can describe hard and soft constraints as lambda equations by using sensor service types, indexes, the name of attributes in Python syntax, such as *lambda i: motion1[i].delay <= 0.6*. The encoder then encodes the lambda equations into Sugar Syntax with variable names.

By solving the problem, the appropriate combination is selected and we confirm sensor service composition has been realized by the proposed method. It takes .251 seconds to encode the problem and .600 seconds to solve the problem (.900 seconds in total). The execution time increases with the scale of problem. The complex event we show in Section 4 is composed with three atomic events at most, however, complex events will be bigger with more atomic events in general such as more detailed events. As the number of atomic events in a complex event increases, the number of sensor service types for detecting them will be needed and it leads to the exponential growth in the number of sensor service instances and the number of QoS attributes. Therefore, the scale of problem can be regarded as determined by the number of sensor service instances and the number of QoS attributes. The former can be reduced by our definition of domains proposed in this paper, however, improvement of efficiency in recalculation is a critical issue because attributes differ with the sensor service and its increase is inevitable.

So far in COP, dynamic COP which improves efficiency when recalculation is required by reusing the solution of previous problems [14] or generating robust solutions toward the future problem based on probabilities [15] and we consider they are applicable to sensor service composition problem. However, it has not become clear how big problem needs dynamic COP and detailed simulation about changes in the scale of problem is required for future work.

VI. Related Works

A. Event-Driven SOA

As part of SOA research, event-driven SOA which introduces mechanisms into event-driven architecture (EDA) has been investigated. Spiess et al. proposed the integrated architecture, which abstracts interfaces with Web service standards and focuses on event-based messaging in order to enable access to devices in real world by using the service-oriented approach in enterprise services [5].

For realizing event-driven SOA, services needs to handle event. Juric proposed the extended model for managing EDA in SOA to yield event-driven service orchestration and services as event producers with event sources producing events or event consumers with event sinks receiving events [6]. Therefore, such service has event sources and/or sinks for responding to events in addition to regular service interface. Potocnik et al. proposed a model based on Complex Event Aware (CEA) services which correspond to complex events for realizing Complex Event Processing (CEP) which is not supported in existing SOA platforms [7]. Sensor service, the target of this research, is also an event service because it responds to events.

B. Sensor Service

In SOA, sensor service was proposed to realize the decoupling of sensors by wrapping sensor functionalities as services such as Sensor Data as a Service or Sensing as a Service (SenaaS). Perera et al proposed a model for using sensing as a service to address the issues of technology, economy and social aspects in the domain of Smart Cities [8]. Also, Kyusakov et al. proposed improvements in the efficiency of deploying sensor nodes as Web services accessed by users via SOAP without any middleware [9].

Sensor service is often mentioned in the context of accessing sensors from the cloud or data accumulation. We, on the other hand, focus on sensor service as an input interface of Event-based System (EBS) and achieving real time utilization. Alam et al. proposed a sensor virtualization framework for realizing event-driven SOA in IoT domain by using the concept of SenaaS [10]. Therefore, a key research issue is how to select and compose sensors virtualized as services for real time utilization.

C. Sensor Selection and Composition

The combinational problem of sensors has been investigated for sensor selection and sensor composition. In horizontal service composition for Web services, selection and composition are almost synonymous because both presuppose input/output inclusion relation of services and several service instances should follow the workflow. However, in the case of sensor services, the workflow is not always given and therefore sensor service selection and composition do not have the same meaning. That is, sensor selection subsumes sensor composition.

Sensor selection is the problem of deciding the set of sensors that satisfy functional requirements. Perera et al. took sensors to be services and proposed a model and algorithms of context-aware sensor search that could realize similar functionalities from a huge numbers of sensor service candidates by selection and ranking [11]. Beyond location information they employ comprehensive information including QoS of sensor services as context information.

On the other hand, sensor composition is, unlike sensor selection, the problem of deciding the set of sensors that have the desired relationships. Gao et al. considered sensor services as an event service in event-driven SOA and proposed a QoS-aware composition method of event services according to the rules of complex events for detecting complex events, not single event [12]. They consider rules of complex events in the same way as workflows in Web services and realize event service composition in accordance to rules by defining QoS aggregation schemata and utility functions for each logical operator in complex event rules with optimization by a genetic algorithm (GA).

VII. Conclusion

In this research, we introduced horizontal service composition by using COP which has been used for Web service composition and proposed a position-sensitive sensor service composition method. First we defined the domain with spatio-temporal requirements of sensor services toward atomic events and formalized the problem as COP by setting different aggregation functions for each logical operator in complex events. Moreover, we extended sensor service composition to include the detection of complex events with temporal relations by introducing temporal constraints. Also we realized our proposed method by implementation on Sugar and clarified the necessity of dynamic COP when the scale of problem will be bigger. Future work includes identifying how big a problem needs to be to trigger the use of dynamic COP by conducting detailed simulations of problems with varying scales.

Acknowledgment

This research is supported by the Leading Graduates Schools Program, "Collaborative Graduate Program in Design" by the Ministry of Education, Culture, Sports, Science and Technology, Japan. Also, this research was partially supported by a Grant-in-Aid for Young Scientists (A) (17H04706, 2017-2020) and a Grant-in-Aid for Scientific Research (A) (17H00759, 2017-2020) from Japan Society for the Promotion of Science (JSPS).

References

[1] D. Miorandi, S. Sicari, F. D. Pellegrini, I. Chlamtac, Internet of things: Vision, applications and research challenges., Ad Hoc Network, Vol. 10, No. 7, pp. 1497-1516, 2012.

[2] K. M. Lhaksmana, Y. Murakami, T. Ishida, Analysis of Large-Scale Service Network Tolerance to Cascading Failure., IEEE IoT Journal, Vol. 3, No. 6, pp.1159-1170, 2016.

[3] L. Zeng, B. Benatallah, A. H. H. Ngu, M. Dumas, J. Kalagnanam, H. Chang, QoS-Aware Middleware for Web Services Composition., IEEE Transactions on Software Engineering, Vol. 30, No. 5, pp. 311–327, 2004.

[4] M. Otani, T. Ishida, Y. Murakami, T. Nakaguchi, Event Management for Simultaneous Actions in the Internet of Things., IEEE 3rd World Forum on Internet of Things (WF-IoT), pp.64-69, 2016.

[5] P. Spiess, S. Karnouskos, D. Guinard, D. Savio, O. Baecker, L. M. S. Souza, V. Trifa, SOA-Based Integration of the Internet of Things in Enterprise Services., IEEE International Conference on Web Services (ICWS), 2009.

[6] M. B. Juric, WSDL and BPEL extensions for Event Driven Architecture., Information and Software Technology, Vol. 52, No. 10, pp. 1023–1043, 2010.

[7] M. Potocnik, M. B. Juric, Towards Complex Event Aware Services as Part of SOA., IEEE Transactions on Services Computing, Vol. 7, No. 3, pp. 486–500, 2014.

[8] C. Perera, A. Zaslavsky, P. Christen, D. Georgakopoulos, Sensing as a service model for smart cities supported by Internet of Things., Wiley Transactions on Emergin Telecommunications Technologies., Vol. 25, Issue. 1, pp. 81-93, 2014.

[9] R. Kyusakov, J. Eliasson, J. Delsing, J. Deventer, J. Gustafsson, Integration of Wireless Sensor and Actuator Nodes with IT Infrastructure Using Service-Oriented Architecture., IEEE Transactions on Industrial Informationcs., Vol. 9, Issue. 1, pp. 43-51, 2013.

[10] S. Alam, M. M. R. Chowdhury, J. Noll, SenaaS: An Event-driven Sensor Virtualization Approach for Internet of Things Cloud., IEEE International Conference on Networked Embedded Systems for Enterprise Applications (NESEA), 2010.

[11] C. Perera, A. Zaslavsky, C. H. Liu, M. Compton, P. Christen, D. Georgakopoulos, Sensor Search Techniques for Sensing as a Service Architecture for the Internet of Things., IEEE Sensors Journal., Vol. 14, Issue. 2, pp. 406-420, 2014.

[12] F. Gao, E. Curry, M. I. Ali, S. Bhiri, A. Mileo, QoS-Aware Complex Event Service Composition and Optimization Using Genetic Algorithms., Service-Oriented Computing: 12th International Conference, ICSOC 2014, Paris, France, November 3-6, 2014. Proceedings, Springer, pp. 386–393, 2014.

[13] A. B. Hassine, S. Matsubara, T. Ishida, A Constraint-Based Approach to Horizontal Web Service Composition., Proceedings of the 5th International Conference on The Semantic Web, pp. 130–143, 2006.

[14] J. F. Allen, Maintaining knowledge about temporal intervals., Communications of the ACM, Vol. 26, No. 11, pp. 832–843, 1983.

[15] N. Tamura, N, A. Taga, S. Kitagawa, M. Banbara, Compiling Finite Linear CSP into SAT., Constraints, Vol. 14, No. 2, pp. 254–272, 2009.

[16] S. Minton, M. D. Johnston, A. B. Philips, P. Laird, Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems., Artificial Intelligence, Vol. 58, No. 1, pp. 161–205, 1992.

[17] H. Fargier, J. Lang, T. Schiex, Mixed constraint satisfaction: A framework for decision problems under incomplete knowledge., Proceedings of the Thirteenth National Conference on Artificial Intelligence, pp. 175–180, 1996.