A Hybrid Complex Event Service Based on IoT Resource Models

Yang Zhang
State Key laboratory of Networking and Switching Technology
Beijing University of Posts & Telecommunications, Beijing 100876, China
YangZhang@bupt.edu.cn

Jun-liang Chen
State Key laboratory of Networking and Switching Technology
Beijing University of Posts & Telecommunications, Beijing 100876, China
chjl@bupt.edu.cn

Abstract—Traditional Complex Event systems (CEP) did not consider the computation requirements of continuous dynamic behavior such as differential equations. In addition, the event composition rules were predefined before the CEP engine began working. The rule defining task is error-prone and cumbersome. In this paper, therefore, a hybrid complex event service is proposed, which deals with not only discrete events but also continuous behavior computation based on IoT (Internet of Things) resource models. In order to satisfy the real-time constraints of processing IoT events, a divide-and-conquer principle is adopted, where we give a combination theorem such that different events can be processed on different IoT resources and then these processed results can be combined to derive complex events. Based on the formal IoT resources and event knowledge, we define interest goals to direct event composition without enumerating event relation to define event composition rules. We finally present event composition algorithms and evaluation to show our idea.

Keywords—*CEP; Satisfiability Modulo Theories; IoT Resource; Publish/Subscribe*

I. INTRODUCTION

Existing Complex Event Processing (CEP) systems often aim at processing many flows of events to discover situations of interest. In these CEPs, the processing takes place according to user-defined rules which specify the relations between the observed events and the phenomena to be detected. We claim that the complexity of writing such rules is a limiting factor for the diffusion of CEP. In addition, describing all rules for many kinds of events is also error-prone and cumbersome. For an IoT application, it is often required to automatically discover such rules to detect critical events.

To discover event composition rules is not a trival task. In an IoT application, we can define event schema, type and semantic for atomic events; and define resource models for monitored physical entities. But we can not enumerate all relations among events and resources. There exist two cases for discovering the rules to compose events, which are as follows:

(1) There are formal knowledge descriptions about IoT resources and events, and some event composition rules can be extracted from the knowledge when the composite events represent the relation change among IoT resources, or are logical consequence. The only thing to do is to extract the rule knowledge and to derive some composite events with minimizing computation cost.

(2) The composite events are not the logical consequence of knowledge. Sometimes, the monitored events from physical environments imply the appearance of some new knowledge, and we should observe the event occurrence to discover it to compute composite events from event flows, *i.e., creating new rule knowledge by learning*.

Given the IoT resource models and event knowledge, how to discover event composition rules for computing complex events is not obvious. When a great deal of events appear, the change of event relation may be a clue. But distinguishing normal relation change from event composition rule is a challenging task. It is impossible to list all event relation to identify abnormal ones. The space and time relation change among different events are a good starting point, while the quantity change of one attribute in an event can be directly defined and is a certain and simple indicator without additional computation.

(1)*The change of event space relation.* The monitoring system often collects all events about concerned entities in one local area and publishes these events in the publish/subscribe middleware. We can subscribe all these events. For each event from one location, we check whether its space neighbors have changed. Using this way, the space relation change can be discovered by minimal costs. The next step is to evaluate the significance of such discovery, and to check whether it represents the actual relation change between IoT resources.

(2)*The change of event time relation.* We can check its before-happened events to see whether its ancestors are different or their time distance has changed when we get an event from the publish/subscribe middleware. In order reduce the storage cost, only direct ancestors are recorded. The next step is also to evaluate the significance of this time relation change.

(3)*Actively learning knowledge.* Only listening to the events in the publish/subscribe middleware is a passive way to get rule knowledge. In some cases, we should actively publish some allowable composite events to see the response of IoT applications, where we can rapidly get some new knowledge and rules. Certainly, when the composite events are published, we can use the ways in (1) and (2) to analyze the response of IoT applications.

When we discover the space or time relation change among different events, it does not imply the actual knowledge to produce a composite event. It may be a random fluctuation, or does not reflect a new trend, or has not useful meanings. We should recognize that it is interesting. So the next step is to evaluate whether it is important and interesting, which is carried out as follows:

(1) Because one stable trend often lasts a period and sensors can continuously monitor it, we observe whether the trend is strengthened or stable with time elapse.

(2) We define our interest goals rather than event composition rules such as safety goal and security goal. If one goal event is a logical consequence of occurring events, the composite events are gotten.

Given the interest goals, we deduce whether the occurring events are related to them, which is based on the IoT resource models. There are two issues to be addressed before the deduction method is used:

(1) The IoT resource model involves not only physical entities' attributes and relations, but also the physical entities' working laws which are often described by high

978-1-4673-7272-5/15 $31.00 © 2015 IEEE
DOI 10.1109/ICWS.2015.88

differential equations, *i.e., continuous dynamic behavior*. We should adopt some hybrid methods to deal with discrete events and continuous computation as the basis of processing complex events.
(2) There are lots of resources and events in the IoT applications such that the real-time requirement of processing complex events is hard to be satisfied. We should adopt a divide-and-conquer principle to carry out on-line composite event deduction.

The contributions of this paper are as follows:
(1) A complex event (or complex event processing) service, *called CEP service*, is proposed, where the procedure of describing event composition rules is avoided, and interest goals are defined based on formal IoT resource models and event knowledge such that the new event composition rules are discovered from logical deduction to produce complex events.
(2) A hybrid approach, *called satisfiability modulo theories* (SMT), is used to cope with the continuous computation requirement of IoT applications, where we propose a combination theorem to make it possible for the CEP service to decompose deduction task on different IoT resources with distributing complex event processing on a large scale.

II. PRELIMINARIES

A. Satisfiability Modulo Theories

A signature Σ is a set of functions, and constants are 0-arity functions. Term is made on Σ and a set V of variables, and the term set is denoted by $T(\Sigma,V)$. The first order logic is built on $T(\Sigma,V)$, the predicate set P, and logical symbols such as $\neg,\wedge,\vee,\rightarrow,\exists,\forall$. A predicate element in P and an equation between terms are called atom as well as *false* and *true*, and one atom or its negation is called literal. A formula φ consists of a set of literals and logical connections connecting the literals. Then, a sentence is a formula with no free variables. A theory $\Sigma-TH$ is a set of sentences. A formula φ is satisfiable if it is true under the interpretation over the theory $\Sigma-TH$; otherwise it is unsatisfiable.

Satisfiability modulo theories (SMT) is a problem to decide if a formula is satisfiable over some theories [13], [14]. There are lots of significant developments of SMT in the past decade, which include fast Boolean satisfiability solvers (SAT), some efficient decision procedures for many expressive theories, and modular combination theorems [15], [16], [17]. SMT has become a standard tool in industrial applications to complete automated reasoning and verification. In this paper, we only assume the resolution procedure combined by background theory testing is sound and complete without further discussion.

B. IoT Resources and Events

The digital entities representing the physical entities are called IoT *resources* [18]. An IoT resource model consists of an object model and a lifecycle model. An object model expresses sensors, physical objects, and their relationship. A lifecycle model describes the working laws of resources, as well as the transition between two lifetime states.

Definition 1. IoT Resource Lifecycle Model [18]. *A resource lifecycle model is a 5-tuple*

$$Life::=(State_0,State,PR,Label,Map_{S\_L\_S})$$

where $State_0$ *is an initial state set of an entity,* $State$ *is a state set of the resource,* P_R *is a set of propositions which are used to define resource states,* $Label$ *is a name set to label a state transition,* $Map_{S\_L\_S}$ *is a map over the sets* $State$ *and* $Label$ *to define state transitions:* $State\times Label\rightarrow State$.

To share the events among different IoT services, the events are named, *i.e., assigning a topic name to each kind of event.* The relationship among multiple event names is often represented by a name tree. According to the topic names and consumers' subscriptions, the events can be delivered to the consumers. The definition of event is given in *Definition 2*.

Definition 2. Event [18]. *An event is defined as*

$$event::=topic\ \ subject(topic\ \ verbing,content)$$

where $topic\ \ subject(topic\ \ verbing)$ *forms a topic name to represent a kind of events; and* $content$ *represents an event body which is often defined by a XML (Extensible Markup Language) Schema.*

In the above definition, *verbing* may be empty.

III. FRAMEWORK

A. Basic Concepts and Ideas

We use the first-order predicate logic to define events and IoT resources, and to compute composite events. In this section, we only sketchily describe the basic concepts and ideas.

The event names and its content schema form events' sort set. The schema consists of multiple name/value pairs, and only a sub-set of name/value pairs forms a session which is also used to act as an event sort. The timestamp and resource identifiers corresponding to resource models are often necessary in the event content. Although events are generated by resources in IoT applications, and include IoT resources' property update events, state transition events, command events, and so on, there are some events which have some additional computation features such as encryption and decryption. So these functions are introduced to describe events. The event representation with knowledge form in our CEP service is as follows:

-Event sorts: an event sort $s::=(topic\_name,XML\ Schema)$, all sorts form a sort set S.

-Individual events: $e_1:s_1,e_2:s_2,\cdots$; $s_1,s_2,\cdots$ are event sorts.

-Variables: $x_1:s_1,x_2:s_2,\cdots$ form a variable set X.

-Functions and Equations: $f(x_1:s_1,\cdots,x_n:s_n):s$ is a function with n-arities events as inputs and one event as output with sort s; two functions f_1 and f_2 may form an equation $f_1=f_2$; functions and individual events form a term signature set $\sum$, and equations form a set E.

-Event Predicate: $K(e)$ means the system perceives the event e, where different predicates are allowed to form a predicate set P.

-Knowledge: a set of known clauses such as

$K(e_1) \wedge \cdots \wedge K(e_n) \rightarrow K(e)$.

Given an event representation, we can define some knowledge. For example, $e_{s_{afety}}$ represents the safety events in a coal mining system, e_{high-c} represents the events of methane gas being high, and $K(e_{high-c}) \rightarrow K(e_{s_{afety}})$ is an event knowledge meaning that e_{high-c} is a kind of safety event $e_{s_{afety}}$. Given an event knowledge as well as its underlying resource model, users can express their interests such as $\rightarrow K(e_{s_{afety}})$. Compared with traditional CEP systems, it is only required in our CEP service that event knowledge and resource models are defined without repeatedly defining event composition patterns for different goals and different event sources. For a user, it is error prone and cumbersome to define event composition patterns during using the CEP systems.

For example, the user only expresses his/her interest $\rightarrow K(e_{s_{afety}})$ for a coal mine system. When the system monitors that the wind speed is dropped, our CEP service may compute how much the methane gas will increase in the mine with the decrease of air quantity, which is based on the methane gas resource model. If the increasing of methane gas induces a state transition, then $K(e_{high-c})$ holds. With $K(e_{high-c})$ and $K(e_{high-c}) \rightarrow K(e_{s_{afety}})$, our CEP service will output $\rightarrow K(e_{s_{afety}})$ even though the user has not defined patterns and concerns about wind events. It is computed based on knowledge. From the example, we know the resource models are very important for our inference.

For the relationship defined in a resource model, we can use ontology to represent it which is not our focus and we only assume there are ontologies to represent these relations in the resource models. We care for resource lifecycles including state, transition, running constraints, and so on. The resource model is represented in the first-order logic as follows:

- a set of properties: VP .
- a set of states: $States$ are defined by properties and arithmetical operators on them such as $+, -, \times, \div, <, >$, which form formula φ .
- a set of initial states: $States_0$.
- a set of state transitions: $s \in States \wedge \varphi \rightarrow s' \in States$.
-running constraint in each state: in each state, its running is represented by the differential functions $\dot{f}$, then the formula φ including differential functions describe the relation among resource properties in a state. When some attribute exceeds the limits represented by the state formula, the state transition occurs and new running feature formula including new $\dot{f}$ is used for new running state.

For the coal mine example, there are two resources: the methane gas model and the mine ventilation model. In the methane gas model illustrated in Fig. 1a, there are two states: a normal state identified by a proposition $gas < Up$ and an unsafe state identified by a proposition $gas >= Up$, where, in the normal state, the emission speed of methane gas from coal bed is described by a function $eGas = emit(loc, time)$ with location loc and time $time$ as parameters, the diffusion speed of methane gas is described by the function $dGas = diff1(win, time)$ with wind speed win and time $time$ as parameters, the functions may not have explicit expressions and only function relations are considered in this paper; in the unsafe state, only the diffusion function is different. In mine ventilation model illustrated in Fig. 1b, there are two states: an initial state identified by a proposition $location = 0$ meaning fans' position and a mine state identified by a proposition $location > 0$ meaning a coal mine position receiving wind from $location = 0$, where, in the initial state, the fan's wind pressure is described by a function $p_0 = \sum_{i=0}^{5} a_i q_f^i$, the wind quantity is described by $q_f = q_N + q_l$ with q_N being the effective wind quantity and q_l being the leaked wind quantity; in the mine state, the wind speed v and pressure can be computed according to the differential functions and mine tunnel topology $G(E,V)$; the state transition labeled by $G(E,V)$ means that, given an initial state, the mine tunnel topology may decide the ventilation attributes in one location of the mine. two resource models have a relation, *i.e., the diffusion speed of methane being decided by the mine ventilation model.* When the wind speed and quantity become larger in one location, the diffusion speed of methane becomes rapid in the location. The parameter win of $diff1(win, time)$ is decided by the mine ventilation model.

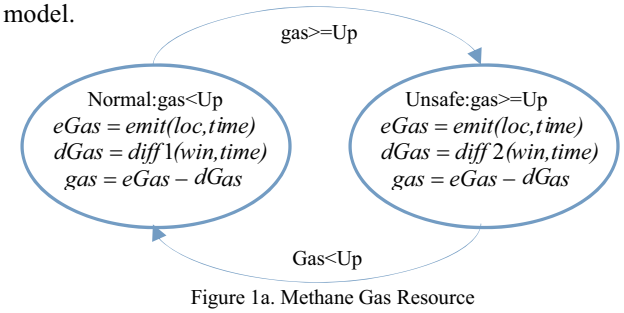

Figure 1a. Methane Gas Resource

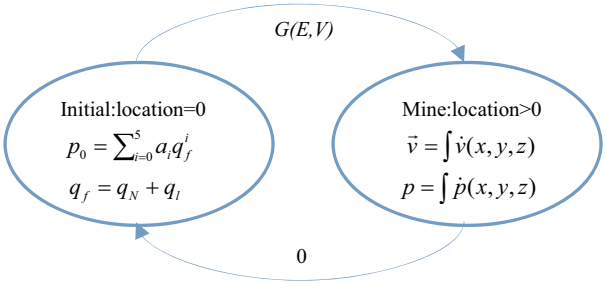

Figure 1b. Mine Ventilation Resource

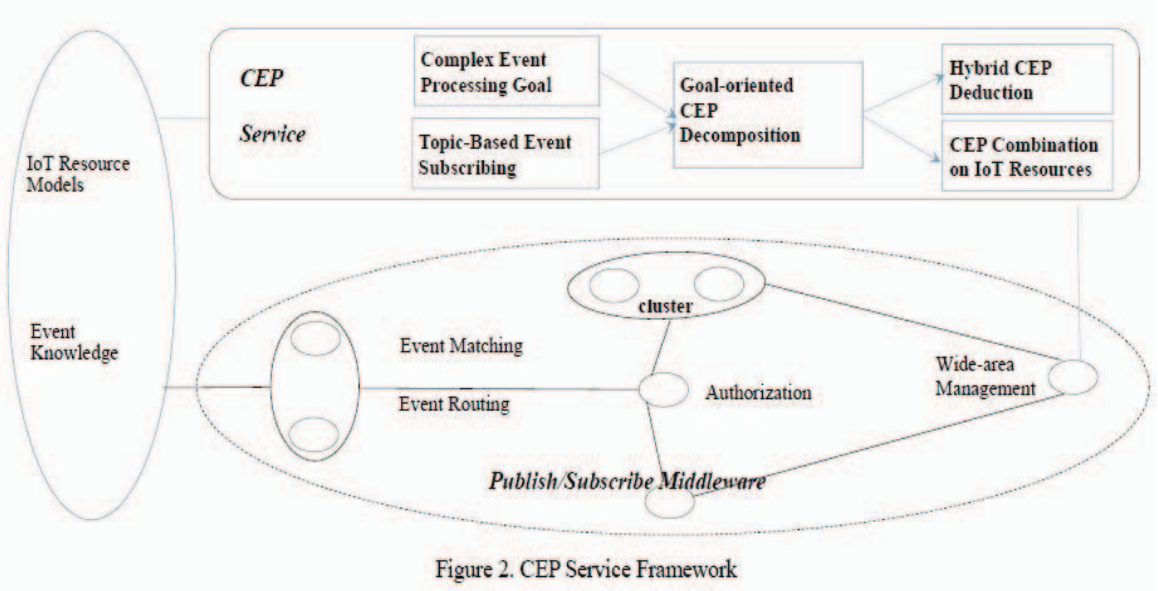

Figure 2. CEP Service Framework

The two resource models can be used by SMT solvers. The state and state transition can be represented by formula as the SMT's Boolean constraints, and the SMT's background theories use these resources' functions which represent the working laws of physical entities.

B. Basic Framework for CEP Service

The complex event service subscribes concerned events and receives event flows from our publish/subscribe middleware. The service, then, uses its knowledge based on predicates, to infer composite events without depending on specific event composition patterns.

For example, in a coal mine, when an emergent event occurs, dispatchers will do some actions to restore the system into a safety state. Because the system is very complex, there is not a completely automatic scheme to deal with all things and people participate in the emergency handling. Not all dispatchers have the same high level of skills and experiences such that their handling results are very different. Even if the same dispatcher copes with the same event at different time, the handling results are also different because the current real environment is too complex to fully match the last scenes. We should use the CEP service to discover some rule knowledge for helping the dispatchers deal with the emergent event.

The CEP service includes three functionality parts: preprocessing event goals to subscribe concerned events, carrying out goal-oriented event deduction based on divide-and-conquer methods, and combining continuous computation results with distributed event deduction results to derive complex events, which is illustrated in Fig. 2. It accesses to the resource models and the event knowledge, and then computes the processing scope of goal events including resource scopes and event knowledge scopes. After that, the CEP service subscribes the concerned events by topic names to the publish/subscribe middleware. When a scoped event arrives at the CEP service, it decomposes the deduction task according to the resource scopes. Each deduction fragment is carried out over one resource model with continuous function computation in background theories. The deduction procedures over different resource models interact each other, and then the goal event as well as proof chain are output as a combined result.

For example, when the initial wind speed is suddenly dropped, the CEP service receives the event from the publish/subscribe system. The pre-computed scopes say that the two resource models of methane gas and mine ventilation are scoped according to the goal. The methane gas model is firstly used, and a resolution procedure is carried out. The resolution procedure with testing in background theories says if the methane gas diffusion speed becomes low, the goal will be inferred. It requires that the wind speed in one location satisfies a proposition to mean the wind being low, which is shared by the two resource models. Then, we carry out the inference on the ventilation model, which finally requires that the initial wind speed and quantity are low enough. If it is true, the CEP service publishes the goal event and the proof chain, *i.e., our CEP acting as a service to subscribe events and publish its processed results as composite events*.

IV. COMBINATION THEOREM

It is desirable to make concurrent event deduction on different resource models. There are many approaches to address the combination issue in SMT field [15], [16]. But they often require that the Boolean constraints are explicit and are decomposed before actual deduction. Such assumptions are often not true in an IoT application because the resource models may be very complex with implicit

expression such as high-order differential equations and some Boolean constraints are computed during system running. In addition, different resource model may not represent a different theory over different term symbols. In our paper, the events are processed on the same term symbols, and the deduction decomposition is carried out over natural resource autonomy. Therefore, we introduce a combination theorem based on the resolution method to act as a basis.

We introduce the modular concept for a knowledge set, where we extend the partition-based proof method in [19] to solve the SMT combination problem. $\{\Lambda_i\}_{i\le n}$ is a partitioning of a knowledge set Λ, *i.e.,* $\Lambda=\bigcup_i\Lambda_i$, where Λ_i is a subset. Each Λ_i is called a partition, $L(\Lambda_i)$ is its signature. $\Gamma(\Lambda_i)$ is a language over $L(\Lambda_i)$, the set of formulae built with $L(\Lambda_i)$. Each such partitioning defines a labeled graph $G=(V,\varepsilon,l)$, called *intersection graph*. In the intersection graph, each node i represents an individual partition Λ_i ($V=\{1,\cdots,n\}$); and two nodes i,j are linked by an edge if $L(\Lambda_i)$ and $L(\Lambda_j)$ have a symbol in common ($\varepsilon=\{(i,j)\mid L(\Lambda_i)\cap L(\Lambda_j)\neq\Phi\}$), Φ being empty. We refer to $l(i,j)$ as the labeled symbols between partitions Λ_i and Λ_j.

We assume that partitions share term symbols including constants, functions, and variables. We partition the formula into different parts overt the same theory. One formula part is modularly proved on one resource model, where the resource model is also represented by a set of formulas, and different proof units on different resource models interact with each other.

Definition 3 (Modular Proof) [19]. *For a partitioning* $\Lambda=\Lambda_i\cup\Lambda_j$, *a graph* $G=(V,\varepsilon,l)$, *and a query formula* q, *the modular proof decomposes* q *into* q_1 *and* q_2 *with* $q=q_1\wedge q_2$, *and works over the theory* T *as follows:*

(1) For the edge $(i,j)\in\varepsilon$, *do:*

- *(a) Perform resolution for query* q_1 *in* Λ_j *over the theory T. If* Λ_j *includes the clause* q' *(as input or resolvent for resolution) which is in* $\Gamma(l(i,j))$, *then add* q' *to* Λ_i.
- *(b) Perform resolution for query* q_2 *in* Λ_i *over the theory T. If* Λ_j *includes the clause* q'' *(as input or resolvent for resolution) which is in* $\Gamma(l(i,j))$, *then add* q'' *to* Λ_j.
- *(c) Go to (a) until some fixed points are achieved or empty clause is achieved.*

(2) If q_1 *and* q_2 *is proven , return* ***TRUE***.

In order to prove the **Modular Proof** is sound and complete, a below assumption is introduced, saying that the predicates being satisfied on one resource is also satisfied on the combination of the two resource models. In general, the assumption does not hold because they share the same underlying theory, and G. Nelson and D. C. Oppen [15] discussed such combination problem on disjoint or non-disjoint signatures. However, our assumption is still valuable because the IoT resources can be physically partitioned to support independent proof. In addition, the combination method of [15], [16] used the signature difference as their basis, which is not suitable in IoT applications because the symbols of IoT resource models are overlapped and complex.

Independence Assumption 1. *Given a formula* $A_1\wedge\cdots\wedge A_l\wedge C_1\wedge\cdots\wedge C_m\wedge B_1\wedge\cdots\wedge B_n$ *being the conjunction of clauses, which is partitioned into* $A_1\wedge\cdots\wedge A_l\wedge C_1\wedge\cdots\wedge C_m$ *and* $B_1\wedge\cdots\wedge B_n\wedge C_1\wedge\cdots\wedge C_m$, *and their background theory* T, *we assume that* $A_1\wedge\cdots\wedge A_l\wedge C_1\wedge\cdots\wedge C_m$ *and* $B_1\wedge\cdots\wedge B_n\wedge C_1\wedge\cdots\wedge C_m$ *being true in* T *implies* $A_1\wedge\cdots\wedge A_l\wedge C_1\wedge\cdots\wedge C_m\wedge B_1\wedge\cdots\wedge B_n$ *being true in* T, *called Independence Assumption.*

Theorem 1. *Given a partitioning* $\Lambda=\Lambda_i\cup\Lambda_j$, *a graph* $G=(V,\varepsilon,l)$, *and a query formula* $q=q_1\wedge q_2$ *without requiring* $q_1\cap q_2$ *being empty, if Independent Assumption holds, the modular proof is complete and sound, i.e.,* $\Lambda\models q$ *iff (if and only if) the* **Modular Proof** *outputs* **TRUE**.

Proof.

Soundness.

Initially, if $\Lambda_i\models q_1$ *and* $\Lambda_j\models q_2$ *has been correct without formulae exchange, then* $\Lambda_i,\Lambda_j\models q$ *according to the independence assumption, i.e.,* $\Lambda\models q$ *with* $\Lambda=\Lambda_i\cup\Lambda_j$.

During the modular proof with formulae exchange, Λ_i *and* Λ_j *are updated, and some formulae from* Λ_j *are added into the* Λ_i *as well as the others from* Λ_i *being added into* Λ_j. *Thus, we get* Λ_i' *and* Λ_j'.

If $\Lambda_i'\models q_1$ *and* $\Lambda_j'\models q_2$, *then* $\Lambda_i',\Lambda_j'\models q$ *according to the independence assumption, i.e.,* $\Lambda'\models q$ *with* $\Lambda'=\Lambda_i'\cup\Lambda_j'$.

Assume $\Lambda_i'=\Lambda_i\cup\Delta_j$, $\Lambda_j'=\Lambda_j\cup\Delta_i$ *where* $\Lambda_j\models\Delta_j$ *and* $\Lambda_i\models\Delta_i$, *then* $\Lambda'=\Lambda_i\cup\Delta_j\cup\Lambda_j\cup\Delta_i$.

$$\Lambda'\models q\Rightarrow\Lambda_i\cup\Delta_j\cup\Lambda_j\cup\Delta_i\models q$$

and $\Lambda_i, \Lambda_j \models \Delta_j, \Delta_i$

Then $\Lambda \models q$ *with* $\Lambda = \Lambda_i \cup \Lambda_j$ *because the redundant premises can be deleted.*

Completeness.

If $\Lambda \models q$, *then* $\Lambda \models q_1$ *and* $\Lambda \models q_2$.

According to Craig's Interpolation Theorem [20, 21], there are $\Lambda_j \models q'$, $\Lambda_i, q' \models q_1$ *or* $\Lambda_i \models q''$, $\Lambda_j, q'' \models q_1$ *for* q_1. *Without loss of generality, we assume* $\Lambda_j \models q'$, $\Lambda_i, q' \models q_1$ *for* q_1, *and* $\Lambda_i \models q''$, $\Lambda_j, q'' \models q_2$ *for* q_2.

Then we assume $\Delta_j = q'$ *and* $\Delta_i = q''$, *and we get* $\Lambda_i' = \Lambda_i \cup \Delta_j$, $\Lambda_j' = \Lambda_j \cup \Delta_i$, *and* $\Lambda_i' \models q_1$ *and* $\Lambda_j' \models q_2$.

Craig's interpolation theorem is true for resolution-based theorem proving and the resolution procedure is refutation complete [19, 22] such that the resolution can generate possible clauses to show refutation. That is to say, Δ_i *and* Δ_j *can be generated by the resolution procedure. So, the* ***Modular Proof*** *is also complete.*

V. COMPLEX EVENT SERVICE

Because there are lots of IoT resource models and events, we need a way to avoid unnecessary processing for satisfying the real-time requirements, where three problems should be solved. The first is that, under what conditions, the CEP service starts working. Given appropriate starting conditions, the number of CEP service instances can be substantially reduced. The second is that, given all resource models and goals, the CEP service works on what goals and resource models. Given the working scopes, the CEP service need not visit all IoT resources and events. The third is how to efficiently carry out the deduction procedure to get the event composition result.

The CEP service consists of three algorithms to solve the three problems, which is goal-oriented without event composition patterns, and is based on IoT resource models to decompose deduction tasks and deal with continuous computation. The first is *CEP Starting Algorithm* which decides whether a CEP service instance is created.

Algorithm 1. **CEP Starting Algorithm.**

Input: An event.

Output: Empty, or a CEP service instance.

1. For the event, if its included resource attribute exceeds some threshold or it is a state transition event, then the algorithm continues; otherwise the algorithm jumps to step 3.

2. The algorithm checks whether the event is related to the goal event according to the event knowledge. If the checked result is false, the algorithm returns empty, otherwise jumps to step 4.

3. The algorithm checks the space and time relation between the received event and the stored recent events. If the relation does not change, the algorithm returns empty.

4. The algorithm creates a service instance to carry out logical deduction, and returns the instance.

For example, when an event $e_{wind-low}$ appears to mean the fan's wind pressure and wind quantity drop rapidly, the algorithm 1 creates a CEP service instance to compute the composite event because the attributes are below the bottom limits. The algorithm 2 computes the deduction scope for a given event, i.e., what resource models being used.

Algorithm 2. CEP Working Scope Algorithm.

Input: An event.

Output: Related resource models and events.

1. For the event, the algorithm extracts the resource identifier from the event content.

2. The algorithm uses the resource identifier to get the resource model, and then enumerates all related resource models by recursively finding the connection between one resource model and others.

3. For each related resource model, the algorithm checks whether its events are in the event knowledge related to the interest goal, and gets the related events.

4. The algorithm returns the related resource models and events as well as goal events.

For example, the goal e_{safety} represents the safety events in a coal mining system, e_{high-c} represents the events of methane gas being high, and $K(e_{high-c}) \rightarrow K(e_{safety})$ is an event knowledge meaning that e_{high-c} is a kind of safety event e_{safety}. When $e_{wind-low}$ arrives, the algorithm 2 returns the methane gas model and mine ventilation model as scoped resource models, and returns $K(e_{high-c}) \rightarrow K(e_{safety})$ as related events. The algorithm 3 does the logic deduction to compute composite events with on-line interaction among different deduction fragment on different resource models, which is based on the combination theorem.

Algorithm 3. Partitioned Deduction

Input: Source event e_s, goal event e_g, scoped event set Set_e, scoped resource Set_{res}.

Output: $\rightarrow K(e_g)$ or $\rightarrow \neg K(e_g)$, a proof tree (chain) $PTree$.

1. For a resource $Resource_i$ *in* Set_{res}, *the algorithm gets related event knowledge such as* $K(e_1) \wedge \cdots \wedge K(e_n) \rightarrow K(e)$ *in* Set_e.

2. The algorithm forms a clause set $ClauseSet$ $\{K(e_1) \wedge \cdots \wedge K(e_n) \rightarrow K(e)\}$, *add* $\rightarrow \neg K(e_g)$ *into* $ClauseSet$.

3. The algorithm invokes Algorithm 3.1.

4. If the algorithm 3.1 returns True, the algorithm outputs $\rightarrow K(e_g)$ *and proof tree, and returns. If the algorithm 3.1 returns False, the algorithm jumps to step 1 if* Set_{res} *is not empty. Otherwise, the algorithm invokes Algorithm 3.2.*

5. Goto 3, until the proof is finished.

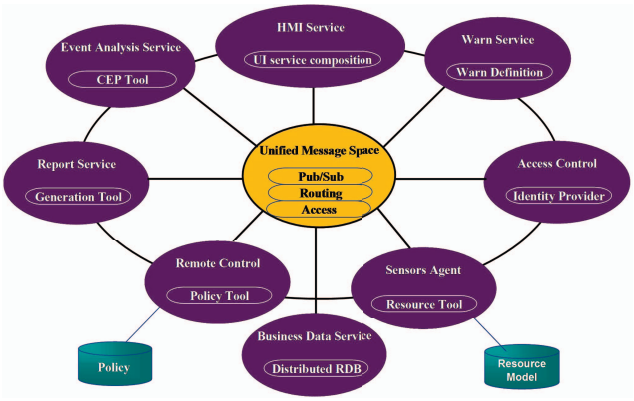

Fig. 3a. The CMCS function

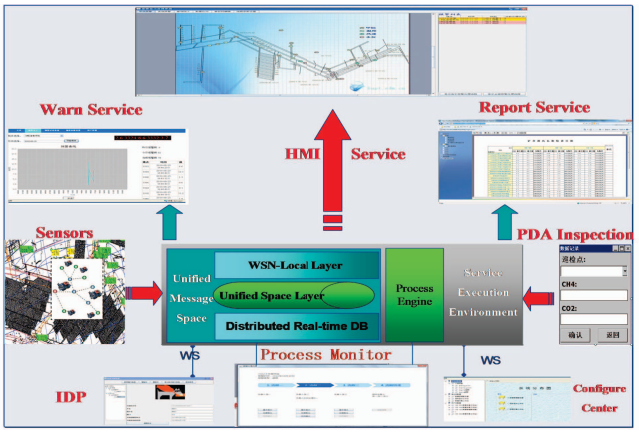

Fig. 3b. The CMCS application in one coal mine

In the above algorithm, the invocation of *Algorithm 3.1* often generates common constraints which are common clauses shared by two resource models. That is to say, the deduction on one resource model produces some clauses which should be satisfiable on the other resource model.

Algorithm 3.1. Getting Common Constraints

Input: A clause set $ClauseSet$, resource model $Resource_i$.

Output: Common constraints, False, or True.

1. If e_s belongs to $Resource_i$, the algorithm adds $K(e_s)$ into $ClauseSet$.

2. If the resource model $Resource_i$ is not empty, the algorithm uses the SMT solver to carry out deduction on $Resource_i$ and $ClauseSet$. Otherwise, the algorithm returns False.

3. If the SMT solver outputs a proof to include common constraints between $Resource_i$ and others, the algorithm returns them. If the SMT solver has proved $\rightarrow K(e_g)$, the algorithm returns True and proof tree (chain). Otherwise it returns False.

Algorithm 3.2. Interaction & Deduction

Input: Common constraints $Interpolant_1,\cdots,Interpolant_m$, Resource model $Resource_j$, $ClauseSet$.

Output: *Common constraints, False, or True.*

1. The algorithm adds $Interpolant_1,\cdots,Interpolant_m$ into $ClauseSet$.

2. The algorithm the SMT solver to carry out deduction on $Resource_j$ and $ClauseSet$.

3. For clauses in deduction, the algorithm tests their satisfiability in the background theory and computes common constraints.

4. If the common constraints are gotten, the algorithm returns them. If the SMT solver has proved $\rightarrow K(e_g)$, the algorithm returns True and the proof steps, i.e., proof tree (chain). Otherwise, it return False.

Compared with traditional CEP systems, it is only required, in our CEP service, that event knowledge and resource models are defined without repeatedly defining event patterns for different goals and different event sources.

VI. Application and Experiment

We implement the CMCS (Coal Mining Monitor-Control) prototype system to show how to use and deploy our CEP service in an IoT application. Fig. 3a illustrates the CMCS function architecture. From it, we know it is a flat structure, where services and components collaborate to achieve the business goal, and the CEP service subscribes events from other services and publishes composite events to them. A set of service tools is provided such as UI composition tool, report generation tool, warn customization tool, CEP knowledge tool, resource modeling tool, etc. Using these tools, an engineer can generate specific services with

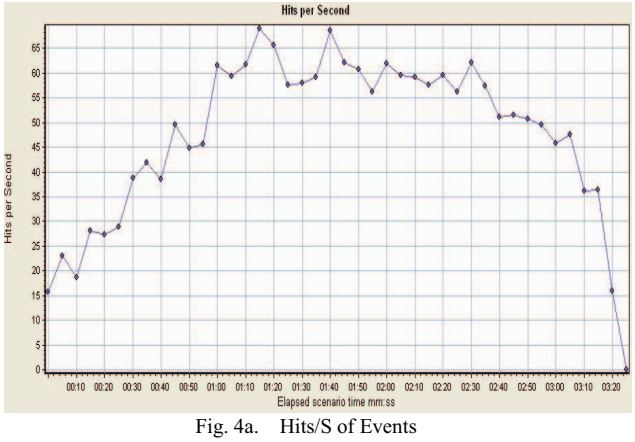

Fig. 4a. Hits/S of Events

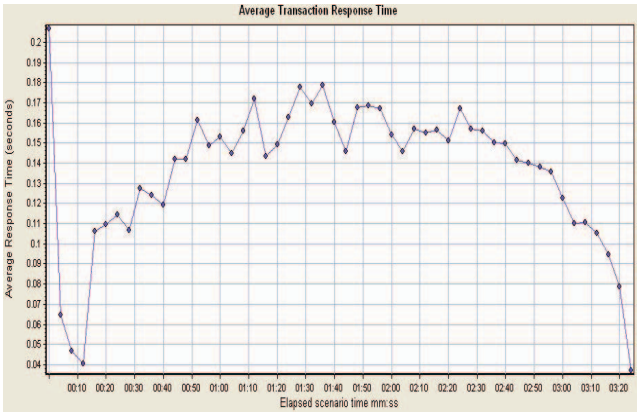

Fig. 4b. Response Time of Event Processing

respect to specific requirements such as HMI service, report service, warn service, and so on. The application in Fig. 3b is made using the tools and functions in Fig. 3a.

In order to test whether our CEP service can efficiently work in such IoT application, some experiments have been done using LoadRunner. We choose Apache ODE as a BPEL orchestrator where the IoT BPEL processes work as the application scheduler. Our CEP service subscribes atomic events and provides composite events to the BPEL processes. The sensors are simulated by the scripts of LoadRunner. The event from sensors only includes CH4 and WIND, and the number of the randomized event is 10000, CH4 and WIND holding 50% respectively. They are deployed on a PC cluster. We focus on the event response time and hits-per-second. From Fig. 4a, we can see that the application needs 3 minutes and 20 seconds to complete all event processing, and the max hits per second is 69, and the average hits per second is 48.309. From Fig. 4b, we can get the average event response time is 0.136 seconds, which implies our CEP service's average processing delay is less than the number.

VII. Related Work

Complex event processing originated from active databases [1], [2], and flourished in the field of business process management [3], [4] to improve the business performance for grim market competition. Certainly, it was also explored in the service-oriented computing area [5], [6], [7]. Good CEP introductory materials are available in [8] and [9]. CEP works in [10], [11], [12] explored different facets of CEP technologies. These works did not focus on IoT applications to address the issues of hybrid event processing, avoiding defining event composition rules and concurrent and distributed processing.

In the 21th century, the SMT technology developed rapidly and many efficient SMT solvers appeared, where the conflict-driven clause-learning algorithm played a key role, some dedicated decision procedures for background theories actually promoted the SMT into practical industries such as linear arithmetic over the rationals, bit-vectors, and their combinations [13], [14]. The problem of combining decision procedures was proposed 30 years ago [15], [16]. However, the two works and lots of succeeding works focused on disjoint or partly disjoint term signatures. In our combination theorem, the term signature is shared.

Description Logic had become one of the main knowledge representations, which was widely used in the OWL web ontology language [23], [24], [25], [26] defined by the World Wide Web Consortium such that it acted as the theoretical backbone for information systems in many disciplines. Because the description logic focuses on concepts and relations, OWL-DL reasoners [27] could not cope with arithmetical operations and function-related reasoning such that they are not suitable for event processing in IoT applications.

VIII. Conclusion

In existing CEP systems used for IoT applications, there are three issues not to be addressed: processing discrete events with evaluating continuous dynamic behavior, avoiding the error-prone and cumbersome description of event composition rules, and decomposing event composition tasks to satisfy the real-time requirements. In this paper, SMT technologies are adopted as a formal basis to define the event knowledge and IoT resource knowledge. Given the formal knowledge of events and IoT resources, a combination theorem is proposed to direct decomposing event processing tasks, where discrete events are processed at the upper Boolean constraint layer based on the resolution method, and continuous computation is done in the background theories. Interest goals are defined to replace event composition rules, which is based on logical deduction, and reduces the error-prone manual description. We finally present the event composition algorithms and evaluation to show our solution is applicable. To learn event composition rules from event occurrences is our future work.

ACKNOWLEDGMENT: This research is supported by the National Key Technology Research and Development Program of China (Grant No. 2012BAH94F02); National Natural Science Foundation of China under Grant No. 61372115, 61132001);Program for New Century Excellent Talents in University (Grant No.NCET-11-0592); National High-tech R&D Program of China (863 Program) under Grant No. 2013AA102301.

References

[1] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. Data Knowl. Eng., 14(1), pp. 1-26, 1994.
[2] A. Adi and O. Etzion. Amit - the situation manager. The International Journal on Very Large Data Bases, 13(2), pp. 177-203, 2004.
[3] R. Ammon, C. Emmersberger, T. Greiner, A. Paschke, F. Springer, and C. Wol. Event-driven business process management. In Proceedings of the 2nd International Conference on Distributed Event-Based Systems (DEBS'08), Rome, Italy, July 2008.
[4] G. Cugola, E. D. Nitto, and A. Fuggetta. The jedi event-based infrastructure and its application to the development of the opss wfms. Transaction of Software Engineering (TSE), 27(9), Sep 2001.
[5] H. Chen, J. Yin, L. Jin, Y. Li, and J. Dong. Jtang synergy: A service oriented architecture for enterprise application integration. In Proceedings of the 11th International Conference on Computer Supported Cooperative Work in Design (CSCWD 2007), pp. 502-507, Melbourne, Australia, Apr. 2007.
[6] M. Cilia, C. Bornh ovd, and A. Buchmann. Event handling for the universal enterprise. Information Technology and Management - Special Issue on Universal Enterprise Integration, 5(1), pp. 123-148, Jan. 2005.
[7] K. Gomadam, A. Ranabahu, L. Ramaswamy, A. Sheth, and K. Verma. A semantic framework for identifying events in a service oriented architecture. In IEEE International Conference on Web Services (ICWS 2007), pp. 545-552, Salt Lake City, Utah, USA, July 2007.
[8] D. Luckham. The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley, 2002.
[9] A. Hinze and A. Buchmann, editors. Handbook of Research on Advanced Distributed Event-Based Systems, Publish/Subscribe and Message Filtering Technologies. IGI Global, 2009.
[10] R. Blanco, J. Wang, and P. Alencar. A metamodel for distributed event based systems. In DEBS '08: Proceedings of the second international conference on Distributed event-based systems, pp. 221-232, New York, NY, USA, 2008.
[11] S. White, A. Alves, and D. Rorke. Weblogic event server: a lightweight, modular application server for event processing. In Proceedings of the Second International Conference on Distributed Event-Based Systems (DEBS 2008), pp. 193-200, Rome, Italy, July 2008.
[12] M. Bauer and K. Rothermel. An architecture for observing physical world events. In 11th International Conference on Parallel and Distributed Systems (ICPADS 2005), pp. 377-383, Fuduoka, Japan, July 2005.
[13] C. Barret, R. Sebastiani, S. A. Seshia, C. Tinelli. Satisfiability Modulo Theories, Chapter 26, pp. 825– 885, 2009.
[14] R. Sebastiani. Lazy Satisfiability Modulo Theories. Journal on Satisfiability, Boolean Modeling and Computation, JSAT 3, 3-4, 141–224, 2007.
[15] G. Nelson and D. C. Oppen. Simplication by cooperating decision procedures. ACM Transactions on Programming Languages and Systems, 2(1), pp. 245-257, 1979.
[16] R. E. Shostak. Deciding combination of theories. Journal of the Association for Computing Machinery, 1(31), pp. 1-12, 1984.
[17] http://www.spass-prover.org/index.html.
[18] Y. Zhang, J. L. Chen. Constructing Scalable IoT Services Based on Their Event-driven Models. Concurrency and Computation: Practice and Experience, to appear, 2015.
[19] Eyal Amir, Sheila McIlraith. Partition-based logical reasoning for first-order and propositions theories. Artificial Intelligence, Volume 162, 2000, pp. 49-88.
[20] William Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. J. Symbolic Logic, 22(3): 250–268, 1957.
[21] Raymond M. Smullyan. First-Order Logic. Dover Publications, New York, NY, 1995. First published by Springer in 1968.
[22] J.A. Robinson. A machine-oriented logic based on the resolution principle. Journal of the ACM, 12(1), pp. 23-41, 1965.
[23] Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. Journal of Web Semantics, 4(3), 2006.
[24] Amandeep Sidhu, Tharam Dillon, Elisabeth Chang, and Baldev S. Sidhu. Protein ontology development using OWL. In Proceedings of the 1st OWL Experiences and Directions Workshop (OWLED 2005), volume 188 of CEUR Workshop Proceedings.
[25] S. Staab and R. Studer, editors. Handbook on Ontologies. International Handbooks on Information Systems. Springer, 2nd edition, 2009.
[26] Heiner Stuckenschmidt, Christine Parent, and Stefano Spaccapietra, editors. Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, volume 5445 of Lecture Notes in Computer Science.Springer, 2009.
[27] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. Journal of Web Semantics, 5(2):51-53, 2007.