Situation-Aware IoT Service Coordination Using the Event-Driven SOA Paradigm

Bo Cheng, *Member, IEEE*, Da Zhu, Shuai Zhao, and Junliang Chen

***Abstract*—Internet of Things (IoT) technology demands a complex, lightweight distributed architecture with numerous diverse components, including end devices and applications adapted for specific contexts. This paper proposes a situation-aware IoT services coordination platform based on the event-driven Service-Oriented Architecture (SOA) paradigm. Focus is placed on the design of an event-driven, service-oriented IoT services coordination platform, for which we present a Situational Event Definition Language (SEDL), an automaton-based situational event detection algorithm, and a situational event-driven service coordination behavior model, which is based on an extended event-condition-action trigger mechanism. Moreover, propose a reliable real-time data distribution model to support the effective dispatching sensory data between information providers and consumers, which is based on the grid quorum mechanism to organize those brokers into a grid overlay network to facilitate the asynchronous communication in a large-scale, distributed, and loosely coupled IoT applications environment. We also illustrate the various illustrations for IoT services coordination and alarming disposal process of coal mine safety monitoring and control automation scenarios, and we also report the measurement and analysis of the platform's performance.**

***Index Terms*—Situation-aware, IoT service, event-driven SOA**

I. Introduction

INTERNET of Things (IoT) technology interconnects large numbers of physical entities and integrates them in cyberspace. Compared with existing telecommunication services or Internet applications, IoT services exhibit certain novel characteristics. Specifically, existing services primarily address issues of "man-machine" and "machine-machine" interactions; in other words, existing network services mainly target a binary problem domain, i.e., the user domain and the cyberspace domain. However, IoT services also need to provide seamless integration and dynamic interaction with the physical world [1-2]. Therefore, IoT services essentially face a ternary problem domain that involves the user, cyberspace, and physical space. IoT systems must cope with a large amount of sensor information, and they need to aggregate and integrate this information, distribute it to participants in different applications, and trigger the corresponding business process collaboration in a timely manner. Therefore, the reasonable IoT service delivery model follows the context-aware→situation detection→intelligent decision→services coordination pattern. The introduction of physical space into the problem domain makes environmental awareness a major characteristic of IoT services, and this real-time information sensing causes IoT service systems to constantly remain in a state of dynamic change. An IoT service system should, on the basis of the processing and integration of sensed information from multiple, large-scale distributed sources, support the rapid coordination of manpower and business processes to interact with physical entities across business domains, or even across organizations, thereby endowing the IoT service system with the characteristics of real-time dynamic variation. Currently, the IoT industry is building many low-level application systems with duplicated development efforts, and a common IoT platform is lacking. Research has been conducted by using middleware technology to achieve integration between the physical world and business services [3-4]. However, the integration process is a major obstacle facing a large number of sensor networks and applications, and it is a very cumbersome task to integrate these applications and sensor networks using non-standard interface protocols. To achieve business integration in an across-domain system, in recent years, researchers have incorporated Service-Oriented Architecture (SOA) [5-6] technology into IoT service delivery systems to resolve the heterogeneity of services and physical entities to facilitate interoperability and dynamic service discovery. However, the traditional SOA standards and technologies are designed primarily for the integration of enterprise-class heavyweight services in an Internet environment; therefore, these tools suffer certain limitations in addressing service delivery in an IoT environment, and to address these limitations, several researchers have attempted to put forward a number of lightweight Web service protocols that are suitable for resource-constrained embedded devices. Overall, existing research work has focused on the use of SOA technology or more lightweight service protocols to achieve interoperability

Manuscript received July 3, 2015; revised October 20, 2015; accepted February 18, 2016.This work was supported in part by the National Natural Science Foundation of China under Grant No. 61132001; Program of National High-tech R&D Program of China (863 Program) under Grant No. 2013AA102301; National Key Basic Research Program of China (973 Program) under Grant No. 2013CB329102.

Bo Cheng, Da Zhu, Shuai Zhao and Junliang Chen are with State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications; Beijing, 100876 China, (e-mail: chengbo@bupt.edu.cn)

among different physical entities and enterprise application systems. Indeed, interconnection and interoperability are a prerequisite for the delivery of IoT services; however, another question is how to effectively disseminate information in a distributed, loosely coupled environment between providers and consumers and how to achieve rapid, dynamic process coordination based on changes in the physical domain.

II. Related Works and Discussion

A. Publish/Subscribe Communication Paradigm

Publish/Subscribe is an asynchronous communication paradigm. It is also dynamic and loosely coupled, thereby completely decoupling event service participants in terms of time, space, and control flow, and it can well satisfy the communication requirements of services in an open, distributed, and dynamic coordination environment. Message Queue Telemetry Transport (MQTT) [7] is a topic-based publish/subscribe communication paradigm, which has been adopted in IoT environment [8], which is a lightweight protocol that aims to use limited processing and memory capabilities, best suited to resource constrained devices and low bandwidth networks. The other similar publish/subscribed based communication paradigms include Extensible Messaging and Presence Protocol (XMPP) [9], Advanced Message Queuing Protocol (AMQP) [10], Data Distribution Service (DDS) [11], and others. The publish/subscribe communication model can distribute the messages in an asynchronous fashion on demand, however, when the scale of the brokers in publish/subscribe communication model is growing up in IoT environment, and the packet delivery rate of data packet will decrease dramatically. Hence, the current publish/subscribe communication model has higher delay and poor scalability disadvantages.

B. SOA and EDA Architectures

Service Oriented Architecture (SOA) is a flexible architectural pattern used for system integration and automation of business processes, which allows easier service composition and coordination over a network. However, when an IoT service application becomes larger and more complex, it may involve an extremely large number of third-party systems or legacy enterprise systems, which requires the IoT service system to have a strong integration capability and adaptability to a heterogeneous and dynamically changing environment, along with the ability to quickly address business requirements through rapid development or the reuse/integration of existing resources. Event-Driven Architecture (EDA) refers to an abstract concept of a responsive system [12-13]. Such a system uses push mode for event communication, provides concurrent responsive processing, and is especially suitable for applications that use loosely coupled real-time communication and support sensing. EDA defines design guidelines for the creation and construction of application systems. Because an event message is transmitted through the Publish/Subscribe method, an event can be sent to multiple subscribers, thereby allowing an enterprise to respond to and process these events in a rapid and appropriate manner.

C. Complex Event Processing

A complex event-processing (CEP) system constantly monitors large volumes of continuously incoming events generated by applications or environments. It uses methods such as rule matching and deduction to detect an event profile and continuously sends out assembled composite events to trigger relevant subsequent processing. Before the event system can detect a composite event, an event modeling language for composite events (event pattern) should be provided. However, most of the existing complex event processing languages, such as Data Stream Processing (DSP) language [14], Continuous Query Language (CQL) [15], also have some limitations to describe the temporal relations expression and complex event pattern, which can easily cause the semantic ambiguity for situational events. Hence, it is difficult to describe the complex situation in IoT service environment.

D. Web Service Coordination and ECA Model

Web Service Business Execution Language (BPEL) [16] is a service orchestration language, which defines the service processes' logic structure and the detailed operations in each step. Web Service Choreography Description Language (WS-CDL) [17] is a service assembly language that describes point-to-point cooperation between service participants. It is an interesting approach to use Event-Condition-Action (ECA) rules for controlling workflow processes. In reference [18], the authors apply the rule-based idea for service composition in an event-driven service-oriented architecture that integrates semantic, Web service, and computing grids to enable seamless integration of information services from large-scale, scattered sources. In reference [19], the authors propose a new approach to the automatic execution of business processes using ECA rules that can be automatically triggered by an active database. Another rule-based composition method is proposed in reference [20]. This system caches pre-defined and new process templates for abstract service workflows in the form of rules. These abstract workflows are mapped to executable service graphs consisting of concrete services. For the FP7 IoT.est project [21], which attempts to develop a framework for service creation and testing in an IoT environment, the architecture design extends the existing IoT reference architecture and enables a semantics-based management of the entire service lifecycle. ThingWorx [22] is a widely-adopted IoT platform that can help companies deliver new value via the design and runtime environment.

E. CoAP and REST Style Approach

Representational State Transfer (REST) [23-24] is an architectural style that makes information available as resources identified by URIs; applications communicate by exchanging representations of these resources using a transfer protocol such as HTTP. This powerful and extensible paradigm is quickly spreading to IoT applications, letting developers of Web-based applications continue to use their existing skills. Constrained Application Protocol (CoAP) is a specialized Web transfer protocol for resource constrained nodes and networks, which conforms to the REST style and abstracts all objects in the network as resources. Each resource corresponds to a unique universal resource identifier (URI) from which the resources can be operated stateless, including GET, PUT, POST, and DELETE. A CoAP-based IoT device management solution for WSN is proposed in [25-27], and device management-oriented functions, resource identities, and protocols are developed. In our earlier study, a RESTful API interface [28] was designed for monitoring and control of an underground sensor network, and all types of monitoring parameters were collected and transmitted to a monitoring center.

F. Motivations of This Paper

In general, pure SOA and EDA each have their own limitations and application scenarios for which they are more suitable. SOA primarily begins by decoupling the system. It focuses on decomposing the entire application into a series of independent services and defines various standards and infrastructures to facilitate the reuse of these services, i.e., to allow them to be easily used by applications on various platforms. Because EDA uses an asynchronous event Publish/Subscribe mode, the generation of an event can trigger the concurrent execution of one or more services. When a certain event occurs, different services can be triggered automatically, endowing the system with the capability of real-time sensing and rapid response to events in a loosely coupled distributed computing environment. Thus, EDA and SOA complement each other, and a certain degree of service coordination can be achieved among mutually independent service systems through the event mechanism. This behavior is well suited to the characteristics of the IoT, which are "high autonomy inside a domain and efficient coordination across domains"; it both improves real-time response to constantly changing business requirements and minimizes the impact on the existing application system to allow a large-scale distributed IoT service application to be easily developed and maintained. By integrating the advantages of SOA and EDA, an event-driven and service-oriented situation-aware IoT service coordination platform is proposed, and the on-demand distribution of situation-aware information and dynamic event-driven service coordination are implemented. To summarize, the primary contributions of this paper are as follows:

1. A comprehensive and systematic event-driven, service-oriented IoT service coordination platform architecture, which effectively integrates the advantages of SOA and EDA, is proposed. SOA technology is used to resolve interoperability issues among large numbers of heterogeneous services and physical entities as well as application reusability issues in a large-scale distributed IoT environment; EDA technology is used to address the problem of the cross-business-domain, or even cross-organization, on-demand distribution of sensed information as well as the problem of dynamic event-driven service coordination.

2. A Situational Event Definition Language (SEDL) is designed. Various types of information monitored by sensors are defined as atomic events, and a series of concise and strictly temporal event operators are used to define complex event pattern with rich semantics; simultaneously, flexible event selection and consumption strategies are provided for users to create flexible configurations, and an automaton-based situational event detection algorithm is proposed.

3. A situational event-driven service coordination behavior model is proposed. This model is based on an extended event-condition-action (ECA) trigger mechanism, which translates the service's global process model into a series of local processes and event flow models to generate a set of executable ECA rules and define a coordinated execution sequence for global services to implement situational event-triggered active coordination for the global coordination process.

4. A reliable real-time data distribution model is proposed, which is based on the concept of the grid quorum mechanism and organizes Publish/Subscribe brokers into a broker-to-broker grid overlay network for efficient topic searching. It also leverages a topic searching algorithm and a one-hop caching strategy to minimize search latency.

The remainder of the paper is organized as follows. Section 3 describes the application scenario. Section 4 describes the proposed architecture. Then, the typical scenario is illustrated in section 5. Finally, conclusions and possibilities for future work are described in section 6.

III. Application Scenario

Safety has become a perpetual topic in coal mining, but it is fundamental in the process of coal production. Therefore, safety is a serious issue for coal mining that needs to be addressed at the highest level. Safety problems occur due to the underground working conditions of coal mines, which involve many hazards, such as methane gas, coal dust, unstable wall rocks, and spontaneous fires in coal seams. The traditional business information system has been unable to meet the safety requirements for coal production and is also unable to address the serious problem of coal mine safety. Sensors and other electronic devices used for underground environmental monitoring should have the capacity for rapid and flexible deployment. However, multiple low-level events from multiple sources detected by sensors do not appear to have any separate relationship, and safety alarms depend on reasonable detection of complex pattern obtained by comparing these multiple sensor events; unusual or less obvious correlations and sensor data should be enriched by the addition of related information from each event to more clearly illustrate how the many simultaneous events are related. In particular, when a safety alarm trigger condition is satisfied by the corresponding complex situation pattern, a higher-level disposal process event should be created, and a human or automated process should be invoked when the trigger condition is achieved. Therefore, it is necessary to detect

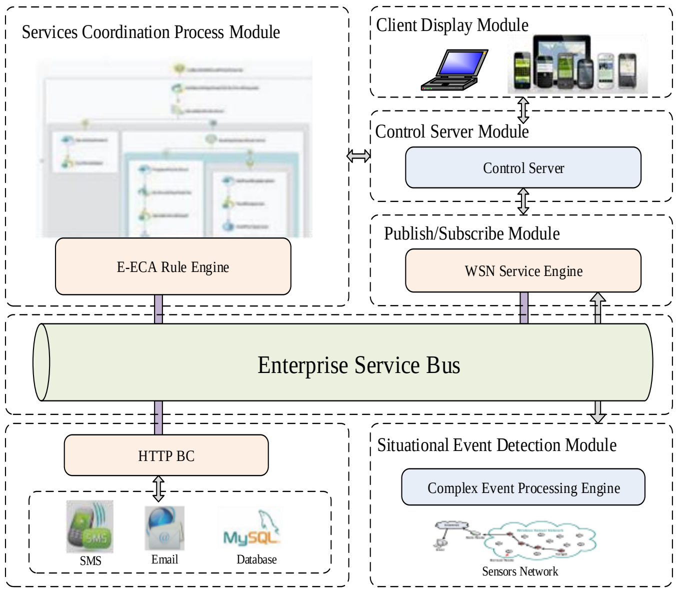

Fig. 1 Situation-Aware IoT service coordination platform architecture, that contains a situational event processing module, a Publish/Subscribe module, a service coordination process module, a control server module, and a client display module.

complex incident pattern and coordinate services to form a corresponding disposal process to improve underground coal mine safety.

IV. Proposed System Architecture

The event-driven, service-oriented IoT service coordination platform is based on SOA and uses Enterprise Service Bus (ESB) as its infrastructure. It effectively integrates five functional modules, including a situational event processing module, a Publish/Subscribe module, a service coordination process module, a control server module, and a client display module. HTTP Binding Component (HTTP BC) is a component, which is used to receive request messages from the clients. The overall system architecture is shown in Fig. 1.

The situational event processing module receives and processes the original sensor-monitored data stream, builds a model for each event, and executes querying and filtering in accordance with a user-defined query rule; it then publishes events that match the rule with a predefined subject to the Publish/Subscribe (Pub/Sub) module.

The Publish/Subscribe module uses the Publish/Subscribe method to facilitate message interaction in the system. The Pub/Sub module acts as a broker between the situational event's publisher and the relevant subscribers in the system, decouples the publisher and subscribers in terms of space and time, and provides external interfaces such as publication, subscription, notification, and subscription cancellation.

The service coordination process module provides an active model of service coordination behavior that describes dynamic behavioral attributes such as the global coordination process, local processes, process flow, event transmission, and state transition and performs situational event-driven service process coordination.

The control server module is responsible for subscribing to situational events of interest and triggering the service coordination process upon receiving such an event; it also interacts with the process to broker interactions between the service coordination process and human users, stores the state of the service coordination process, and actively pushes it to the client display module for display.

The client display module is primarily responsible for interacting with the control server module. When it receives a state-change message pushed from the control server module, it queries the database, updates the display state, and provides the user operation interface, which waits for user operations and interacts with the service coordination process.

A. *Event and Situation Model*

In an IoT system, an event is regarded as a particular abstraction of the environmental context. Event-based description enables more vivid modeling for an IoT service scenario. Certain event-related definitions are provided as follows.

Definition 1 Event: An event is a specific activity or state change in an object of interest. A specific event instance is denoted by e. Every event notification is associated with a corresponding event class, attribute, and occurrence time. A particular event can be classified as a primitive event or a composite event.

Definition 2 Primitive Event: A primitive event is atomic and indivisible and occurs at a specific time. A primitive event can be denoted by $e_p = e_p(id, a, t, l)$, where id is the primitive event's unique identifier; $a = \{a_1, a_2, ..., a_m\}$, $m>0$, which is the primitive event's attribute set; t is the primitive event's occurrence time; and l is the primitive event's location of occurrence.

Definition 3 Composite Event: A composite event is defined as $e_c = e_c(id, a, c, t_s, t_e)$, $t_e >= t_s$, where the first two items, id and a, are similar to their corresponding definitions for a primitive event; $c = \{e_1, e_2, ..., e_n\}$, $n>0$, which is an event set that includes several primitive events or composite events, whose elements collectively constitute the composite event; and t_s and t_e are the starting and ending times of the composite event, i.e., this composite event occurs in the time range from t_s to t_e. Different events can be combined to generate a composite event. A composite event inherits all characteristics of its source events, and its occurrence time is also determined by the source events' occurrence times. This combination of events is defined by the event combination relation as follows:

Definition 4 Event Combination Relation: When events e_1, e_2, ..., e_n are combined by the event constructor into a composite event $e\in E_c$, the event combination relation is denoted by $\{e_1, e_2, ..., e_n\}\succ e$, where each e_1, e_2, ..., e_n is either a primitive event or a composite event.

Definition 5 Event Space: The set of all possible events in a specific service system is called the event space, denoted by E_{space}. An event space should contain all primitive event sets E_p and composite event sets E_c in the service system, i.e., $E_{space} = E_p \cup E_c$.

Definition 6 Event Class: In the event space $\boldsymbol{E}_{space}$, the event class $E^C \subseteq \boldsymbol{E}_{space}$ is a series of events whose condition C satisfies $E^C = \{e \mid e \in \boldsymbol{E}_{space} \wedge C(\{e\})\}$.

Definition 7 Condition: The condition describes the set of properties shared by events (event instances) of the same class.

In any event definition language, the event profile is always the core concept. Any event profile should capture the associated events in a dynamically extended event sequence, and only a matching event profile can satisfy the description rule's semantic constraint. The following definitions are given:

Definition 8 Event Trace: An event trace $E_{trace[ts,\ te]}$ is an event sequence observed in some period $[t_s, t_e]$, where t_s is the observation's starting time and t_e is the ending time; the latter can take an infinite value.

Definition 9 Event Profile: An event profile P_e is a conditional description of an event. An event detection system needs to use P_e to perform a matching evaluation for an event trace.

Definition 10 Situation: The situation describes a collection of states of relevant entities and is a logical description representing a sequence of contexts with invariant characteristics.

B. Situation Event Definition Language

In the event-based, coordination-service-oriented Situation Event Definition Language (SEDL), various types of monitored information are treated as events, and this language uses concise and strictly temporal constraint-abiding event operators to define a complex event profile with rich semantics. It also provides flexible event strategies for users to create flexible configurations. SEDL's superior definition capability makes it suitable for various service-coordination-aware scenarios. The general rule structure of SEDL is defined below:

SEDL RULE STRUCTURE	
Situation	*SituationEvent* (*Attr*$_1$:*Type*$_1$,…, *Attr*$_n$: *Type*$_n$)
TrigFrom	*Event Profile*
Where	*Attr*$_1$=*a*$_1$,…, *Attr*$_n$=*a*$_n$
[Consuming]	*e*$_1$,…, *e*$_n$

In the diagram, **Situation** in the first line defines the event structure for a specific situation, whose parameters include the event's attributes and attribute types (the type is optional). **TrigFrom** in the second line defines a complex event profile structure that consists of a series of simple events that can be connected by logical relations such as AND or OR. The above two lines specify a situational event and its components. The **Where** clause in the third line specifies the constraint conditions that should be satisfied, as defined in the rule, by the situational event's corresponding attributes $Attr_1$, …, $Attr_n$. **Consuming** in the last line is optional and specifies the event consumption strategy. The event selection strategy is primarily reflected in the first three lines of the rule structure.

Here, a situational event is created from basic events or other complex events via calculation or combination. An operator or event relation pattern that is used to define an event relation profile may apply to a single event or multiple events. For instance, unary pattern describe single events and include the pattern for average-value events, maximum-value events, repeated-occurrence events, and range-occurrence events, among others; N-nary pattern define the relations between or among multiple events and include sequential relations and aggregate relations, among others. Detailed event relation pattern include the following:

- Unary Patterns

✓ RepN

RepN (event, n, T_1, T_2) means that between times T_1 and T_2, the event of interest to the system, "*event*," repeats *n* times.

✓ PointE

PointE (event, T) means that at time *T*, the event of interest to the system, "*event*," occurs.

✓ DurE

DurE (event, T_1, T_2) means that between times T_1 and T_2, the event of interest to the system, "*event*," continues to occur.

✓ Not

Not (event, T_1, T_2) means that between times T_1 and T_2, the event of interest to the system, "*event*," does not occur.

✓ Avg

Avg (event, avg_val, T_1, T_2) means that between times T_1 and T_2, some event within the set of events of interest, "*event*," occurs, and its particular value of interest reaches *avg_val*. This profile is only applicable to an event whose occurrence includes some measurable value in which the system is interested.

✓ Inc

Inc (event, T_1, T_2) means that between times T_1 and T_2, the particular value of interest *avg_val* of an event within the set of events of interest, "*event*," continues to rise. This profile is only applicable to an event whose occurrence includes some measurable value in which the system is interested.

✓ Max

Max (event, T_1, T_2) obtains the event with the maximum value of interest in the event set "*event*" between times T_1 and T_2. This profile is only applicable to events whose occurrence includes some measurable value in which the system is interested.

✓ Min

Min (event, T_1, T_2) obtains the event with the minimum value of interest in the event set "*event*" between times T_1 and T_2. This profile is only applicable to events whose occurrence includes some measurable value in which the system is interested.

- N-nary Patterns

✓ Any

Any (m, event$_1$, event$_2$, …, event$_n$, T_1, T_2) means that m events of *event$_1$*, *event$_2$*, …, *event$_n$* occur between time T_1 and T_2.

✓ Or

Or (event$_1$, event$_2$, …, event$_n$, T_1, T_2) means that one event of *event$_1$*, *event$_2$*, …, *event$_n$* occurs between time T_1 and T_2.

✓ And

And (event$_1$, event$_2$, …, event$_n$, T_1, T_2) means that all events of *event$_1$*, *event$_2$*, …, *event$_n$* occur between time T_1 and T_2.

✓ SameT

SameT (event$_1$, event$_2$, … T_1) means that *event$_1$*, *event$_2$*… occur simultaneously at time T_1.

✓ Sub

Sub (event$_1$, event$_2$, T_1, T_2) means that between times T_1 and T_2, *event$_1$* occurs and *event$_2$* does not occur.

✓ Seq

Seq (event$_1$, <interval>, event$_2$, T_1, T_2) means that between times T_1 and T_2, events *event$_1$* and *event$_2$* occur consecutively, separated by the time interval "*interval*."

In general, different applications have different selection criteria for identifying events that satisfy a sequential relation and have different reusability requirements for different events. To illustrate this statement, consider the following scenario: Assume that we define an event profile in a certain period T as (E_1; E_2). If the event's trace in T is (e_1, e_2, e_3, e_4), where e_1, $e_2 \in E_1$ and e_3, $e_4 \in E_2$, and if it is not trivial to determine which specific group of events is a better match to the event profile's definition, then the possible event instance profiles are (e_1, e_3), (e_1, e_4), (e_2, e_3) and (e_2, e_4). To address such an ambiguous scenario for event instance selection, SEDL defines 3 basic event selection operators:

All (Choose all events): when there are multiple matching event instances, all occurring event instances are chosen.

First (Choose first event): when there are multiple matching event instances, only the first occurring event instance is chosen.

Last (Choose last event): when there are multiple matching event instances, only the last occurring event instance is chosen. When an event selection operator, such as ($E_{1\mathit{Last}}$; E_2), is applied to the event profile (E_1; E_2) of the above event trace, particular event instance profiles are obtained; in this case, the chosen profiles are (e_2, e_3) and (e_2, e_4). Other strategies are illustrated in Fig. 2.

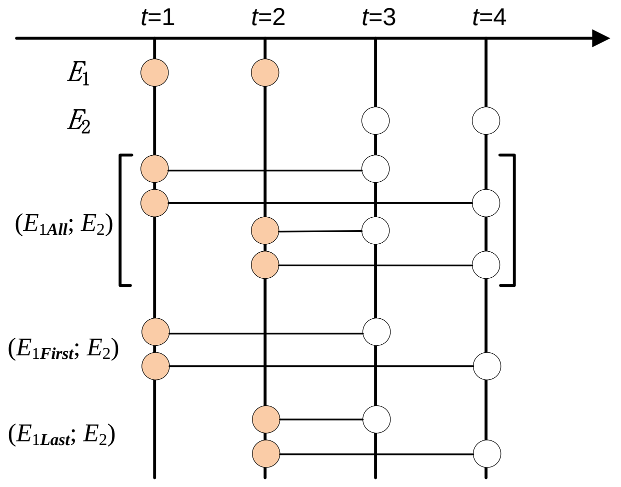

Fig. 2 Event selection strategy

To describe the reliabilities of these event selection strategies over time, *Time* () is defined as a time-dependent function that takes the identifiers of all events as parameters and returns the occurrence time of the event with the specified identifier relative to the current moment. Then, all event selection operators introduced above can be formalized using TRIO logic [29]. Furthermore, SEDL also supports certain special event selection operators, such as ***i-th*** (Choose the i^{th} event from the First). The formal definitions can be given similarly to the above definitions but will not be elaborated here.

The event consumption strategy is specified individually for each application. It defines when an event is chosen from among candidate events and whether that event should be deleted or whether it should be saved to trigger subsequent event detections. In the structure of the SEDL language, the **Consuming** clause is used to define the event consumption strategy. The user can easily specify events that are not "consumed" in subsequent detections of the same rule. To provide a formal semantic definition for the **Consuming** clause, a new TRIO predicate is introduced: *Consumed*. For any given rule identifier *r* and event identifier *d*, before the event with identifier *d* is consumed, *Consumed*(*r*, *d*) is always *false*, and this value only becomes *true* after this event is consumed. In other words, the *Consumed* operator should have the following attributes:

$$Always\ \forall d \in D,\ \forall r \in N\ (Consumed(r, d) \rightarrow \forall t > 0, Futr(Consumed(r, d), t))$$
, where, Futr is a temporal operator.

The SEDL event consumption strategy and event selection strategy have both been introduced above. Because event-processing applications span a broad range, different event selection and consumption strategies are defined for event queries depending on the requirements of different applications. The event selection and consumption strategies define the event reuse and time constraints that are used to impose more detailed constraints on event composition. Fig. 3 shows various event pattern for different combinations of event selection and consumption strategies.

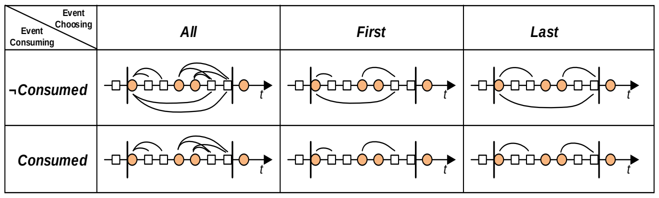

Fig. 3 Event pattern for event selection & consumption strategies

In this example, an event profile is a set of sequential events (E_1; E_2), a circle symbol represents an event instance that belongs to E_1, and a square symbol represents an event instance of E_2. Every set of composite events whose matching condition is true is represented by an arc connection. The time between the two bold vertical lines is the time range for event profile detection. The labels ***All***, ***First***, and ***Last*** along the horizontal direction indicate different event selection operators, whereas the labels ***¬Consumed*** and ***Consumed*** along the vertical direction indicate different event consumption strategies.

C. Situational Event Detection Algorithm

The core concept of SEDL is to use strict temporal logic to describe sequences of events. Confirmation of any event profile must wait for the occurrence of the final event in the event sequence corresponding to the relevant rule. To support SEDL, an automaton-based situational event detection method is proposed. First, a situational event automaton model is provided:

Definition 11 Situational Event Automata (SAs). A situational event automaton (SA) is defined as a six-tuple SA=<*S*, *T*, S_s, S_e, *In*, *Out*>, where

1) *S* is the SA's state set;
2) $T \subseteq S \times S$ represents the transition relations between states, which typically must satisfy certain event constraints;
3) S_s is the SA's initial state;
4) S_e is the acceptable state for the SA, i.e., the state for which a matching event profile is detected;
5) *In* is the event stream that enters the SA; and
6) *Out* is the successfully detected composite event stream.

The primary design principle of automaton-based detection is to construct corresponding automata for each composite event profile and to obtain the corresponding state set and transition set, the corresponding input event domain for each state, and the corresponding event domain for each transition; then, when a primitive event arrives, these constraints are used for matching. If an acceptable state without an outward transition is eventually reached, then a successful match is obtained, and a composite event is successfully detected. To identify the SEDL rule, a specific rule should first be translated into an automaton model; this model is determined by the occurrence order of the events and the various constraints specified by the rule and consists of one or more specific event sequence models. The general procedure is as follows: Based on the rule definition, rule *r* is decomposed into one or more event sequence models M_{ES}; all event sequences (*ES*) are linear deterministic automata, every event related to *r* represents a state in the sequence model, and the event contents and time constraints that can trigger state transitions between the different states of the sequence are shown (S_s is the automaton's initial state; S_e is the final state). After the rule is decomposed into event sequence models based on the constraints and relations of the different states as defined by rule *r*, all sequences are assembled into a common automaton model. To describe how SEDL rule-based automata can be used to identify a specific situation, a processing algorithm for a single M_{ES} is first provided. When a certain event *e* arrives, this M_{ES} begins processing following the steps listed below:

Algorithm: Event automaton processing (EAP) algorithm**.**

```
1) Initialize the event sequence in $M_{ES}$ as $ES_0$.
2) Check whether the incoming event $e$'s class, content, and time identifier satisfy any state transition condition of $M_{ES}$.
3) If not, discard $e$.
4) Otherwise, if in $M_{ES}$, some state $s_i$ of some $ES_i$ can migrate to $s_{i+1}$ when triggered by $e$, then create a copy $ES_{i+1}$ from this $ES_i$.
5) Change the state of $ES_i$ from $s_i$ to $s_{i+1}$.
6) If $s_{i+1}$ is not the final state of $M_{ES}$, go to 2) and continue detection.
7) Otherwise, return the detection result.
8) If some $ES_i$ ($i \neq 0$) has exceeded the time constraint, then delete this $ES$ and go to 2).
```

It should be noted that when *ES*' is created, the original *ES* (not ES_0) is still in state $s_{i.}$ If the processing procedure discovers that some *ES* has exceeded the time constraint, then this *ES* is deleted. However, because the first initialized ES_0 is always in the state that has not been activated by the event, it never needs to be deleted.

D. Situation-Aware IoT Service Coordination

Event-Condition-Action (ECA) rules were originally used in active database systems to provide event-driven, instantaneous responses for conventional database systems. The entire process requires no intervention from users or external applications. In service coordination process control, in addition to sequential control flow mode, composite control flow modes are also available, such as AND-split, AND-join, and XOR-split. However, these modes cannot be expressed in an ECA rule model. To address this shortcoming, Fig.4 resents an E-ECA rule expression method that represents an enhancement of the functionality of ECA rules. Based on ECA, this method of expression offers an increased number of options regarding the action behavior of a given rule.

ON	*Event*
IF	*Condition*
Then DO	*Action*
Else DO	*Alternative Action*

Fig. 4 E-ECA expression diagram

The execution of E-ECA typically follows one of the modes listed below:

(1) E-A mode: after a service object receives a trigger event E, it executes action A and migrates to the next state; i.e., as long as any event that matches the condition occurs, the action in the "Then" branch is executed.

(2) E-C-A mode: only when a service object receives a trigger event E and condition C is true does it execute a certain action A and migrate to the next state.

(3) E-C^n-A^n mode: only when a service object receives a trigger event E and condition C^n is true does it execute A^n and migrate to the next state. The notation used here is defined as follows: C^n = {$\text{Condition}_1 \not\subset \text{Condition}_2 \not\subset \ldots \not\subset \text{Condition}_i$}, $\not\subset \in \{\wedge, \vee\}$, $i \geq 1$; A^n = {$\mathit{Action}_1 \pounds \mathit{Action}_2 \pounds \ldots \pounds \mathit{Action}_j$}, $\pounds \in \{\wedge, \vee\}$, $j \geq 1$.

To establish a model for IoT service coordination control flow, the following control flow modes should be considered: a sequential control structure, parallel actions, alternating actions, and iterative actions. When an E-ECA rule is used to implement IOT service coordination process control in a sequential structure, after every E-ECA rule completes the execution of its own action, the process migrates to a state in the next rule. The sequential control structure is illustrated in Fig. 5.

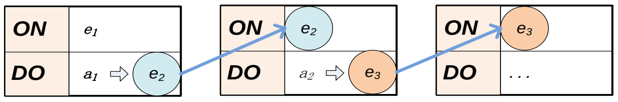

Fig. 5 Sequential control structure

Fig. 5 shows that after rule 1 completes the execution of action a_1, it generates or receives event e_2; event e_2 then triggers the execution of rule 2, which completes action a_2, and similarly, a state transition occurs that executes rule 3. In this manner, a sequential service process control structure is implemented.

The parallel-branch structure is also called the AND branch structure and can be modeled in various ways. Considering the front-end AND-split mode as an example, one method is to use the same trigger event to trigger the execution of a rule in a different service space, as shown in Fig. 6. Another method is to generate multiple events from a previous action to trigger subsequent parallel actions; additionally, the same event can be used to trigger parallel actions under different conditions.

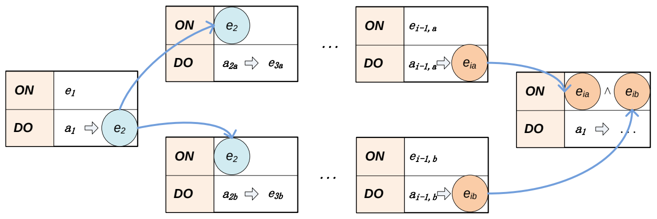

Fig.6 Parallel-branch control structure

Fig. 6 shows that after the initial rule completes action a_1, an event e_2 is generated or received and simultaneously triggers two different rules, creating two branches, (e_2, a_{2a}, e_{3a}) and (e_2, a_{2b}, e_{3b}). At the tail end, e_{ia} and e_{ib}, generated by branches ($e_{i\text{-}1,\ a}$, $a_{i\text{-}1,\ a}$, e_{ia}) and ($e_{i\text{-}1,\ b}$, $a_{i\text{-}1,\ b}$, e_{ib}), respectively, trigger and implement the AND-join mode.

The "select"-branch structure can be further divided into the XOR-branch structure and the OR-branch structure. In the former structure, the process flow can select and execute one from among multiple transition paths to reach the next state. An ON-IF-THEN-ELSE mode E-ECA rule can be used to implement control for an XOR-branch control structure, as shown in Fig. 7. Its end path can use the disjunction of two events triggered by two different branches as the E to trigger subsequent actions.

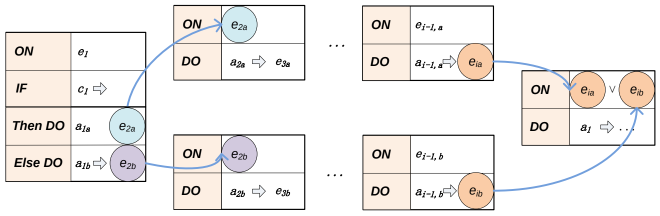

Fig. 7 XOR-branch control structure

One representation of the OR-branch control structure is shown in Fig. 8. The same event triggers several rules under different conditions.

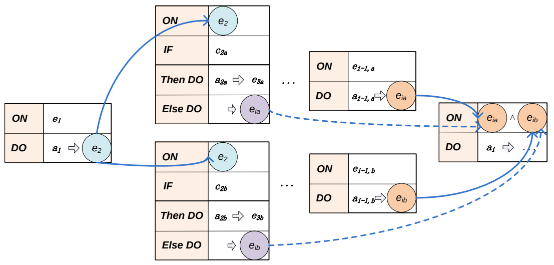

Fig. 8 OR-branch control structure

An E-ECA rule-based control scheme in a loop structure includes two components. The first is the control scheme for the DO-WHILE loop, and the second is the control scheme for the WHILE-DO loop. Both can leverage the XOR-split mode to facilitate control. A DO-WHILE loop is illustrated in Fig. 9.

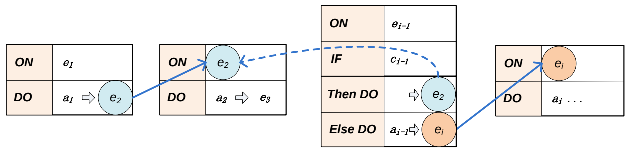

Fig. 9 DO-WHILE loop control structure

Fig. 9 shows that after rule 2 receives trigger event e_2, it first executes action a_2 in the loop. This order is followed until rule i-1 is executed. If the loop condition for rule i-1 is true, then the process proceeds to rule 2 and executes the loop body; otherwise, if the loop condition is false, then after the execution of action $a_{i\text{-}1}$, it jumps out of the loop and executes rule i. In WHILE-DO loop control, when rule 2 receives event e_2, condition c_2 is evaluated. If that condition is true, then the loop body is executed; otherwise, after the execution of the action, the process jumps out of the loop, as shown in Fig. 10.

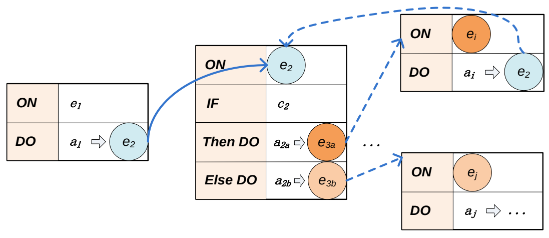

Fig. 10 WHILE-DO loop control structure

In an IoT environment, a global service coordination process G-Process is primarily composed of E-services or S-services. An ECA rule primarily invokes a service or generates an event notification to coordinate these two types of services. All events have classes and attributes, and a rule trigger may be either a primitive event or a composite event composed of various event constructors.

E. Data Distribution Overlay Network

Unified message distribution network uses the Publish/Subscribe mechanism to distribute messages, which provides event publication, subscription, notification, and routing functions. Therefore, a unified message space can provide distributed coverage managed by a group of event brokers, as shown in Fig. 11.

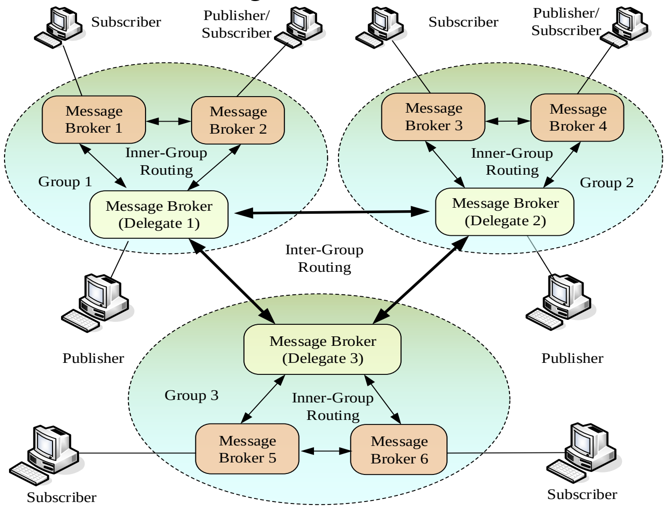

Fig. 11 Publish/Subscribe-based distribution network

In the clustered topology shown in Fig. 11, the message distribution network should support both intra-cluster and inter-cluster routing. In every cluster, a particular event broker, named the delegate broker, is responsible for inter-cluster routing. A publisher or subscriber connects to an event broker in the unified message space and publishes or subscribes to an event of interest via this broker. When a broker receives a subscription from a client, this broker forwards the subscription to an adjacent node inside a cluster. The delegate broker for this cluster is responsible for forwarding subscription information to other clusters. Likewise, when a broker receives a published event from a client, it forwards the event, via the message distribution network, to brokers that match the subscription. Then, these brokers deliver the event to interested subscribers. While in the dynamic IoT network environment, the network topology is dynamic; however, the routing infrastructure must be reconfigured as the system evolves, i.e., it must adapt to the changes in the underlying physical network without interrupting the normal operation of the system. In particular, we extend the existing tree-based message routing with a notion of reconfiguration, and only routing information is established dynamically as new subscriptions, un-subscriptions, and possibly advertisements are issued. Specifically, the Publish/Subscribe brokers are organized into a broker-to-broker overlay network based on the grid quorum mechanism. A quorum [30] is a set of nodes in the network, and the group of all quorums forms a quorum system. Then, given two quorums, q_i and q_j, there exists a node $k \in q_i \cap q_j$; these two quorums have a non-empty intersection, which can ensure at most two hops between any two nodes. A grid quorum system Q is a universal set of nodes U={1, 2, ..., n} that is arranged as a $|\sqrt{n} \times \sqrt{n}|$ grid matrix. A sub $q_i \subseteq Q$ is defined as the set of nodes on the row and the column passing through node i in the array. Thus, it is clear that $q_i \cap q_j \neq null$, for any node i and j, $1 \leq i, j \leq n$. Based on the definition, for any node i, the size of q_i is $2|\sqrt{n}| - 1$ in a grid quorum system. In a grid-quorum-based Publish/Subscribe overlay network, each broker node can only maintain a broker set that contains $O(\sqrt{n})$ nodes, where n is the number of broker nodes. At the same time, each broker node can reach any of these nodes within two hops. More details are included in our previous work [31]. A root server exists to maintain the distributed grid quorum system structure. The structure information consists of the broker lists for the nodes and the QuorumKey, IP, and Index information. The QuorumKey of a node is its identifier in the grid quorum system. The Index of a node is the node's position in the grid matrix, which can be calculated by the root server. When a broker node joins the grid-quorum-based Publish/Subscribe overlay network, it will contact the root server and receive its QuorumKey in the grid, its position index and its broker list in the matrix. The initialization time for the overlay network depends on the number of broker nodes. Each broker node runs a simple maintenance program that periodically communicates with the other existing broker nodes to maintain the list of topics and communicates with the root server to maintain the broker list.

V. Typical Scenarios

Coal mine safety monitoring is a typical IoT application scenario. Sensors deployed in all coal mine areas send back to the information center various measured environmental data, such as temperature, humidity, location, methane (CH_4) concentration, and oxygen concentration. Now, assume that there is a need to implement a gas alert in certain monitored areas. Depending on the particular environment, the application requirements, and user preferences, the potential danger of gas explosion can be defined in various ways, as shown below:

Situation 1	GasAlert (MeasuredCH_4: double)
TrigFrom	Smoke () and
First	CH_4Concentration (Value > 5%)
	Within 5 minutes from Smoke
Where	MeasuredCH4= CH_4Concentration.Value

Situation 2	GasAlert (MeasuredCH_4: double)
TrigFrom	Smoke () and
All	CH_4Concentration (Value > 5%)
	Within 5 minutes from Smoke
Where	MeasuredCH_4= CH_4Concentration.Value
Comsuming	CH_4Concentration

Situation 3	NotAscendingCH_4Cc(Area: string)
TrigFrom	CH_4Concentration (Value = β.v and
	Area=β.a_0) as C_1 and
Last	CH4Concentration (Value >= β.v and
	Area=β.a_0) as C_2
Within	5 minutes from C_1 and
Not	CH_4Concentration (Area=β.a_0)
	between C_2 and C_1
Where	Area= C_1.Area

Situation 4	GasAlert (Area:string, MeasureCH_4:double)
TrigFrom	Smoke(Area=Pa.a_0) and All CH_4Concentration
	(Value>5% and Pa.a_0) within 5 minutes from Smoke and
All	O_2Concentration(Value>30% and Area=Pa.a_0)
	Within 5 minutes from Smoke and
Not	BlowerOn(Area=Pa.a_0) between Smoke and O_2Concentration
Where	Area=Smoke.Area and MeasuredCH_4=CH_4Concentration.Value

Compared with scenario (1), the scenario described by this rule adds an association with an oxygen (O_2) concentration event. This rule, in the construction of a gas alert scenario, primarily captures two automaton event sequences, one of which consists of a CH_4 concentration event and a Smoke event and the other of which consists of an O_2 concentration event and a Smoke event; the two share the Smoke event. In fact, this situation is common to many application scenarios. As mentioned before, the confirmation of any event profile must wait for the occurrence of the final event in the event sequence corresponding to the relevant rule, and this final event also determines the triggering time of the situational rule. In addition to the above shared association, in the above scenario, the parameter operator ***Pa*** is also used to impose the constraint that the CH_4, Smoke and O_2 concentration events must be co-located and the ***Not*** operator is used to impose the constraint that between the Smoke and O_2 concentration events, no "BlowerOn" event occurs. Based on the above descriptions, the SEDL rule (let us call it rule *r*) can be further translated into the event detection automaton model. The general procedure is as follows: based on the rule description, rule *r* is decomposed into one or multiple event sequence models M_{ES}; all event sequences (*ES*) are linear deterministic automata; every event related to *r* represents a state in a sequence model; and event contents and time constraints that can trigger a state transition between different states of the sequence are represented, as shown in Fig. 12 (S_s is the initial state of the automaton; S_e is the ending state).

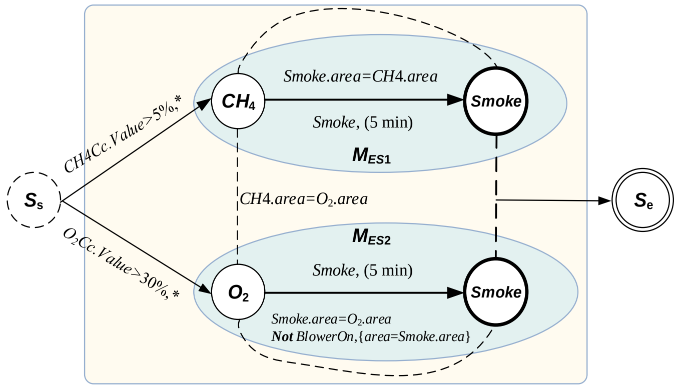

Fig. 12 Event-based situation detection automata

After a rule is decomposed into event sequence models, the constraints and relations between different states that are defined based on rule *r* (such as co-location relations or ***Not***-operator-defined conditions, as in Fig. 12) can be used to integrate the various sequences into a common automaton model. In the following paragraph, situation (1), a specific detection instance, is used to illustrate automaton processing for situation detection. For simplicity, the monitoring period considered for this example is 1 min. In Fig. 13, a set of monitoring events is listed in the upper right corner.

1) At t=0, the initial sequence ES_0 of one of the event sequence models, M_{ES1}, is in the event-awaiting state.

2) At t =1, the CH_4 concentration event C_1 enters the automaton. Once its class, content, and time are examined, it is determined that C_1 satisfies the requirements for triggering the ES_0 state transition. Thus, ES_1 is created, and its state transitions to state CH_4. Likewise, at t=2, C_2 enters the automaton; thus, ES_2 is created, and the state of ES_2 transitions to CH_4.

3) At t=4, the Smoke event S_1 arrives. Because the location is $Area_5$ and, prior to this event, no CH_4 concentration event matching the condition has been detected in $Area_5$, S_1 is discarded.

4) At t=7, the ES_1 created at t=1 has passed its time constraint of 5 min, which invalidates the condition for the subsequent state transition; therefore, ES_1 is deleted. Simultaneously, the arrival of C_3 results in the creation of ES_3 and its transition to the CH_4 state.

5) At t=8, for the same reason as before, ES_2 is deleted. The arrival of S_2 results in the creation of ES_{31} and its transition to the Smoke state. Now, the first matching event profile (comprising C_3 and S_2) occurs.

6) Similarly, at t=9, S_3 arrives, and ES_3 is created and transitions to the Smoke state. The second matching event profile (comprising C_3 and S_3) occurs.

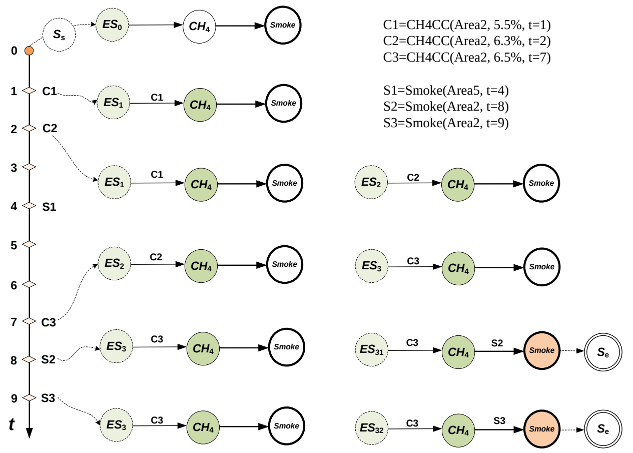

Fig. 13 Example of automaton processing for situation detection

The matching rules described by the event processing language are used to detect complex events of interest, such as an alarm event, and the Pub/Sub module is used to send that event to the service system engine. Based on the received alarm event, an appropriate alarm disposal process is triggered. Each alarm is associated with a set of disposal processes, including several disposal procedures. An E-ECA rule is used to control the disposal processes for different levels of alarms. An overview of a hierarchical coal mine safety alarming service is illustrated in Fig. 14.

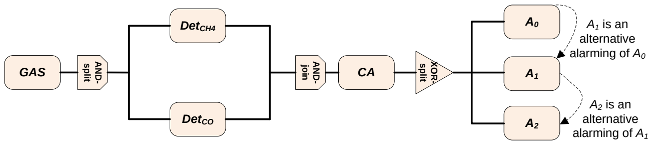
GAS
AND-split
Det_CH4
Det_CO
AND-join
CA
XOR-split
A0
A1
A2
A1 is an alternative alarming of A0
A2 is an alternative alarming of A1

Fig. 14 Hierarchical coal mine safety alarming service

As shown in Fig. 14, this alarm disposal IoT services coordination process contains three control flow modes: AND-split, AND-join, and XOR-split. Specifically, when an alarm disposal process is initiated, an AND-split flow control operator exists, which means the Det_{CH4} (Det_{CH4}, detection of an event related to the methane concentration) service and the Det_{CO} (Det_{CO}, detection of an event related to the carbon monoxide concentration) service run simultaneously. Once this safety situation event pattern is successfully detected for the methane concentration or the carbon monoxide concentration, the Classified Alarming (CA) service is invoked and returns the classified safety result. Then, the alarm disposal process based on this classified safety result invokes the services set $\{CA, A_0, A_1, A_2\}$, and an XOR-split flow control operator exists, which allows selection of a corresponding action to complete the safety alarm disposal task.

VI. Performance Measurement and Analysis

A. *Experimental Setup*

The situation-aware IoT services coordination platform was installed on a PC server (2.5 GHz Intel Xeon E5420 with 10 G of RAM). A functional coal mine was selected as the basic experimental environment, in which 30 Mica2 sensor motes were deployed in a tunnel wall approximately 10 meters wide and 5 meters high, and the cluster-tree network mode based ZigBee wireless sensor network was chosen and deployed. In addition to the controller area network bus technology and the Web of things technology, all types of parameters were collected and transmitted to the remote monitoring center for analysis to provide decision-making information for the clients. Furthermore, 10 other similar coal mines were used in experimental environments to evaluate the performance of the situation-aware IoT services coordination platform.

B. *Accuracy for Situational Event Detection*

The experiment was performed to demonstrate situational event detection performance. There are four kinds of events are involved. Here, Esper [32] is an open source software that aims to address the requirements of applications that analyze and react to events. Therefore, Esper is used for performance comparisons. Fig. 15 is the performances comparison between our automata based detection approach and Esper.

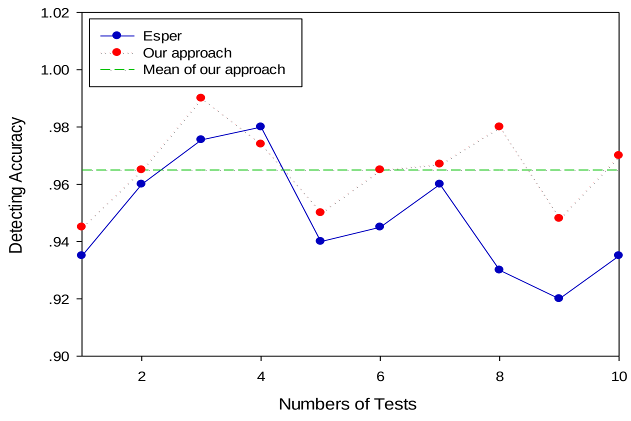

Fig. 15 Accuracy comparison

The experiment was repeated 10 times, and Fig. 15 shows that the accuracy of situational event detection by our proposed situational event pattern detection approach was higher than that of the Esper software. This occurred because of our automata based detection approach provides powerful expression for complex situational event pattern. However, the Esper outperforms our detection approach when the events pattern is simple.

C. *Scalability of the Data Distribution Model*

To study the scalability of the grid-quorum-based approach, the event arrival rate was fixed to 1 event every 10 seconds, the proportion of subscribers was set to 10%, the number of users for each broker was set to 10, and numbers of broker nodes between 32 and 2048 were considered. Fig. 16 shows the average search delays for different numbers of broker nodes.

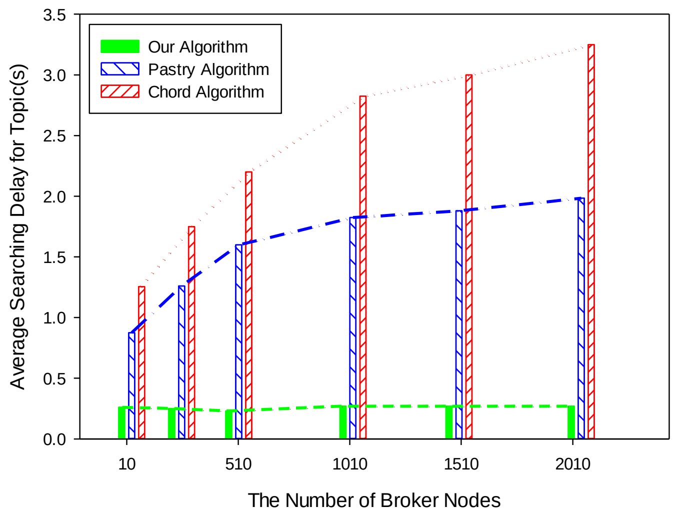

Fig. 16 Average search delays for different broker nodes

It is apparent from Fig. 16 that when the number of nodes in the overlay network reaches 2048, the average search delay for the grid-quorum-based method is 81.7% lower than that of the Pastry algorithm and 88.8% lower than that of the Chord algorithm. When one broker receives the topic message from the local host, the topic search delay is close to zero, and once the topic message forwards to other neighbor broker, and which can cause some search delay. However, the topics in each broker can route from one broker to other broker within two hops in Grid Quorum network. So, the average search delay is not significantly affected by the number of broker nodes. As expected, the average search delay for the grid-quorum-based approach is not significantly affected by the number of broker nodes. These results indicate that the grid-quorum-based approach demonstrates good scalability in terms of the number of brokers.

In the third experiment, to determine the relation between the event arrival rate and the average search delay, the event arrival rate was varied from 1 to 128 per second. In this experiment, the number of broker nodes was 256. For a given number of broker nodes, the total network packet traffic will increase as the event arrival rate rises. This usually leads to an increase in the average search delay. The results are shown in Fig. 17.

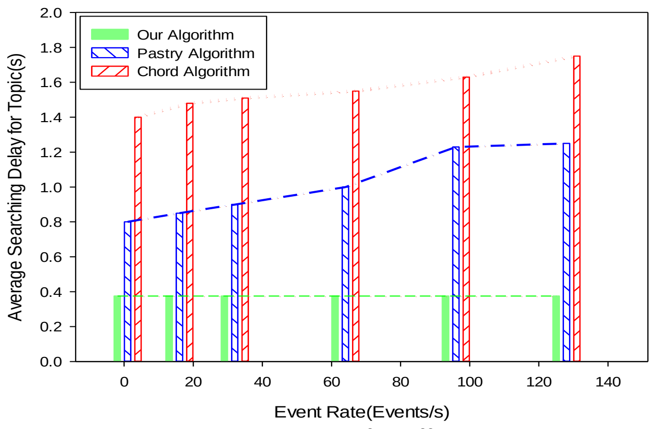

Fig. 17 Average search delay for different event rates

As shown in Fig. 17, the average search delay for the Grid Quorum-based approach is 81.7% lower than that of Pastry and 88.8% lower than that of Chord. To a given number of brokers, the event packets traffic will grow up with the event rate is increasing in both Pastry and Chord, also the search delay is increasing. However, the topics in brokers for Grid Quorum network route from one broker to another broker within two hops, hence, the average search delay is stable and not significantly affected by the number of broker nodes increasing. The results indicate that the Grid Quorum-based approach has a better scalability among the number of brokers, the Pastry and Chord may be unsuitable due to their high lookup costs.

D. *Response Time for Service Coordination*

The alarming disposal process of the coal mine safety was used to test the performances of Event Driven SOA-based (EDSOA), framework-based, such as ReCEPtor [33], and conventional request-response-based IoT services coordination approaches. The BPEL language was used to orchestrate a request-response-based coal mine safety alarm disposal service process. Fig. 18 shows the corresponding response time for those three different approaches.

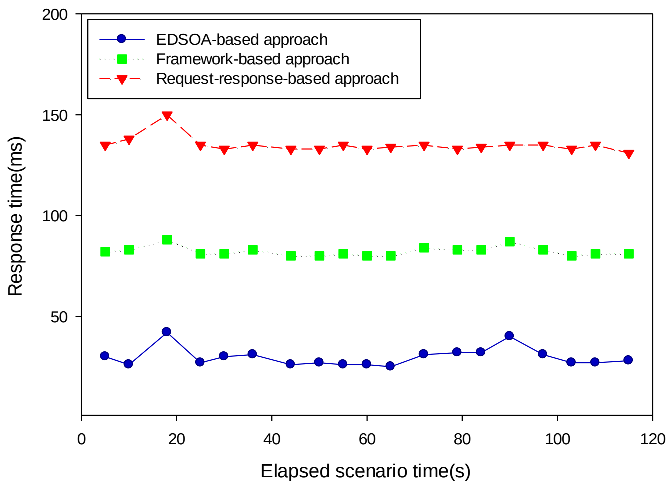

Fig. 18 Comparison of response time

As shown in Fig. 18, in the case of the conventional request-response services coordination approach, the maximum response time was 50 ms, and the average response time was 134 ms. In the case of the framework based services coordination approach, the maximum response time was 90 ms, and the average response time was 78 ms. However, for the event driven SOA-based services coordination approach, the maximum response time was 42 ms, and the average response time was 30 ms. Thus, the processing speed of event-driven SOA-based services coordination was less than those of the conventional request-response and framework based services coordination approaches. This occurred because a central control point handles the "request-response" BPEL process, which may cause all messages to propagate along a longer route and wait in a central queue for processing. In contrast, event-driven SOA-based services coordination uses the event's asynchronous communication mechanism to support the concurrent coordinated execution of multiple services. In addition, the event driven SOA-based services coordination approach has several advantages over the framework-oriented approach for developing dynamically adapting business processes, which provides general models that need to be instantiated and customized before they are implemented. Because those models operate at a higher level of abstraction than frameworks, they impose fewer initial constraints upon the system being developed. Therefore, the event-driven SOA-based services coordination approach is more suitable for IoT application scenarios that are highly delay sensitive and demand a strict real-time response.

VII. Conclusions and Future Work

This paper presented a situation-aware IoT service coordination platform based on the event-driven SOA paradigm. We proposed the SEDL language, an automaton-based situational event detection algorithm, and a situational event-driven service coordination behavior model, and a Publish/Subscribe-based unified message space is proposed to facilitate the asynchronous communication and on-demand distribution of sensory data in a large-scale distributed IoT environment. We deployed a prototype to test the system's validity. Experiments were performed and analyzed. The results demonstrated that the situation-aware IoT service coordination platform functioned well, as expected. Nevertheless, several issues yet remain to be addressed. First, as an expansion of existing lightweight service mashup middleware, visualization technology, such as 3D technology, can further improve the visibility of sensor objects. Second, it will be essential to optimize the real-time data distribution service and the data congestion scheduling strategy with different QoS for large-scale IoT application deployment. Such research efforts are currently in progress in our laboratory.

References

[1] L. Atzori, A. Iera, and G. Morabito, ''The internet of things: A survey,'' *Comput. Netw*., vol. 54, no. 15, pp. 2787–2805, Oct. 2010.
[2] J. Zheng, D. Simplot-Ryl, C. Bisdikian, and H. T. Mouftah, ''The internet of things,'' *IEEE Commun. Mag*., vol. 49, no. 11, pp. 30–31, Nov. 2011.
[3] Wang M M, Cao J N, Li J, ''Middleware for wireless sensor networks: A survey," J Comput Sci Technol, vol.23, no.3, pp.305-326, 2008.
[4] Bandyopadhyay S, Sengupta M, Maiti S, ''A survey of middleware for Internet of Things: Recent Trends in Wireless and Mobile Networks," *in*

Proc. The Third International Conference on Computer Networks & Communications, pp.288-296, 2011.

[5] J. Leu, C. Chen, and K. Hsu, "Improving heterogeneous SOA-based IoT message stability by shortest processingtime scheduling," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2013.30.

[6] D. Guinard, V. Trifa, S. Karnouskos, P. Spiess, and D. Savio, "Interacting with the SOA-Based Internet of Things: Discovery, Query, Selection, and On-Demand Provisioning of Web Services," *IEEE Transactions on Services Computing*, vol. 3, no. 3, pp. 223-235, July-September 2010.

[7] Dave Locke. Mq telemetry transport (mqtt) v3. 1 protocol specification. IBM developerWorks Technical Library], available at http://www.ibm.com/developerworks/webservices/library/ws-mqtt/index.html, 2010.

[8] P. Saint-Andre, "Extensible messaging and presence protocol (XMPP): Core," Internet Engineering Task Force (IETF), 2011.

[9] Al-Fuqaha A. , Khreishah A. , Guizani M. , Rayes A. "Toward better horizontal integration among IoT services," *IEEE Communications Magazine*, vol.53 , no.9, pp.72 – 79, 2015.

[10] OASIS Advanced Message Queuing Protocol (AMQP) Version 1.0, 29 October 2012, available from http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf

[11] Object Management Group (OMG), "Data Distribution Service for Real-time Systems," http://www.omg.org/spec/DDSII.2.

[12] Medjahed, B., "Dissemination Protocols for Event-Based Service-Oriented Architectures," *IEEE Transactions on Services Computing*, Vol. 1, No. 3, July-Sept. 2008, pp.155-168.

[13] Overbeek S, Klievink B, Janssen M, ''A flexible, event-driven, service-oriented architecture for orchestrating service delivery," *IEEE Intell Syst*, pp.31-41, 2009.

[14] B.Babcock, et al, ''Models and Issues in Data Stream Systems," *in Proc. ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, New York, pp.1-16, 2002.

[15] A.Arasu, S.Babu, ''CQL: A language for continuous queries over streams and relations," *in Proc.9th Workshop on Database Programming Languages*, DBPL 2, pp. 1-19, 2003.

[16] OASIS, ''Web Services Business Process Execution Language (version 2.0) ,"http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, April 2007.

[17] N. Kavantzas, et al, ''Web Services Choreography Description Language Version 1.0," W3C Candidate Recommendation, http://www.w3.org/TR/ws-cdl-10/, Nov. 2005.

[18] Laliwala, Z.; Khosla, R.; Majumdar, P.; Chaudhary, S. "Semantic and Rules Based Event-Driven Dynamic Web Services Composition for Automation of Business Processes," *in Proc. IEEE Services Computing Workshops*, pp. 175 – 182, 2006.

[19] Bae J, Bae H, Kang SH, Kim Y., "Automatic control of workflow processes using ECA rules," *IEEE Trans. on Knowledge and Data Engineering*, vol.16, no.8, pp.1010-1023, 2010.

[20] Majithia,S., Walker, D.W., & Gray, W.A, "Automated Web service composition using semantic Web technologies," *in Proc. of the International Conference on Autonomic Computing*, pp.306-307, 17-18 May 2004.

[21] IoT.est: IoT Environment for Service Creation and Testing (http://ict-iotest.eu/iotest/).

[22] ThingWorx. Available at: http://www.thingworx.com/platform/

[23] Palattella, M.R.; Accettura, N.; Vilajosana, X.; Watteyne, T.; Grieco, L.A.; Boggia, G.; Dohler, M. "Standardized Protocol Stack for the Internet of Things," *IEEE Communications Surveys & Tutorials*, vol.15, no.3, pp. 1389-1406, 2013.

[24] Sheng, Z.; Wang, H.; Yin, C.; Hu, X.; Yang, S.; Leung, V.C.M. "Lightweight Management of Resource-Constrained Sensor Devices in Internet of Things," *IEEE Internet of Things Journal*, vol.2, no.5, pp.402-411, 2015.

[25] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP), IETF RFC 7252, June 2014; https://tools.ietf.org/html/rfc7252.

[26] Raza, S.; Shafagh, H.; Hewage, K.; Hummen, R.; Voigt, T. "Lithe: Lightweight Secure CoAP for the Internet of Things," *IEEE Sensors Journal*, vol.13, no.10, pp.3711-3720, 2010.

[27] Bormann, C.; Castellani, A.P.; Shelby, Z. "CoAP: An Application Protocol for Billions of Tiny Internet Nodes," *IEEE Internet Computing.*, vol.16, no.2, pp. 62 – 67, 2012.

[28] C. Bo, Q. Xiuquan, W. Budan, W. Xiaokun, S. Ruisheng, C. Junliang, "RESTful Web Service Mashup based Coal Mine Safety Monitoring and Control Automation with Wireless Sensor Network," *in Proc. IEEE 19th International Conference on Web Services (ICWS)*, pp. 620-622, 2012.

[29] A. Morzenti, D. Mandrioli, and C. Ghezzi, "A model parametric real-time logic, "*ACM Trans. Program. Lang. Syst.*, vo.14, no.4, pp.521-573, 1992.

[30] S. Cheung, M. Ammar, and A. Ahamad, "The Grid Protocol: A High Performance Scheme for Maintaining Replicated Data", *in Proc. ICDE*, pp. 438-445, 1990.

[31] Y. Sun, X. Qiao; B. Cheng; J.L. Chen, "A Low-Delay, Lightweight Publish/Subscribe Architecture for Delay-Sensitive IoT Services," in Proc of IEEE ICWS, pp.179 - 186, 2013.

[32] Esper product. Open source project. Available at: http://www.espertech.com.

[33] M. Wei, I. Ari, J. Li, and M. Dekhil, ''ReCEPtor: Sensing Complex Events in Data Streams for Service-Oriented Architectures," Accessible via http://www.hpl.hp.com/ techreports/2007/HPL-2007-176.html.

[34] Gabriel Hermosillo, Lionel Seinturier, Laurence Duchien,"Using Complex Event Processing for Dynamic Business Process Adaptation," *in Proc. of IEEE SCC*, pp.466-473, 2010

Bo Cheng is a professor at the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He received his Ph.D. degree in computer science at University of Electronics Science and Technology of China in 2006. His research interests include Service Computing, Internet of Things and multimedia communications.

Da Zhu is a Ph.D. candidate in the State Key Laboratory of Networking and Switching, Beijing University of Posts and Telecommunications, Beijing, China. His current research interests include web service composition, communication web service, and Internet of Things.

Shuai Zhao is a postdoctoral fellow in State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His current research interests include Internet of Things and service computing.

Junliang Chen is a professor of Beijing University of Posts and Telecommunications. His research interests are in the area of service creation technology. Prof. Chen was elected as a member of the Chinese Academy of Science in 1991, and a member Chinese Academy of Engineering in 1994.