Event-driven SOA for IoT Services

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

Abstract—Although lots of IoT (Internet of Things) applications have been developed, the systematic method to construct IoT services is still obscure. In this paper, an Event-driven Service-oriented Architecture (EDSOA) for IoT services is discussed, where distributed events act as a primary mechanism for each IoT service to share independent meaningful events, to express its requirements and capabilities, and to decouple itself from other services. Such distributed events however do not provide powerful expressiveness to describe business logic in SOA because business activities are not completely independent each other.

To fill the gap, we propose an information-centric session mechanism to describe service behavior working upon distributed events, called event session. This paper also discusses how to build an Event-driven SOA infrastructure, where we can use resource information to create IoT services, use independent and shared events to run the IoT services, and use event session to coordinate the IoT services. Some applications and experiments are given to show concept proof for such event-driven SOA.

Keywords—IoT Services; Service Coordination; Process Theory; Publish/Subscribe; Event-driven SOA

I. INTRODUCTION

For IoT services, the primary step is to introduce physical entities in the physical world into the information world. The digital entities corresponding to physical entities are called resources. A resource provides functionalities about accessing to the physical entity’s properties and actuating the physical entity. We construct the resource as a standard service, called bottom-layer service or resource service, for sharing, interaction, composition, and so on. A resource service is defined by its resource model and its lifecycle model. Its resource model describes what properties it has and what relations it has with other resources. Its lifecycle model describes what way it runs and lifetime state transitions. The interface of resource service is about its properties accessing and state transitions in its lifecycle. Business services operate on the lifecycle of resource service to induce its state transitions. The service operation is carried out based on events. In order to achieve scalability, the resource service is duplicated in each local site according to its models. The soundness of accessing localized resource services is that the majority of distributed accessing is to read entities’ properties, and only weak data consistency is required. The weak data consistency assumption is that the distributed data copies of entities’ properties in different sites are not required to be the same at each instant time, but are required to have very close mean value during a time slice with the same state transitions. Distributed resource pools located in different sites should be implemented as a fundamental utility in EDSOA (Event-driven Service-oriented Architecture).

Distributed events play a key role between business services and resource services, which drive IoT services to progress during running. The upper-layer IoT services, i.e., business services, use events to access to the resources’ properties or their actuation capabilities. Each IoT service can locally obtain events, react to their arrival, and publish its output as a distributed event for other IoT services to locally use. Furthermore, it is required that each event is independent, i.e., each IoT service being able to consume it without knowing its from and to. The events exist in a specific distributed event-based service environment which is a fundamental utility in EDSOA to support service running and real-time event distribution.

Because each resource service maybe has a local instantiation at each site, a business service (also called upper-layer service) is thus assumed to directly access to the resource’s states. A resource state is represented by a collection of atomic propositions. In the state, each proposition in the collection is required to be true. The collection can be represented by a proposition formula, i.e., the conjunction of all atomic propositions in the collection. The upper-layer service actuates a physical entity through publishing a distributed command event being received by resource service (or actuation agent service), where the event contains actuation parameters to stimulate the state transition of resources.

We hierarchically establish IoT services based on distributed events, i.e., resource services and business services with distributed events and resource propositions as their connections. Although events are independent in the service environment, each service has its own behavior and interacts with other services. The service behavior and interactions say how to exchange events among service activities and how to react to event arrivals. That is to say, there are dependency relationships among events in the service system. In order to fill the gap and keep consistency with event-driven methodology, we propose an information-centric session mechanism to describe service behavior and interactions working upon distributed events, called event session. The event session means using the content in events to correlate different service activities, through which the activity relationship in a service behavior can be defined. A utility, called Event Relation Synergizing Utility, is built to use the event session to coordinate service activities.

To the best of our knowledge, there are no existing works about EDSOA to thoroughly discuss how to keep both event independency in the distributed event-based service environment for sharing, and event dependency in one specific service system satisfying business requirements, and how to use IoT resources as a basis. The contributions of this paper are as follows:

• An EDSOA infrastructure (a platform to realize EDSOA) for IoT services is proposed, where a distributed resource-supporting utility is defined to pool resource properties and states, a specific distributed event-based service environment is given to support real-time event distribution with service running support, and an event relation synergizing utility is specified to coordinate services collaborations with decoupling. In the infrastructure, we establish hierarchical IoT services, i.e., resource services and business services with distributed events and resource proposition as their connections.

• In order to address the service coordination issue, we first formally define IoT services in process theory, and then we propose the information-centric session mechanism to describe service behavior and interaction working upon distributed events (called event session). An execution model for event-driven IoT services is then proposed to decouple services at behavioral layer.

In our EDSOA infrastructure, we can use resource information to create IoT services, use independent and shared events to run the IoT services, and use event session to coordinate the IoT services.

The remainder of the paper is structured as follows. Section 2 describes the preliminaries. Section 3 gives an EDSOA for IoT...
services. Section 4 gives some applications. Section 5 describes the related work. Finally, conclusions are drawn in Section 6.

II. PRELIMINARIES

We adopt the theory of Hybrid Process Algebra (HyPA) defined in [10] to describe IoT services, where IoT services’ discrete behavior can be directly embedded into the bigger hybrid system model. Because the work of [9] is more canonical, we adopt the notations in [9] to express the process theory in [10].

Term is defined over a set of variables and a signature which consists of constant and function symbols. An equational theory is a tuple \((\Sigma, E)\), where \(\Sigma\) is a signature and \(E\) is a set of equations of the form \(s = t\) \((s, t\) are terms). Process theories are special equational theories, and the objects being described by a process theory are referred to as processes. The primary functions are as follows (readers can refer to [9], [10] for more details):

- \(a_\_\) denotes action preface, where the process \(a\_x\) executes action \(a\) then proceeds as term \(x\);
- \(+_\_\) denotes alternative composition or choice, i.e., \(x + y\) behaving either as term \(x\) or as term \(y\), but not as both;
- \(-_\_\) denotes parallel composition, i.e., \([x|y]\) behaving either as term \(x\) or as term \(y\), or as both with arbitrary interleaving or merging;
- \([\varphi]\) denotes guard command, i.e., the process \([\varphi]\_x\) executing the term \(x\) if the proposition formula \(\varphi\) is true.

IoT services consist of two layers. For an upper-layer service (business service), the below features should be described:

- Accessing to resources. Resource states are represented by proposition formulae, and the proposition formulae are embedded in the upper-layer service as parameters. The upper-layer service uses the propositions to represent directly accessing to resources.
- Service session. When services depend on independent events to interact, it is hard to manage a series of service interactions as a transaction. Some event contents are correlated together to form a service session identifier.

**Definition 1 (Upper IoT Service)** An upper-layer IoT service \(\text{IoTs} := (\text{ACT}, \text{PR}, \text{SUB}, \text{PUB}, \psi)\) is a 5-tuple, where \(\text{ACT}\) is a set of events (or actions); \(\text{PR}\) is a set of atomic propositions including three parts: \(\text{PR}\_\_\) session proposition set, \(\text{PR}_\_\) resource proposition set, and \(\text{PR}\_\_\) inner proposition set; \(\text{SUB}\) is a set of subscription interfaces represented by \([\varphi]\_a\) where \(\varphi \subseteq \text{ACT}\); \(\text{PUB}\) is a set of publication interfaces represented by \([\varphi]\_a\); \(\psi\) describes the behavior of IoTs. \(\psi\) is defined as follows:

- get a parameterized process theory \((\text{HyPA} + \text{PR})(\text{ACT}, \gamma)\);
- the service behavior is represented by a process in \((\text{HyPA} + \text{PR})(\text{ACT}, \gamma)\), where the interfaces such as \([\varphi]\_a\) in \(\text{SUB}\) and \(\text{PUB}\) are included in the process.

The communication function \(\gamma\) is defined as follows (\(t\) event name (topic name), \(d\) event content):

- \(t!d\) (send data element \(d\) through name \(t\));
- \(t@d\) (receive data element \(d\) through name \(t\));
- \(t@d\) (send and receive data element \(d\) through name \(t\)).

In the definition, each event/action is guarded by a guarded command \((GC)\). The guard in a guard command is represented by a proposition formula \(\varphi\) consisting of multiple atomic propositions in which we allow for negative proposition only for convenience. \([\varphi]\_x\) means ‘if \(\varphi\) then \(x\)’, where \(x\) is an action. As to service session, some event contents are correlated into a session identifiers.

**Example**

- \(e = \text{command}(\text{resourceID} = \text{id}, \text{control} = \text{ON})\),
- \(P = \text{command}(\text{id}, \text{ON})\),
- \(P_1 = [\text{resourceID} = \text{id}; \text{exclusive} = \text{yes}]P\_1\),
- \(P_2 = [\text{resourceID} = \text{id}; \text{status} = \text{ready}]; \text{command}]x\),
- \(M = \parallel P_2\).

where \(e\) is an event with name \(\text{command}\) and requires the resource with identifier being \(\text{id}\) to do action \(\text{control} = \text{ON}\); the session proposition is \(\text{resourceID} = \text{id}\); \(P_1\) means that, when it is in “exclusive” status (inner state) and \(\text{resourceID} = \text{id}\) is true, it will be enabled to publish a control command \(e\) through name \(\text{command}\); \(P_2\) means that, when the resource is in “ready” status and \(\text{resourceID} = \text{id}\) is true, it will be enabled to receive a control command \(x\) through name \(\text{command}\). In the process \(M\), \(P_1\) and \(P_2\) are able to communicate only when they are enabled and their session propositions are the same, i.e., \(\text{resourceID} = \text{id}\).

III. EVENT-DRIVEN SOA FOR IOT SERVICES

Our EDSOA infrastructure consists of three parts illustrated in Fig. 1: Distributed resource pool, DEBS-based service environment (called unified message space), and Event relation synergizing utility. The DEBS-based (DEBS: Distributed Event-based Systems) service environment uses distributed events as the primary mechanism to define service interface, run IoT services, exchange interaction messages among services, and so on, where event is identified by topic name, and no interaction destination and source are concerned.

![Fig. 1. EDSOA Infrastructure for IoT Services](image)

The scalability supporting from DEBS-based service environment is based on the decoupling feature of Publish/Subscribe paradigm and policy-inducing isolation method. Sensors, physical entities, and IoT services are often distributed such that some distributed resource utility is needed for different digital entities to read real-time data. Distributed resource pool is thus constructed to support the real-time data accessing by multiple distributed IoT services. In this utility, there are multiple local resource pools based on the weak data consistency assumption, which construct the most recent resource states by subscribing sensed events to the DEBS-based service environment. In the DEBS-based service environment, events are independent while, in a IoT service system, one event may depend on other events. We establish an event relation synergizing utility to help designers analyze the event relationship among multiple IoT services and coordinate IoT services to ensure the event relationship when they run concurrently.

A. Using Resource Information to Create IoT Services

In an event-based resource pool, local resource services are instantiated and managed. A resource service consists of a resource model and a lifecycle model, and provides a service interface for business services to access to its properties in its resource model and to operate on its lifecycle states. A resource model expresses resources, entities, and their relationship, where resources represent sensors and their monitored properties, entities represent physical objects in a real world, the relationship represents the binding between a resource and an entity, i.e., an object having a specific attribute. In addition, the resource model also describes the relationship between one resource and other resources, and the relationship among entities. For instance, Fig. 2a shows the wind speed sensor model with observation accuracy description; Fig. 2b shows an entity model which includes the properties of entity, the relationships between entities.
The two services may be a part of business process and may have a temporal relationship. From Fig. 4, we know the business services are information-centric, i.e., reading a resource’s properties according to its information model, operating on its lifecycle states and operation being based on informative events. It is worthwhile to point out that a resource service can be used to represent business objects such as business artifacts [25], [26], and business services are still information-centric.

![Figure 4: The Relationship between Business Service and Resource Service](image)

Although we use the entity model in services, we do not distinguish between resource and entity, denoting them both by resources. When a resource service is locally instantiated, its states or properties are updated through subscribing named events. Each resource service instance corresponds to a physical resource or entity. There are multiple resource service instances for one resource model. In two different event-based resource pools, there possibly exist two copies for one resource service. We assume that, with high possibility, the two copies have the same state, even if, at a specific instant, the two copies have different states. A resource pool often uses a real-time database or real-time tuple space to store resource states of resource service instances.

### B. Using Independent Events to Run IoT services

Business services use events to communicate with resource services, and interact with each other though resources or business artifacts (a special kind of resource). The unified message space is a utility to support service interactions based on events. In the unified message space, the event consumers (services) can express their interests to the DEBS-based service environment by subscription. The environment delivers the named event to the consumers if their subscriptions match against the topic of event, where topic is hierarchical and each topic is unique and each topic may correspond to a set of events with the same topic. The relationship among multiple topic names is often represented by a name tree. Fig. 5 illustrates an example. In Fig. 5, JinFang is a company who provides heating provision service for residents in the winter. It has a heating provision system which produces and consumes events named Telemetry, Telesignalling, Remote Control, and so on. The Telemetry name has some child topic names such as Water Temperature, Water Pressure, and so on. Each topic name in the name tree in Fig. 5 has its own attributes $Attr$, but an access policy $AP$ is made for one sub-tree such as Telemetry. According to event names and consumers’ subscriptions, the unified message space can carry out matching and routing. According to event names and access policies, the unified message space can control event flow behavior such as delivery speed, dissemination scope, and so on. In a unified message space, there are multiple such name trees, and events are independent from each other for any consumer services to consume them. The relationship between topic name and event is defined in Definition 2.

**Definition 2 (Topic Name and Action (Event))** The topic name in IoT is constructed as

\[
name := \text{topic_subject(topic_verbing)}
\]
The action (event) in IoT is defined as topic_subject(topic_verbing●), where ● is an option item which may include content in the event, direction, and others.

In the above definition, option item is not concerned in the unified message space, but in the service system, it cannot be simply neglected and direction may be publication (out) or notification (in) from a service’s view. We give an example in Fig. 6 to illustrate the event concept. In Fig. 6, the “wsnt:Topic” tag indicates a topic name, called BoilerAlarmTopic; the “wsnt:Message” tag indicates an event content which consists of a series of name/value pairs such as Level2 and Value82.3. When a topic name is given, its elements in the content, i.e., tags, are determined.

```xml
 xmlns:wsnt="http://docs.oasis-open.org/wsnt/2-0">
  <env:Body>
    <wsnt:Notify xmlns:wsnt="http://docs.oasis-open.org/wsnt/2-0">
      <wsnt:Topic>BoilerAlarmTopic</wsnt:Topic>
      <wsnt:Message>
        <TemperatureAlarm>
          <Location>7522gas boiler</Location>
          <Measure Parameter="SWT">Measure Parameter</Measure>
          <Information>below low limit</Information>
          <Value>82.3</Value>
          <Level>2</Level>
          <TemperatureAlarm>
        </TemperatureAlarm>
      </wsnt:Message>
    </wsnt:Notify>
  </env:Body>
</env:Envelope>
```

Fig. 6. An Event Example

From the above example, we know that events are primary entities in services interactions, and service runtime is driven by the information (properties, states, and state transitions). That is to say, a service can complete its execution when it receives an event, i.e., changed information in the resources, and retrieves resource properties and states from the local resource pool, i.e., unchanged information in the resources. In practice, physical entities, sensors, and IoT applications often have their own geographical boundaries and natural ownership. The DEBS in the unified message space connects together distributed services and entities lying in different autonomy domains, where DEBS is wide-area and Internet-scale through grouping and isolating on overlay structure. As illustrated in Fig. 7, the unified message space builds on DEBS a service interface layer through which services use Publish/Subscribe operations to define their own interfaces, and use the Register operation to claim themselves. The unified message space also provides a service container, where the Publish/Subscribe service interface is used as interface contract to support programming based on contracts.

The unified message space is often constructed based on a Publish/Subscribe communication paradigm. Compared with common Publish/Subscribe systems [1,2,3,4,5,6], the unified message space has distinguished features for IoT services as follows:

- Different event consumers may have different receiving rate requirements and the unified message space can optimize space performance by appropriately splitting event flows.
- Non-re-transmission. According to the constraints of real-time and multicast communication, the unified message space does not adopt the re-transmission method to realize reliability, and the continuous flow assures that consumers will receive parts of all messages even if some packets are lost.
- Differentiated reliability. For some messages with high reliability requirements, the reliability method is realized by applications and the unified message space can act as an assistant to help applications to achieve reliability goals. That is to say, the unified space can cache some packets, and directly respond to the applications’ request with cached packets [7].

Fig. 7. Unified Message Space: DEBS-based Service Environment

Some above features are considered by the publish/subscribe-based WAMS-DD infrastructure proposed by the GridStat project [8] which is designed for a smart grid.

The service container in unified message space creates service instances and arranges each activity in the instances, i.e., triggering Sub (subscription) activities according to event arrivals and putting events from Pub (publication) activities into the distributed event-based system. In addition, it also records historical data, provides a summary of running service instances, monitors the consistency of services.

The event-driven method makes IoT services decouple from each other at the time, space, and control dimensions. Hierarchical topic name structure makes it possible for the service environment to handle more metadata. In order to further satisfy the scalability requirements, the DEBS-based service environment (unified message space) behaves as a distributed network and new nodes can join it on demand. Furthermore, policies are used to describe how to confine the dissemination scope of events while the event independency is kept. The policies are used during generating routing table, and the forwarding procedure does not directly involve the policies with keeping real-time delivery.

C. Using Event Session to Coordinate IoT Services

In the above two sections, we discuss that business services are designed based on resources, i.e., their properties and lifecycle states, and are executed based on events from resources, i.e., properties events and lifecycle events. However, in a service system (business process), there are multiple resources involved,
where different resource lifecycle should be coordinated, and different service operations on one resource lifecycle should also be coordinated. For example, there is a Switcher resource and a Device resource such as transformer in a power station, illustrated in Fig. 8, where the Switcher should be transited into the OFF state in its lifecycle before the Device is transited into the Repair state in its lifecycle. The Control Service operates on the Switcher’s lifecycle to induce that the Switcher moves from the ON state into the OFF state. The Repair Service operates on the Device’s lifecycle to induce that the Device moves from the Normal state into the Repair state, and the device is checked and maintained by servicemen. For safety, the Control Service should be executed before the Repair Service is executed, i.e., coordinating two services.

We introduce an event session concept to assume the coordination task because the service operations are executed based on events which finally induce the state transition of resources’ lifecycles. That is to say, we use event sessions to ensure event causality and event confliction among a service system (business process).

![Fig. 8. Lifecycles & Services Coordination](image)

According to the event content, event session is defined by a series of name/value pairs in the content (a subset of all name/value pairs in the content). Two events belong to the same session if they have the same name/value pairs as session identifiers. An action constrained by a session proposition in the service behavior can react to a notified event if the name/value pairs in the event content include the name/value pairs in the session proposition, i.e., each included pair in the session proposition having the same name and value as the name/value pair in the event content. For example, the action $\{\text{Level} = 2; \}$ on the BoilerAlarmTopic content can receive the event in Fig. 8.

In the unified message space, events from different services are independent while in the system consist of these services, each service has its behavior, and there are interactions among services such that these events are not independent in the system. An event is identified by its topic name in the unified message space, and one topic name may correspond to multiple events. In order to describe the service behavior and interaction in a service system, not only the event topic is concerned, but also the content contained in the event is used.

When a topic name is used to identify a kind of events and used as a subscription to match events in the unified message space, the session identifier is used to identify a specific event, i.e., different instances for one topic name with different values in name/value pairs. A service uses a topic name to describe what events (changes on properties or lifecycle of a resource) it needs and produces, and uses session as a fine-grained mechanism to describe what event instance and event content it needs and produces. According to the session mechanism, the event causality can be expressed. That is to say, a subscription action’s session proposition describes the direct cause event of the action because the notified event should satisfy the session proposition before the action is enabled. Compared with the channel mechanism in request/reply-based service interaction, the session mechanism for Publish/Subscribe-based service interaction is information-centric with characteristics as follows:

- Using topic name as subscription is efficient for simply matching the subscription against events in the unified message space. The NDN network architecture also uses data name as a basic mechanism to describe a customer’s interest and data in the network, which provides efficiency for large-scale data.
- The session mechanism is used to identify event content and specific event instances, express event causality and confliction, and form service transactions crossing multiple services and multiple interaction rounds, which keeps with the event-driven methodology, i.e., reacting to event arrivals with no from or to being concerned.

In our service description, the action in Pub/Sub interface is defined by topic name (event name) which describes what event type is needed and produced, atomic session propositions are used to describe data content and event relation, and $\{\varphi|a\}$ is then defined by event name (topic), event content and event relation such that the Pub/Sub interface $\{\varphi|a\}$ can be viewed as a meaningful independent unit. That is to say, if the event name (topic), event content and relation in $\{\varphi|a\}$ is satisfied, the functionality represented by $\{\varphi|a\}$ is provided. Event Relation Synergizing Utility is established to fill the gap between the unified message space with all events being completely independent and the service system with events being dependent, and to keep service decoupling in behavior.

The service behavior and interactions say how to exchange events and how to react to event arrivals. That is to say, there are dependency relationships among events in the service system. The process represents service behavior and its structure indicates event relation. A flow event structure views a concurrent system as a set of events [24]. The “flow” term represents “possible immediate causes”. We will use the flow event structure to represent the interactions among services such that event causality and confliction are gotten to encode services’ session proposition. We give an example in Fig. 9 to illustrate an interaction among multiple services with multiple rounds.

![Fig. 9. A remote control service interaction](image)

In the example, we use a collaboration diagram to graphically illustrate interactions among multiple services in a service system. The Fig. 9 presents a collaboration diagram [27] for a remote control service interaction in typical IoT systems. The diagram contains three services: Prime Dispatcher, Slave Dispatcher, and Actuator Agent. In this example, the prime dispatcher sends a session negotiation event for remote control. The event is received by the slave dispatcher and the actuator agent. The actuator agent calculates the session identifier and sends the session response event to the prime dispatcher and slave dispatcher. The prime dispatcher then sends a control command event and the slave dispatcher sends a control confirm event to confirm the correctness of the control command. This dual dispatcher mechanism is used to assure the critical devices are rightly operated. When the control events from the prime and slave dispatcher are received, the actuator agent prepares the control context and sends the control ACK event to the prime dispatcher to indicate the preparation is ok. Finally, the prime dispatcher sends...
the command execution event after receiving the ACK event.

![Diagram](image)

**Fig. 10.** An event dependency relation in the remote control service interaction

Fig. 10 illustrates the event causality in Fig. 9, which can be described as an event flow structure.

The service behavior of Prime Dispatcher is as follows:

- PrimeDispatcher \( t_1, B_1 \) \( \{ \text{random1}=\text{random2}; \text{random2}, \text{A1}\} \) \( \text{sid}, \text{random2} \).
- \( \text{SID}=\text{sid}, \text{random2} \).
- \( \text{sid}, \text{random2}, \text{rid} \).
- \( \text{rid} > \text{limit1} \).
- \( \text{t2}(\text{SID}, \text{rid}, \text{down}) \).
- \( \text{SID}=\text{sid}, \text{random2} > \text{limit}, \text{t2}(\text{SID}, \text{rid}) \).
- \( \text{PrimeDispatcher} \)

where \( t_1, B_1 = \text{SID Negotiation}, t_2, A_1 = \text{SID Response}, t_3 = \text{Control Command}, t_4 = \text{Control ACK}, t_5 = \text{Command Execution}; \text{rid} > \text{limit} \) means that the value of resource with identifier \( \text{rid} \) must be greater than the threshold \( \text{limit} \) in order to enable the action \( t_4(\text{SID}, \text{rid}) \).

Random1 and random2 are initializations for session negotiation. From the example, we know each action in Pub/Sub interface is described by topic name, and Pub/Sub interface \( [\varphi] \) is described by topic name, session propositions, resource propositions and other propositions.

At the behavioral layer, we hope that the events from one IoT service system are independent for the other service systems while, in the specific IoT service system, there often exists dependency relationship among these events. For example, in a system, a service is in charge of actuating a Switcher and another service is in charge of dispatching workers to repair some devices in a power station. Before the repair event takes place, the actuating off event must have happened. But in a data mining system, these two events are not required to keep one event happen before the other. Furthermore, each service operation is carried out on shared resources such that we hope each service operation is independent and replaceable. Therefore, we adopt the information-centric principle to obtain an event-driven execution model with each service operation as a meaningful and independent unit such that each Pub/Sub interface of a service can be executed if its cause events occur and conflict events do not occur, i.e., behavioral decoupling with one service system being independent from other service systems.

![Diagram](image)

**Fig. 11.** Service Execution Model

In Event Relation Synergizing Utility, the behavioral decoupling of services is exploited by a service execution model which is an abstract about how to run the event-driven service such that we are not lost in service execution details and can analyze event relations. In the execution model illustrated in Fig. 11, each service has a business logic layer which is translated from a flow event structure, and each Pub/Sub interface in the service process is translated into an execution unit. Multiple units for the service share the same business logic layer. That is to say, for one IoT service system, we first define multiple service processes, and then use a flow event structure to define the event dependency relationship among them as their interactions. The latter (flow event structure) is translated into a business logic for each service. The whole system is a composition of multiple service execution units and business logics, where the service execution units are managed by the service container and the business logics are managed by Event Relation Synergizing Utility. Whether the dependency relationship among events in the whole system is assured can be checked by model checkers. If the checking results are positive, the business logics of the IoT service system makes that events in one system are independent for other systems. In this utility, verifying methods and tools are provided as well as generation of service execution models and running the business logics.

IV. APPLICATIONS & EXPERIMENTS

An EDSOA-based district heating service system is illustrated in Fig. 12, where heat is generated by a power plant or boiler room, and sent to residential homes and commercial buildings in the district. In Beijing, China, heating provision is assumed by many traditional coal-fired and gas-fired boilers. In order to reduce carbon emission and improve air quality in Beijing, we should establish a new efficient district heating system which operates at the highest levels of efficiency. In the new district heating system, there are multiple substation systems. In a sub-station, we deploy many meters which produce real-time metered data collected by PLC (Programmable Logic Controller), or receive actuation instruction from PLC through RS-485 communication interface. The Monitor Agent Service communicates with PLC through the standard ModBus protocol.

The entire system mainly consists of three parts: multiple substation service systems, one headquarter service system, and one government heating management system, where they are connected with distributed event-based service environments, and there is a local resource pool at each site as well as a local event relation synergizing utility. The substation service system is composed of Monitor Agent Service, Actuator Agent Service, and resource state maintain service, called Constructing Resource Service, where the monitor agent service collects sensed raw data at the local site, generates alarm events from raw data according to the resource model and resource state in the resource pool, and then publishes the observed data and alarm events into the distributed service environment for other services to consume; the actuator agent service receives actuation events from the distributed service environment, and creates a session with some upper remote control service to safely actuate the local devices; and the constructing resource service subscribes the observed data and alarm events to the distributed service environment and maintains resources’ states in the resource pool. The resources’ states in the pool are directly accessed by other services, and considered to be standard services. The headquarter system also includes these services similar to the substation system as well as HMI Service, Warning Service, and other business services. In the figure, the DEBS-based unified message space utilizes the Publish/Subscribe mechanism to disseminate events, and the event broker acts as a critical component of unified message space to support the event publication, subscription, notification and routing functions. The headquarter service system uses the distributed events to interact with the government heating management information system which subscribes emergency events and statistical data. The different substation systems and headquarter system are all connected through event brokers of unified message space. In general, only one event broker is used in a substation system to process events among its own different sub-services. At the same time, this broker also deliver events to the headquarter system. Since different service systems interact with each other in an event-driven pattern, the system coupling degree is greatly reduced.

Because the DEBS-based service environment is global and
determined, and plays a key role in our EDSOA infrastructure, we make experiments in event processing in the infrastructure. In order to evaluate the event processing performance of our infrastructure, average event processing speed is used as the metric. The average processing speed refers to the number of events getting/putting events in the DEBS-based service environment. The average event processing speed with time axis is shown in Fig. 13a. We did the experiment many times with different event consumers or producers. In each experiment we produce more than 50,000 events. It is observed from Fig. 13a that with the increasing time from 60s : 00s to 5 min : 00s, the average processing speed of our service environment remained about 420 hit/s ∼ 490 hit/s. In the planned system scale, the event processing speed can meet the real-time demands. In Fig. 13b, the results were non-optimal, i.e., the jitter being less than 8.06%.

Because our DEBS-based service environment is distributed where different services in different sites may get the same events at different instant time. We choose 30 sites (event brokers) to form the distributed service environment and produce 100,000 events for each experiment. Then, the delay between different services to obtain the same events is illustrated in Fig. 14a, where the total delay for 100,000 events averaged for different services was 600 ms ∼ 1800 ms, i.e., 0.006 ms ∼ 0.018 ms for one event. "4s/4t" means that there are 4 event producers and 4 event consumers, and so on. In Fig. 14b, the delay stability with time increasing is illustrated, where average difference of delivery events in different services was about 460 events/s ∼ 1900 events/s for about 25,000 events/s throughput in the distributed service environment, i.e., the jitter being less than 8.06%.

V. RELATED WORK

SOA [15] is used to break application silos, and to get the functionality reuse and interoperability. In SOA, the entire application is broken down into multiple independent services described by the standard interface specifications. EDA (Event-driven Architecture) [14] is a loosely-coupled application coordination pattern, which specifies how to produce, detect, consume and react to events. In EDA, these independent services can be composed together by event flow in a dynamic way. Event-driven architecture can complement service-oriented architecture by using events to replace function invocation [17]. The work of [18] had explored the integration of SOA with EDA. The works of [19], [20], [21] discussed an event-driven and service-oriented architecture for services. The work of [22] proposed a simple extension to UML that supports both SOA and EDA. Our research team discussed how to use EDSOA in Internet of Things [23].

EDSOA is information-centric, and information-centric methodology is often adopted to design services and business processes such as business-artifacts-based in [25], [26]. Business-relevant objects are called business artifacts which are generated, evolved, and stored when they pass through a business. The business artifact is often defined by an information model and a lifecycle model. The information model defines data of the business objects used in their lifetime. The lifecycle model defines the possible state transition and timings of business objects, which can be used to describe tasks on these objects. The information model in the business artifacts is not suitable for IoT services because there are no resource concepts and binding concepts between resources and entities. Our resource service can be naturally used to represent business artifacts. They did not discuss how to use events to complete service coordination either.

For distributed events, the works in [13], [14] addressed the service coordination issue based on the distributed events. The formal model of service coordination in [13] was proposed, and a middleware based on the former was implemented. Their framework allowed one to design and program services coordination policies relying on event notification only. The coordination poli-
cies were regulated by specifying how services react to events. In fact, the coordination policies are processes in a process calculus. However, in their formal model, receivers must be explicitly specified when an event was published and a point-to-point channel was used. This is not the case in our scenarios. The work of [14] was an extension of [13] that only addressed the service choreography problem. Their formal model was tailored to express coordination patterns within the event notification paradigm by combining suitable mechanisms such as event and network hiding, network reconfiguration and multicast communication. They did not introduce event session concepts.

For services collaborating, it is necessary that an interaction between multi-participants is managed as a session, which can be explicitly defined and handled, and especially necessary for transactions. The works in [11], [12] proposed some session handling mechanism for service-oriented computing. We adopt the method in [12] to handle service sessions, where a session is defined as a correlation set. A correlation set is a set of name/value (name/value) pairs in processes. If different participants have the same correlation set, i.e., same process names and same values, they possess the same session. They did not focus on events.

VI. CONCLUSION

This paper discusses how to integrate EDA and SOA, the event session concept is introduced, which was often neglected in most existing works about EDOSOA. This paper is then proposes an EDOSOA infrastructure for IoT services, which consists of distributed resource pools, distributed event-based service environment, and event relation synergizing utility. Some applications and experiments are given to show the concept proof for such event-driven SOA for IoT services.

VII. ACKNOWLEDGMENT

This work has been supported by is supported by National Natural Science Foundation of China (Grant No. 61372115,61132001); 973 program of National Basic Research Program of China (Grant No.2012CB315802); Project of New Generation Broadband Wireless Network under Grant (Grant No.2012ZX03005008-001).

REFERENCES


