IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received October 22, 2017, accepted December 19, 2017, date of publication January 1, 2018, date of current version March 9, 2018.

*Digital Object Identifier 10.1109/ACCESS.2017.2788410*

# CodingBlind: Automated Cloud Services Generation From Printed Forms and BPMN

**HAN YU[1], CONGCONG YE[1], HONGMING CAI[1], (Senior Member, IEEE), LIHONG JIANG[1], CHENG XIE[2], AND BOYI XU[3]**

[1]School of Software, Shanghai Jiao Tong University, Shanghai 200240, China
[2]School of Software, Yunnan University, Kunming 650504, China
[3]College of Economics and Management, Shanghai Jiao Tong University, Shanghai 200052, China

Corresponding author: Hongming Cai (hmcai@sjtu.edu.cn)

This work was supported by the National Natural Science Foundation of China under Grant 61373030 and Grant 71171132.

**ABSTRACT** Cloud service is a new trend in constructing enterprise applications. However, implementing cloud services is time consuming, costly, and error prone. Moreover, there are concerns about the isolation and flexibility of multitenancy cloud services. To solve these problems, we propose a novel and easy approach: CodingBlind, which automatically generates cloud services from the most commonly used printed forms and graphical business process modeling notation (BPMN). CodingBlind provides a natural solution for the deployment and migration of systems in clouds. First, printed forms are mapped onto entity-relationship models (ER models) to generate data services. States of data and a finite state machine (FSM) are constructed from the BPMN. Based on the FSM and the ER models, quadruple models for cloud services are generated. Then, a complete system package is generated according to the quadruple model. Finally, all of the components are packaged and deployed to a cloud. CodingBlind is easy to use because it only takes printed forms and BPMNs as inputs. Self-defining data and business logic make the generated cloud services flexible. Moreover, the business logic is strictly controlled by the FSM, which provides natural isolation for cloud applications. Case studies and experiments are conducted to assess the proposed approach, and the results show that it is feasible, convenient, and flexible.

**INDEX TERMS** Automated service generation, cloud service, multi-tenancy isolation, self-service, model-driven development.

## I. INTRODUCTION

In the era of the Internet, many enterprises have hoped to migrate their information systems to clouds to provide business services at a low cost and in a safe environment. Thus, using cloud services has become a new trend in enterprise application construction [1], [2] in fields such as manufacturing [3], [4], health [5], and education [6]. To help enterprises build their cloud applications, services in three levels have been developed: Software-as-a-Service (SaaS) [7], Platform-as-a-Service (PaaS) [8] and Infrastructure-as-a-Service (IaaS) [8]. However, there are still some obstacles that can prevent enterprises from constructing their own services. (1) To that end, SaaS provides standard data interfaces and fixed business processes for various enterprises. However, unique requests, data or business processes are not satisfied by these fixed services. Enterprises must either align or give up their data and activities; otherwise, they must pay a large amount of money to obtain self-defined services from service providers over a long period of time. (2) PaaS and IaaS only provide a cloud platform or servers for enterprises, and more efforts are needed for enterprises to develop their own cloud applications. Consequently, for Small and Medium-sized Enterprises (SMEs), the ability to rapidly construct cloud services is absent. (3) During the runtime of cloud services, although the cloud platforms provide multi-tenancy isolation measures [9], [10] such as virtual machines and containers in the platform layer, isolation in the business layer is difficult to guarantee.

To provide an easier and more economical way for SMEs to quickly obtain their own cloud services, three goals must be fulfilled by a new approach: (1) the approach needs to be fully automated such that no manual work is needed during the cloud-services-generation process to save time and money for enterprises. (2) The new technique needs to provide flexible self-service for enterprises that can process their self-defined data and business processes. (3) The new technique must be able to ensure isolation among multi-tenant cloud services.

To automatically generate cloud services, the two critical technical factors are data and functions [11]. The data are the fundamental elements that represent every entity involved in a system, and the functions are the interfaces that abstract business processes and provide services for users. If we allegorize the data as the blood of a system, the business processes are the blood vessels that transport the data to every corner of the system. Therefore, for a self-defined cloud service to be constructed, the obtained data must be involved in the enterprise's processes and business activities.

Previously, research has been done on domain elements extracted from a unified model [12], [13], and the specifications of the use cases were employed to obtain the metadata used in a system [14]. However, these approaches still need experts to design input models or detailed specifications, which involves large amounts of time and manpower. For the functions, there have been many attempts in business process modeling and management [15] to build simple models to identify functions [16] and drive systems to work [17]. For example, Petri net [18] is a traditional system modeling language, jBPM [19] is a development-oriented business process management tool, and BPEL [20] is a formal business process execution language that is generally used in web service development. These languages and tools are not friendly to users with little development experience for the complicated rules. Hence, a technique with simple inputs that allows even end users to design and implement cloud services is needed.

To solve the problem, note that the data in an enterprise are fully stored in the format of printed forms even if they have enterprise systems [21]. For example, purchase order forms record the materials to buy and the suppliers to use, and work orders record the tasks to do and the factories in which the work is assigned. These forms have similar structures and carry almost all of data in an enterprise. With templates of various forms, the data involved in selecting the men, machines and materials [22] can be easily extracted [23]. For the functions, using diagrams and notations, BPMN [24] is a convenient bridge from process design to executable code for stakeholders who raise requirements but lack coding experience.

In this paper, we propose an automated approach to generate cloud services that we call CodingBlind. This approach takes blank printed forms and BPMN from the standard business process library or user designs as input and generates self-defined cloud services in an independent package. First, the property trees are extracted based on the structure of the printed forms. Furthermore, ER models [25] are generated from property trees. Then, a global ER model is constructed by integrating the generated ER models. In addition, an FSM is constructed from the BPMN, and the state transitions of the entities are identified from this FSM. By matching the entities with the data in the FSM, a stateful ER model is generated. Next, a quadruple model is proposed to represent the cloud services. The quadruple model combines the information in the ER models and the FSM. Finally, based on the Model-View-Controller (MVC) design pattern [26], a cloud service system in an independent package that contains all of the constructed components is generated. By deploying the system package in the cloud, enterprises can easily access their self-defined and safe cloud services. In conclusion, four main contributions are made in this paper:

- An automated cloud-services-generation technique is proposed. This approach takes printed forms and the BPMN as inputs, which provides an easy way to develop cloud services and a natural solution to deploy and migrate cloud systems.
- Data self-service is implemented in the approach. Different tenancies can use different data in printed forms and business processes in BPMN to define their own services, which provides convenient access to cloud services.
- A natural isolation of the cloud service systems is guaranteed in the business layer by utilizing an independent package of cloud services that includes data resources, services, and views. This setup reduces the difficulty of isolation in containers and virtual machines.
- A prototype tool and a series of experiments are presented to demonstrate that the proposed approach is a feasible way to generate an information system automatically.

The rest of the paper is organized as follows. Section II introduces the basic concepts and essential prerequisite for the paper. Section III presents the detailed process and methods of our approach, and Section IV assesses the approach with several case studies and experiments. Section V discusses the results of our work and related work in this field. Finally, Section VI concludes the paper and illustrates the future directions of this work.

## II. PREREQUISITE

### A. STRUCTURE OF BUSINESS FORMS

Before the work begins, we first analyze the structure of the forms. According to our research on more than 500 business forms from the results of Google Picture Search,[1] we conclude that the skeletons of business forms all have a similar structure composed of four parts, as shown in Figure 1:

#### 1) TITLE

The title implies the entity that the form describes. As an example, consider the form on the left in Figure 1. This form includes the information from a piece of a sales order. However, the title is optional in the form structure. In the form on the right in Figure 1, if the detailed information contains its own title, then we believe that the first title that appeared in the header of the form is not the title.

#### 2) RELATED INFORMATION

The related information contains the detailed information of the related entities in the business process. These entities can

[1] http://www.google.com/search?site=imghp&tbm=isch&source=hp&biw=1436&bih=780&q=business+form

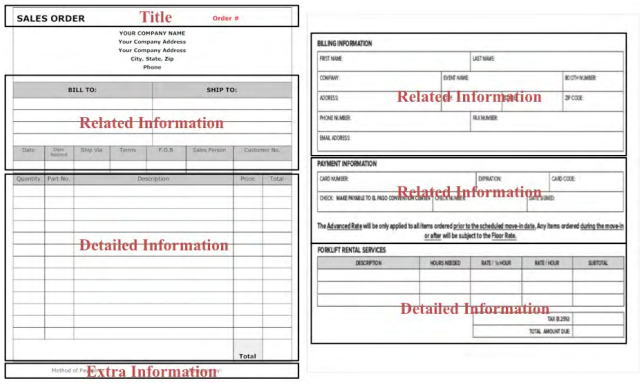



**FIGURE 1.** The structures of business forms with examples. The form on the left has the standard four-part structure, and the form on the right has a variant four-part structure.

have their own titles and extra information. If we take a sales form as an example, the customer's detailed information is listed before the list of the order details. However, only the customer ID is directly related to the order, and the detailed information consists of the properties of the customers. The related information is optional because it is possible that no related entities are involved, but this occurrence is rare in business processes.

#### 3) DETAILED INFORMATION

The detailed information displays items with the same property fields in a vertical table. In a sales form, the detailed information contains the information of each sold product; in a work form, this information contains the products to produce or the work to do. For most transaction data, this information is compulsory, but it can be omitted in certain business forms, such as a paper information form in a paper review system.

#### 4) EXTRA INFORMATION

In this portion, all of the information except the detailed information and the related information are placed, such as the payment choice, delivery date and method of shipping. This part contains the properties of the entity that the form describes and is also an optional part of a business form.

### B. RESTRICTIONS FOR THE BPMN

To obtain function-related information, some restrictions have to be specified in the BPMN. First, the BPMN models need to follow the BPMN Basic Rules and Clarifications.[2] Next, the naming of the elements in BPMN should follow the BPMN Naming Conventions Best Practices[2]. That is because the names are the key to matching the data and discovering the function operations. With these two basic and simple restrictions, a usable BPMN can easily be constructed. In this paper, we use examples in the BPMN official guide[3] to demonstrate the methods and experiments on BPMNs.

[2]http://www.bpmnquickguide.com/view-bpmn-quick-guide
[3]http://www.bpmn.org/#tabs-examples

## III. PROPOSED APPROACH

Our approach, which we call CodingBlind, is an automated cloud-services-generation process that consists of 5 central procedures: ER Model Construction, FSM Building, Quadruple Cloud Service Modeling, Independent Cloud Services Generation, and Deployment and Execution. The inputs of our approach are the various blank printed forms used in the business process and a BPMN model that describes this process. The schematic diagram is shown in Figure 2. Users provide their own in-use business forms, and the ER Model Construction component recognizes the forms and identifies the domain elements from the printed forms. Furthermore, users can choose the existing business process templates from the standard process library or directly design their own processes in the BPMN. The FSM Building component generates an FSM from the BPMN to obtain the states of data and the function operations. Then, the Cloud Service Modeling component constructs a quadruple model of the services to achieve better representations. Moreover, the process of Independent Cloud Services Generation uses the quadruple model to generate the instances of the cloud service package. Lastly, the Deployment and Execution process flexibly deploys the service package based on the MVC pattern in clouds.

### A. ER MODEL CONSTRUCTION FROM PRINTED FORMS

In this phase, printed forms are first transformed into Property Trees (PTs), which are mapped onto ER models via mapping rules. These ER models are integrated into an entire ER model as the basis of a complete data system. Data objects in the BPMN are matched with entities to identify states from the BPMN. Then, the state tables for the entities are added to the integrated ER model. Finally, a stateful ER model is built.

#### 1) STRUCTURE-BASED PROPERTY TREE ABSTRACTION

To extract property trees from business forms, we first use an OCR engine [4] to transform the printed forms into metadata files that provide a tuple containing the coordinates of every character and line element. According to the relative positions of the lines and characters, we decompose the forms into tables and texts.

The four parts mentioned in Section II-A are labeled according to Algorithm 1. First, detailed information is located by detecting the table with a single filled property row and multiple empty rows. Then, the title is identified by the location and font size. If no title is found, an empty title will be applied for further confirmation. The other blocks are labeled as related information or extra information.

Inside some table blocks, there can be more than one entity. For example, in Figure 1, `BillTo` and `ShipTo` are in the same tables but are different entities. By detecting the permutations of both filled cells and empty cells and similar properties, the partition is completed. The filled cells in the tables are the properties of the entities.

After labeling the areas in a form, a property tree (shown

[4]http://ocrsdk.com/

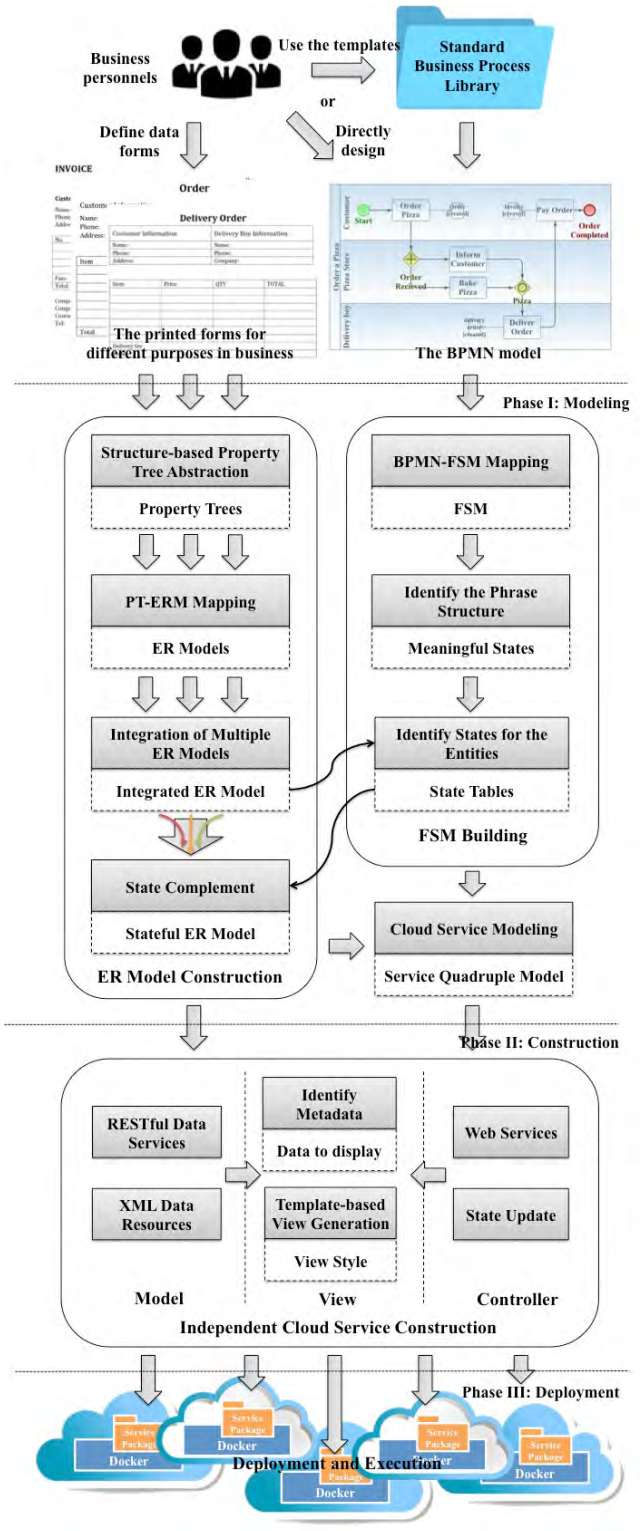



**FIGURE 2.** Framework of the proposed approach.

in Figure 3) is built to model the form. The title that represents the entity described by the form is the root of the property tree. If no title is found, then the entity is temporarily named *Root*. A related entity is a child node of the root, and its properties are children of the entity. The other child nodes of the root

**Algorithm 1** Area Partition

**Input:** Tables $\{Ta\}$, Texts $\{Te\}$
**Output:** Form $F$

**for** $Ta \in \{Ta\}$ **do**
  **if** $Ta.rows = (textrow, (emptyrow)+)$ **then**
    Put $Ta$ in $F.\{Related\}$
  **end if**
**end for**
Find the table $Ta$ with the maximum height
Remove $Ta$ from $F.\{Related\}$; $F.\{Detail\} \leftarrow Ta$
**if** text $Te$ is directly above $F.\{Detail\}$ **then**
  $F.Title \leftarrow Te$
**else**
  Find the $Te$ with the maximum font size and above all of the tables, $F.Title \leftarrow Te$
**end if**
**for** $(Te \in \{Te\}) and (Te \neq F.Title)$ **do**
  **if** $Te$ is single-line and beyond $Ta$ **then**
    $Ta.name \leftarrow Te$
  **else if** $Te$ is single-line and beyond $Te_m$ **then**
    Put $Te_m$ in $F.\{Related\}$, $Te_m.name \leftarrow Te$
  **else**
    Put $Te$ in $F.\{Plus\}$
  **end if**
**end for**
Put all $Ta \notin F.\star$ in $F.\{Plus\}$
**return** $F$

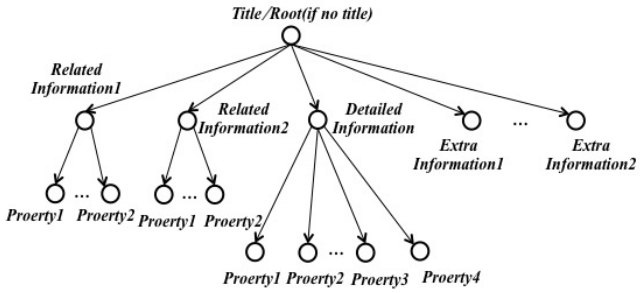



**FIGURE 3.** The structure of the property tree extracted from the business forms.

are the detailed information and the properties in the extra information. The detailed information node is the parent of all of the properties in the detail table. The properties in the extra information area are direct children of the root.

### 2) PT-ERM MAPPING

In this step, the PTs are mapped to ER models to further construct tables in the database. Three types of entities and the relationships among them are generated according to Table 1. In addition, an ER model is generated, as shown in Figure 4.

#### *a: TRANSACTION ENTITIES*

In a system, the transaction data record business activities and are changed frequently. Sales orders, work orders and shipping orders are common transaction data in a business.

**TABLE 1. The mapping rules for transforming a property tree into an ER model.**

| Entity Type | Element in Property Tree | Representation in Entity |
|---|---|---|
| Transaction Entity $E_T$ | Title | $E_T.name$ |
| | Related entities | $E_T.\mathcal{FK}$ |
| | Extra information | $E_T.\mathcal{ATTR}$ |
| Detail Entity $E_D$ | Title | $E_D.name$ |
| | Transaction Entity | $E_D.FK$ |
| | Properties of detailed information | $E_D.\mathcal{ATTR}$ |
| | Title→ Detailed Information | $E_T$: One-To-Many: $E_D$ |
| Related Entity $E_R$ | Name of related information | $E_R.name$ |
| | Properties of related information | $E_R.\mathcal{ATTR}$ |
| | Title→ Related Information | $E_T$: Many-To-One: $E_R$ |

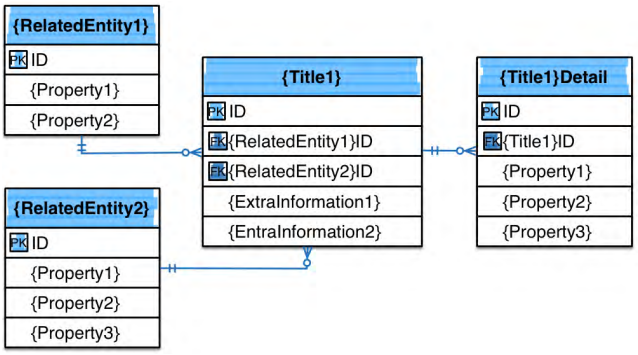



**FIGURE 4. The general ER model constructed based on individual forms.**

Furthermore, customer information or paper information can sometimes be transaction entities in processes such as customer privilege approvals and paper reviews. These data are always presented as sheets and forms for communication across departments. Therefore, a property tree extracted from a form can be directly mapped to a series of transaction entities according to the mapping rules in Table 1. For a given property tree, the title is transformed into $E_T.name$, where $E_T$ denotes the core transaction entity. The related information is transformed into other entities, and only the primary keys of these data are preserved in the core entity as foreign key attributes, namely, $E_T.\mathcal{FK}$. The extra information is directly mapped to $E_T.\mathcal{ATTR}$, which denotes the attributes of the entity.

### b: DETAIL ENTITIES

Detailed information are not stored in the core entity, but in an independent entity $E_D$, where $E_D.name = E_T.name + \text{“}Detail\text{”}$. $E_D$ takes the properties of the detailed information as attributes and refers to the core entity by adding a foreign key attribute containing the core transaction entity id, which is denoted as $E_D.\mathcal{ATTR} = PT.DetailedInformation.\mathcal{P} + E_T.PK$. Correspondingly, the links from the title to the detailed information are mapped to one-to-many relationships from $E_T$ to $E_R$.

### c: RELATED ENTITIES

The related entities evolve from the related information in the property tree. These entities are master data; these data are shared between multiple systems or business processes and are infrequently changed. The Client, Supplier and Customer are typical instances of master data. In the forms, important master data appear in the related information area. Sometimes related entities are also transaction data, for example, a sales order can be a related entity to a delivery order. However, we can still recognize these related transaction data as master data in this case because modifications of the core transaction entity never change the data in the related entities. Because of the independence of the related entities, they can be directly mapped to entities $E_R$ with the properties mapped onto the attributes in $E_R.\mathcal{ATTR}$ in the ER model. The links from the title to the related information in property trees are transformed into many-to-one relations from $E_T$ to $E_R$, and the foreign keys for every related entity are reserved in the core transaction entity. A final generated ER model is shown in Figure 4. All of the unique ID properties are generated in the standard format instead of being derived from property trees to ensure the existence and uniqueness of the ID fields.

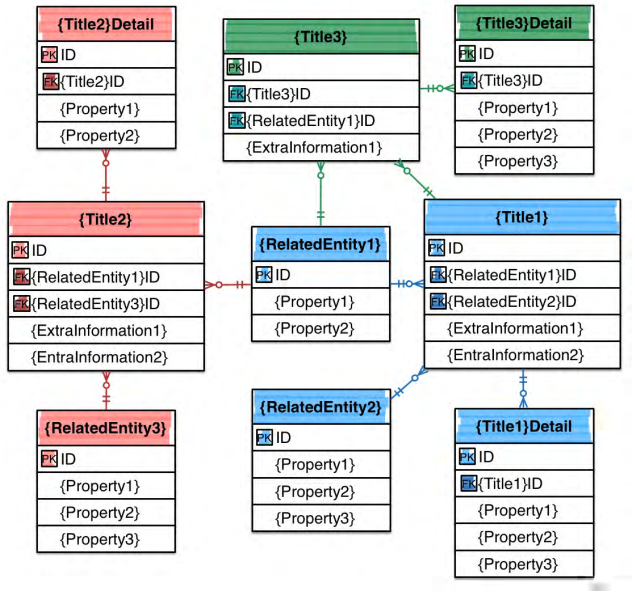



**FIGURE 5. The integrated ER model, which incorporates multiple ER models.**

### 3) INTEGRATION OF MULTIPLE ER MODELS

The next step is to integrate models from different forms into an integrated ER model (shown in Figure 5) as the basis of the data service. As we mentioned in Section III-A2, a transaction entity can be a master datum of another transaction entity; e.g., a sales order is a master datum for a delivery order. Therefore, when the models integrate, there can be more than one generated entity that represents the same real entity. To resolve the duplication issue, the merging is to be conducted as in Figure 6. By comparing the name strings of the entities, the duplicated entities can be found. An entity is a duplicate if and only if one or more entities with the same name are found in all of the ER models. However,

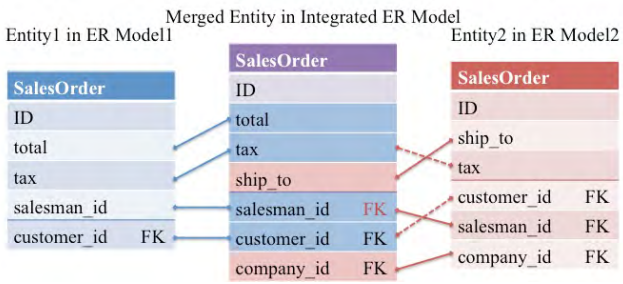



**FIGURE 6.** An example of merging a duplicate entity.

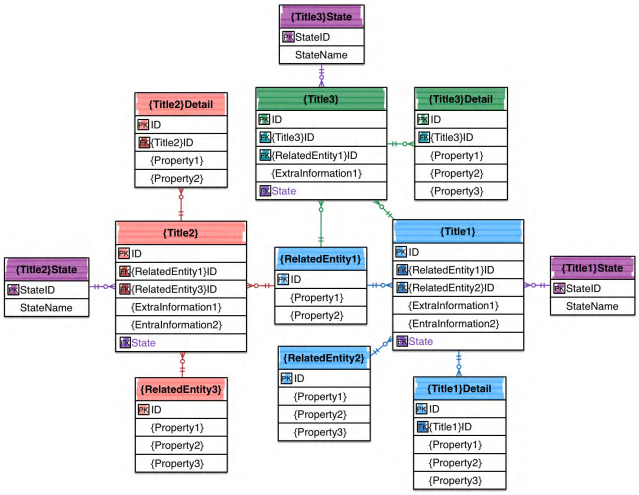



**FIGURE 7.** The final stateful ER model integrating multiple ER models.

the duplicates are not directly abandoned, as those entities with the same name may have various attributes for different usages. As a result, all of the attributes from entities with the same name are merged. If an attribute appears for the first time, it will be preserved; otherwise, it will be abandoned. If one appearance of the attribute is a foreign key, then it will be preserved as a foreign key to maintain the relationship among the entities.

#### 4) STATE COMPLEMENT

The state, which shows the data process in action, is an important attribute of the transaction entities in an enterprise system. Consider a sales order as an example: when it is created by a sales employee, it may have the state `submitted`, and when it is delivered by the shipping department, the state may be changed to `shipping`. Although states continue to change during the business process, no state information can be found only from printed forms. Therefore, we introduce the BPMN into the ER model when it is constructed to obtain the states of the transaction entities in the business processes. The methods used to identify the states of the entities are introduced in Section III-B. As shown in Figure 7, for the entities with multiple states, a state entity is built with the title of `{Title}State`, and an attribute `state` is added to the `{Title}` entity. Finally, a complete stateful ER model for cloud services is built.

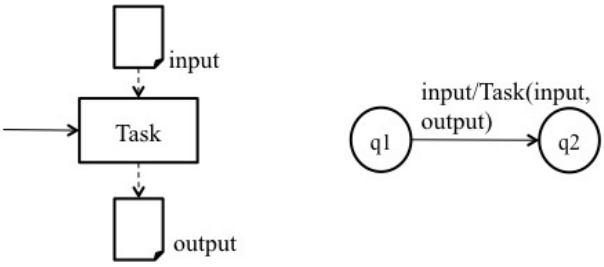



**FIGURE 8.** An example of the task mapping to FSM.

### B. BUILDING AN FSM FROM BPMN

BPMN is an easy and convenient way for users to design business processes. However, it is not formal enough to directly control the process of a system, and the data states are not all explicitly illustrated. Therefore, we transform the BPMN into an FSM as the logic controlling module of the system. An FSM is denoted as a quintuple $(\mathcal{I}, \mathcal{O}, \mathcal{Q}, q_0, \delta, \mathcal{F})$; the notation is defined below.

- $\mathcal{I}$ is a finite set of input symbols,
- $\mathcal{O}$ is a finite set of output symbols,
- $\mathcal{Q}$ is a finite set of states,
- $q_0$ is the initial state in $\mathcal{Q}$,
- $\delta : \mathcal{Q} \times \mathcal{I} \rightarrow \mathcal{Q}$ is a finite set of state transition functions,
- $\mathcal{F} : \mathcal{Q} \times \mathcal{I} \rightarrow \mathcal{O}$ is a finite set of output functions.

In this phase, we first map the BPMN into a formal FSM. Then, a semantic model is used to extract meaningful state data. Finally. we identify the states of the entities to construct a state table in an integrated ER model.

#### 1) BPMN-FSM MAPPING

##### *a: TASK*

A task in BPMN is intended to accomplish a job. We define a task in BPMN as below:

$$Task = [Name, Type, InFlow, OutFlow, InMessage, OutMessage, InData, OutData, Lane, Pool] \quad (1)$$

A task updates the state of a system.Therefore, a task is mapped onto a state transition function $\delta$ and an output function $F$ in the FSM, as shown in Figure 8. $\delta_t$ takes the input elements of Task $t$ as $I$ and advances $q_0$ to the next state $q_1$. In addition, $F_t$ takes the *InData* as $I$ and the output symbols *OutData* and *OutMessage* as $O$. If the input or output of the task is not declared, then $F.I = *$ or $F.O = empty$. If the task has subprocesses, then the state transitions in these subprocesses are between the two states of the task. The mapping rules can be denoted as below:

$$\begin{aligned} \delta_t &:= q_0 \times [t.InMessage, t.InFlow, t.InData] \rightarrow q_1 \\ F_t &:= q_0 \times t.InData \rightarrow [t.OutData, t.OutMessage] \end{aligned} \quad (2)$$

##### *b: GATEWAY*

Three basic gateways are supported to map into the FSM: exclusive, parallel and inclusive gateways [27]. There are

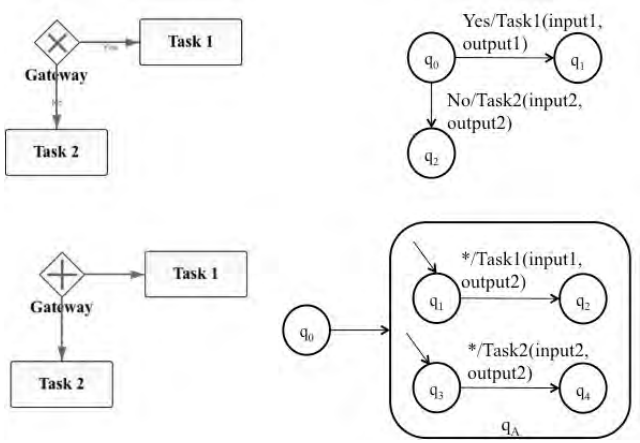



**FIGURE 9.** Mapping rules for diverging gateways.

two types of appearances of gateways: converging and diverging gateways. A gateway in BPMN is denoted as below:

$$Gateway = [Name, Type, InFlowSet, OutFlowSet, Lane, Pool] \quad (3)$$

According to the different types and appearances of gateways, different rules are applied.

An **exclusive gateway** is a choice gateway where only one branch is executable at a time. As shown in Figure 9, for a diverging exclusive gateway structure, $q_0$ is the start state of the gateway, and the conditions of the gateway are the input symbols of $q_0$. According to the condition, the next element $xg.OutFlow_i.to$ composes the output function $F^i_{xg}$ and the state transition function $\delta^i_{xg}$. For the *ith* branch, the transfered FSM process is denoted as below:

$$\begin{aligned} \delta^i_{xg} &:= q_0 \times [xg.OutFlow_i, xg.OutFlow_i.to.InData] \\ &\rightarrow q_i \\ F^i_{xg} &:= q_0 \times xg.OutFlow_i.to.InData \\ &\rightarrow [xg.OutFlow_i.to.OutData, \\ &\qquad xg.OutFlow_i.to.OutMessage] \end{aligned} \quad (4)$$

Take the instance in Figure 9 as an example, the `Yes` branch is mapped onto a state transition function $\delta^1_{xg} := q_0 \times [Yes, Task_1.InData] \rightarrow q_1$ and an output function $F^1_{xg} := q_0 \times Task_1.InData \rightarrow [Task_1.OutData, Task_1.OutMessage]$. Likely, $Task_2$ is mapped onto the output function of $\delta^2_{xg}$. Similar to the diverging gateways, a converging exclusive gateway is a reverse structure such that $Task_1$ and $Task_2$ share a single end state instead of a start state.

A **parallel gateway** is a gateway such that both of the branches execute at the same time. Although the traditional FSM does not support paralleling, a layer model can be used in the FSM to support parallel activity. As shown in Figure 9, for a diverging parallel gateway, $q_0$ is the start state. All of the input symbols are allowed and an empty action is undertaken, which is omitted in the figure. Then, a new layer is created to replace the end state. The new layer contains two new independent FSMs, and each of them is an FSM starts with the successor tasks of the gateway. The mapped FSMs are formally represented by Formula 5 and Formula 6. The structure indicates that two parallel FSMs have started. When a converging gateway appears in the sequence, the FSMs in the inner layer all reach the end and the outer layer continues. Take the instance in the bottom of Figure 9 as an example, the first layer is from $q_0$ to $q_A$. Inside the state $q_A$, two new FSM starts. The first inner layer FSM starts with state $q_1$ and output function $Task1$, while the second inner layer FSM start with $q_3$ and output function $Task2$. The converging parallel gateway is similar; the only difference is that $q_0$ becomes the end state after the FSMs in the new layer have all stopped.

$$\begin{aligned} \delta_{ag} &:= q_0 \times * \rightarrow q_A \\ F_{ag} &:= q_0 \times * \rightarrow empty \end{aligned} \quad (5)$$

*Inside state* $q_A$:

$$\begin{aligned} q^i_0 &:= q_{2i-1} \\ \delta^i_{ag} &:= q_{2i-1} \times [ag.OutFlow_i, ag.OutFlow_i.to.InData] \\ &\rightarrow q_{2i} \\ F^i_{ag} &:= q_{2i-1} \times ag.OutFlow_i.to.InData \\ &\rightarrow [ag.OutFlow_i.to.OutData, \\ &\qquad ag.OutFlow_i.to.OutMessage] \end{aligned} \quad (6)$$

An **inclusive gateway** is a gateway such that if the condition is satisfied, then two branches will both execute; otherwise, the default branch will execute. Because an inclusive gateway can be considered the combination of an exclusive gateway and a parallel gateway, no new mapping rules need to be declared for inclusive gateways. The mapped FSM can be denoted as Formula 7 and Formula 7, where $\delta^t_{og}$ denotes the state transition function represents the condition is satisfied and $\delta^f_{og}$ derives from the default branch.

$$\begin{aligned} \delta^f_{og} &:= q_0 \times [og.OutFlow_f, og.OutFlow_f.to.InData] \\ &\rightarrow q_f \\ F^f_{og} &:= q_0 \times og.OutFlow_f.to.InData \\ &\rightarrow [og.OutFlow_f.to.OutData, \\ &\qquad og.OutFlow_f.to.OutMessage] \\ \delta^t_{og} &:= q_0 \times * \rightarrow q_T \\ F^t_{og} &:= q_0 \times * \rightarrow empty \end{aligned} \quad (7)$$

*Inside state* $q_T$:

$$\begin{aligned} q^t_0 &:= q^t_0 \\ \delta^t_{og} &:= q^t_0 \times [og.OutFlow_t, ag.OutFlow_t.to.InData] \\ &\rightarrow q^t_1 \\ F^t_{og} &:= q^t_0 \times og.OutFlow_t.to.InData \\ &\rightarrow [og.OutFlow_t.to.OutData, \\ &\qquad og.OutFlow_t.to.OutMessage] \\ q^f_0 &:= q^f_0 \\ \delta^f_{og} &:= q^f_0 \times [og.OutFlow_f, og.OutFlow_f.to.InData] \\ &\rightarrow q^f_1 \end{aligned}$$

$$F^f_{og} := q^f_0 \times og.OutFlow_f.to.InData \rightarrow [og.OutFlow_f.to.OutData, og.OutFlow_f.to.OutMessage] \quad (8)$$

*c: EVENT*

There are three types of event in BPMN: a start, end and intermediate event. A event is defined as follow:

$$Event = [Name, Type, InFlow, OutFlow, InMessage, OutMessage, Lane, Pool] \quad (9)$$

A start event is mapped as below:

$$\begin{aligned} q_0 &:= q_0 \\ \delta_0 &:= q_0 \times * \rightarrow q_1 \\ F_0 &:= q_0 \times * \rightarrow empty \end{aligned} \quad (10)$$

An end event can be an end, a terminate or an abort event. It updates the state of data. An end event can be transfered into FSM elements as below:

$$\begin{aligned} \delta_n &:= q_{n-1} \times [event.InMessage, event.InFlow.OutData, event.Name] \rightarrow q_n \\ F_n &:= q_{n-1} \times [event.InMessage, event.InFlow.OutData, event.Type] \rightarrow event.Name \end{aligned} \quad (11)$$

For intermediate events (such as task mapping rules), a two-state machine is built:

$$\begin{aligned} \delta_e &:= q_0 \times [event.InMessage, event.InFlow.OutData, event.Name] \rightarrow q_1 \\ F_e &:= q_0 \times [event.InMessage, event.InFlow.OutData, event.Type] \rightarrow [event.OutMessage, event.InFlow.OutData, event.Name] \end{aligned} \quad (12)$$

Based on the mapping rules, each element in BPMN can be mapped onto a component of an FSM. By going through the sequence flow, we connect all the components and build an FSM. If subprocess exists, the whole subprocess will be inserted into the sequence. For the semantic aspect, we have $F.name = element.name$. Similarly, the *Lane* and *Pool* of the elements in BPMN are also preserved in $F$. Next, we match the states in the FSM with the entities we obtained from the printed forms to build a stateful ER model.

#### 2) IDENTIFY THE PHRASE STRUCTURE

In this step, WordNet [28] is introduced to discover the semantic meaning of the functions $\mathcal{F}$ to further extract meaningful states and useful cloud services. First, for each function $F : q_i \times I_i \rightarrow O_i$, we stop the words of $F.name$, and then the parts of speech (POS) and the tags of all of the words and phases in $F.name$ are retrieved from WordNet to obtain the POS pattern $F.name_{pattern}$. Then, $F.name_{pattern}$ is matched to a *verb-object* pattern to uncover the verb word $F.operation$. Next, the past participle of $F.operation$ is the name of the end states $q_j$ of the corresponding state transition function $\delta : q_i \times I_i \rightarrow q_j$.

#### 3) IDENTIFY STATES FOR THE ENTITIES

$\mathcal{F}$ in an FSM takes the data as inputs and outputs. Moreover, the corresponding state transition function $\delta$ transforms the states of the system, which are also the representatives of the states of the data. Therefore, by identifying the input and output of $F$, the states of the entities become obvious.

First, we describe the constructed FSM. For an output function $F$ with certain inputs and outputs, nothing needs to be done. However, if $F.I$ or $F.O$ is empty, then we need to find the inputs and outputs. If a function $F.I = *$, we backtrack based on the outputs of the previous functions. If the output is an entity that is related to $F.O$, then the output of the previous function is given to $F.I$. If no such output exists, then $F.I = *$ still holds. If $F.O = empty$, then the input is given as the output of the task.

After we have determined all of the inputs and outputs of the functions, the data related to the states need to be identified by identifying the states for the entities. To record the entity with the state, for all $q$ in $\mathcal{Q}$, an entity table is preserved. For a state $q$, the functions reaching $q$ are denoted as $F_q$. First, all of the $F_q.O$ are added to $q.table$. Furthermore, all of the $F_q.O$ are matched to the ER model to find the corresponding entities. Then, for each entity, foreign keys are retrieved to obtain the related entities. If a related entity has appeared before $q$, then we consider the related entity to have the state $q$ and the entity is added to $q.table$. In this way, the entity tables of the states are built. By processing all of the tables, the states are recorded according to the entities in the state tables. By adding the state tables to the integrated ER model, the stateful model is constructed.

### C. QUADRUPLE MODEL CONSTRUCTION FOR SERVICE REPRESENTATION

To integrate all of the information retrieved from the printed forms and BPMN models to derive an independent cloud service package, a quadruple model is proposed. The model is denoted as

$$\langle URI, subject, predicate, object \rangle \quad (13)$$

Figure 10 shows an instance of this quadruple model for service. There are four types of resources: Roles, Forms, Conditions and Services; the quadruples show the relations among them. Each service has three types of predicate: `hasRole`, `use` and `hasCondition`. These predicates respectively connect the services with the roles, forms and conditions. A service can own multiple quadruples of the same type. Each condition also has its predicates, as shown in Figure 10. The conditions and functions are directly mapped from the FSM built in the previous steps. In addition, the forms are entities in the ER model, while the roles are derived from the participants in the BPMN model. By using the service quadruple model, an independent service package becomes feasible.

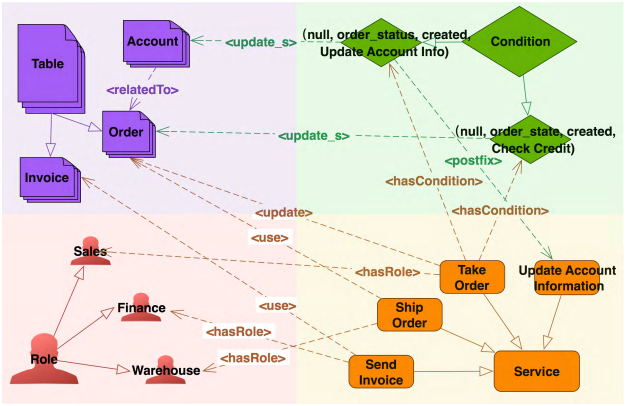



**FIGURE 10.** An example of a service quadruple model.

For an output function $F$, a Service instance is generated. It is notable that the form resource contains all of the properties of the entity, and parts of the properties are actually used in the Service. As a result, the single ER model is used to retrieve the usable properties. For the `use` relation, if $F.I \neq *$ and the resource of $F.I$ is in the ER model of the output resource, the properties that appear in the representative entity of $F.I$ in the ER model are retrieved as the `use` resource. Then, all of the entities and properties that appear in the ER model of the output are organized as the resources that have `updated` relations with the Service.

For a state $q$, a Condition instance is generated. The Condition is connected with the Services that directly lead to the state. The connection is a `hasCondition` relation. By finding the corresponding state transition function $\delta$ after $q$, the input of $\delta$ can be set as the `State` of the Condition, which means that the condition is invoked in this state. Then, the output function of $\delta$ becomes the postfix service of the condition. For the entities in the entity table of $q$, the states of the entities are connected with the Condition in relations of `update_s` with the value of the new state name.

### D. INDEPENDENT CLOUD SERVICES GENERATION

After the services are modeled as quadruples, the cloud services are generated. Then, an MVC design pattern is applied.

#### 1) MODEL

In this phase, the data services are generated on the basis of the ER model. First, the metadata are generated from the stateful ER model. Then, the RESTful data services in the Web Application Description Language (WADL)[5] are generated with separated ER models.

##### a: DATA RESOURCES

According to the ER model, a database is built. Every table stands for an entity in the ER model. Then, the RESTful data resources are generated. The RESTful services retrieve and post the data resources in the format of XML. A template-driven mapping is used to generate the data sources in the XML format. For every table in the database, an XML file is generated with the same name, and a `SELECT * from {Table}` command is embedded into this file. A data transformation middleware is used to deal with the templates. When the file is handled by the middleware, the SELECT sentence is replaced by the execution result, and the complete resources are derived.

##### b: DATA SERVICES

After the data resources are generated, the data services can be easily built on the basis of the data resources. The RESTful services in the WADL are applied in our framework. For a type of resource, basic GET/PUT/POST/DELETE operations are defined on a group of resources and a single resource. DELETE works on a single resource when the ID is provided in the URI and a group of resources when no ID found. GET methods have a group of parameters that come from the resource properties to operate `SELECT` operations on resources. POST and PUT provide modifications and additions interfaces whenever the POST method needs a resource as an input and the the PUT method does not.

#### 2) CONTROLLER

The controller implements the business logic in the system according to the quadruple model. To provide cloud services, the WADL is used to generate advanced web services that implement the functions in the system.

##### a: WEB SERVICES

For an function $F$ in the quadruple model, a web service is generated. First, the function name is assigned to the name of the service. Then, a GET operation is prepared to load the Form resources used by $F$, and a PUT or DELETE operation is employed for the Form resources updated by $F$. Then, to identify the operation to perform, the name of the service is analyzed by WordNet. If the name of the service contains a synonym of `delete` or `remove`, then the DELETE operation is applied. Otherwise, PUT is used. The resources updated by $F$ are loaded as the resources of the operation.

##### b: STATE UPDATE

The operations are done within a certain view, and the post data are sent back to the controller to control the modifications of the model layer. By finding the corresponding Condition in the quadruple model, the forms to update and the next service to use are discovered. Then, the resources are updated and the view jumps to the next view under the control of the quadruple model.

#### 3) VIEW

A view layer is composed of a group of pages that provide an interactive interface for the end users. Three elements of the metadata need to be determined in a view: the title, request data and post data. The title is the name of the page.

[5]http://www.w3.org/Submission/wadl/

The request data stand for the required input data or the static data shown on the page. Lastly, the post data refer to the data to be sent back to the server after user operations have been performed.

#### a: IDENTIFY METADATA

First, three types of metadata are identified from the web services. For every web service, a page is built. The title of the page is derived from the name of the service. The resources in the GET operation are the request data. Similarly, the resources of the other operations are considered as the post data on the page.

#### b: TEMPLATE-BASED VIEW GENERATION

After the metadata are selected and form pages, the formats of the pages will be determined. To ensure that there are fewer distinctive formats for more of the pages, templates are used. View pages are divided into two types: management views and processing views.

The **Management View** is for those web services that need input data. This view provides a table form showing every accessible (with proper states) entry of the GET resources. By providing a management view, users can select the entry to process even if the process is interrupted. For every entry of data, a button named after the function is provided. Once the button is clicked, the view jumps to the process view of the entry and passes over the unique id of the entry for further processing.

The **Processing View** is the process page for the web services to process a data entry. If the post operation in a web service is PUT, a creating view is generated. By retrieving the domains of the PUT resource, the properties to be filled in are obtained and shown as the input boxes in the page. If a GET operation exists in the web service, the instance of the GET resource is obtained according to the unique id passed by the management view and displayed in the input box of the corresponding box. Once the submit button is clicked, the web service is called. For the functions whose next state has multiple branches, a choice view is applied for the user to select the next state.

### E. CLOUD SERVICES DEPLOYMENT AND EXECUTION

After independent cloud services are generated, the data resources are in XML files, the data services and web services are in WADL, and the view pages are in HTML. All of the resource files of the three components are packed in a single independent package. To deploy the generated package on a cloud server, the container tool Docker[6] is applied for quick deployment and multi-tenancy guarantee. By using Dockerfile commands, we define the system image, required dependencies, necessary JDK and run commands in a Dockerfile. Since the package is automated generated, the runtime environment can be fixed. Then, the Dockerfile is added into the service package. With the help of resource sharing tools, such as git and svn, the enterprises can directly pull down and run the package on their cloud server, and at this point, the end users can directly visit the cloud services without any installation or deployment.

When the web service package is executed in the cloud, isolation among the packages is ensured by the Docker in the platform layer and by the generated package in the business layer. First, Docker guarantees the isolation and reliability in physic resource assignment using visualization. Next, our service package provides the isolation and flexible resource management beyond the platform layer by encapsulating all related resources in the file format in the scope of a single package, which is the isolation in the business layer. In this way, we ensure that the web services from packages cannot visit resources in other packages and multiple applications do not interrupt each other in the runtime. Therefore, multiple packages can be deployed in a single server to utilize their advantages. Moreover, different Service Level Agreements (SLAs) [29] can be flexibly applied to packages to control the utilization of resources.

## IV. EXPERIMENTAL ASSESSMENT

In this section, we demonstrate our assessment of CodingBlind. First, case studies are introduced to demonstrate the feasibility of the proposed approach. Then, we design an experiment that invites domain experts to evaluate the correctness and completeness of the systems generated according to the cases.

### A. CASE STUDIES

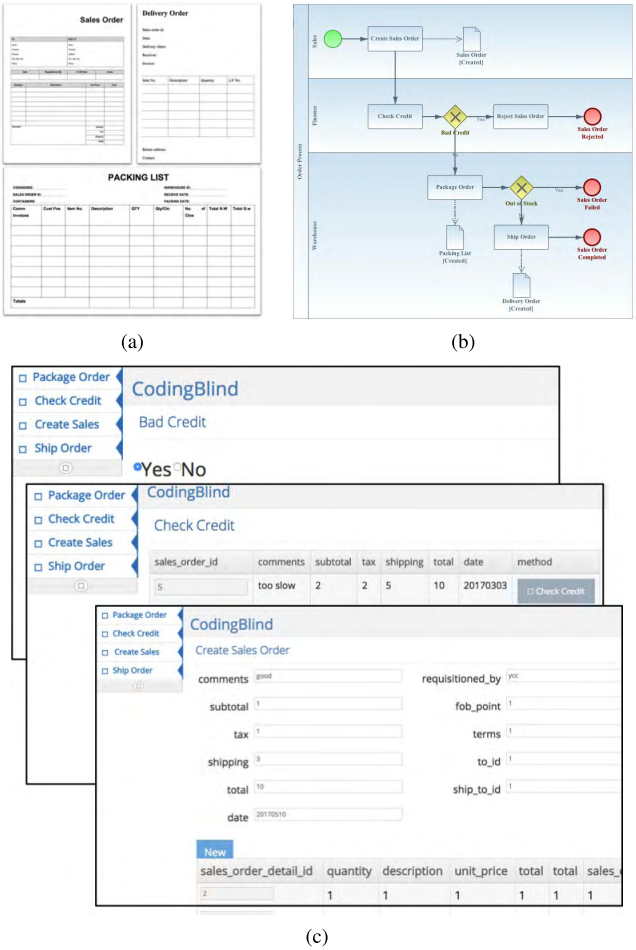
Three categories of real-world processes with various intentions are used to assess CodingBlind. These cases contain different business activities, execution sequences and data, which include all of the basic elements in the BPMNs and various table structures in the real world.

- **Order Process Case Study** describes the process of handling orders in a factory from creating sales orders to shipping until an order is completed. Ten case instances, including a BPMN model and the corresponding printed forms, are provided in this category.
- **Medical Process Case Study** shows the procedure from registering to see a doctor and to finally purchasing medicine. Here, 5 case instances including a BPMN model and the corresponding printed forms are provided.
- **Paper Review Case Study** illustrates how a paper is processed after submission. Only one form is attached to this case, though multiple branches appear in the process. Here, 5 case instances including a BPMN model and the corresponding printed forms are provided.

The BPMN models are created according to the official examples[7] with some simple changes in the names of the tasks. Moreover, we added some data associations. The forms were taken from pictures in the related industries we researched on Google as templates, and they were verified

[6] http://www.docker.com/

[7] http://www.bpmn.org/#tabs-examples




(a) (b) (c)

**FIGURE 11.** An example case and the generation result of CodingBlind. (a) Input printed forms. (b) Input BPMN. (c) Cloud services generated by CodingBlind.

by specialists in the corresponding industries. Based on the above cases, the systems in clouds are generated by CodingBlind. An example input and output is shown in Figure 11.

## B. EVALUATION

### 1) MATRICES

#### a: DATA CORRECTNESS ($D_{cr}$)

For every property in the data resource, *TruePositive* indicates that the property is CORRECTLY transformed from a property term in the printed forms; if it is a foreign key, we verify that it refers to the CORRECT entity. For an entity in the data resources, *TruePositive* denotes that the entity is CORRECTLY transformed from an entity term in the printed forms and that its properties are transformed from the property terms belonging to the corresponding entity term in the printed forms REGARDLESS of the human understanding and spelling mistakes to avoid overlapping between two tallies. Then, the data correctness is defined as follows, where $Correct_P$ and $TruePositive_E$, respectively, denote the number of correct properties and entities and $Total_{DR}$ denotes the total number of terms (of the property or entity) in the generated data resource:

$$D_{cr} = \frac{TruePositive_P + TruePositive_E}{Total_{DR}} \tag{14}$$

#### b: DATA COMPLETENESS ($D_{cm}$)

For every term in the printed forms, *Positive* denotes that it has a corresponding term in the data resources regardless of the correctness of the transformation. $D_{cm}$ denotes the rate of *Positive* terms in the total terms in the printed forms that should be transferred.

$$D_{cm} = \frac{Positive_{PF}}{Total_{PF}} \tag{15}$$

#### c: SYSTEM CORRECTNESS ($Sys_{cr}$)

For each page in a system, 3 criteria are measured: first, whether the DATA entity displayed on the page is the required one; second, whether the FUNCTION of the page satisfies the corresponding task; third, whether the VIEW JUMPING leads to the correct page following the BPMN. Each point contributes 1 point to the score. Therefore, $Sys_{cr}$ is defined as follows, where $Score_p$ denotes the score of a page and $Total_P$ denotes the total number of pages:

$$Sys_{cr} = \frac{\sum Score_p}{3 * Total_P} \tag{16}$$

#### d: SYSTEM COMPLETENESS ($Sys_{cm}$)

For every element (we refer to the tasks, events and divergent exclusive gateways) in the BPMN, if a page is successfully created for it, then it is considered as *Positive*. $Sys_{cm}$ represents the proportion of *Positive* signs in all of the elements.

$$Sys_{cm} = \frac{Positive_{E_{BPMN}}}{Total_{E_{BPMN}}} \tag{17}$$

### 2) PARTICIPANT SELECTION

Measuring the data and the system is a relatively challenging work. We selected 60 volunteers to participate in the experiment to evaluate 20 cases in 3 categories of case studies. Among the 60 participants, 12 are software engineers, and 48 are senior and graduated students majoring in Software Engineering. All of them have taken Software Engineering courses and developed more than one web-based system.

### 3) EXPERIMENT

First, all of the required materials are assigned to the participants. 20 cases are randomly distributed to 60 participants to ensure that every case has 3 evaluators. A piece of material includes the printed forms and BPMN required in the case. In addition, an instrument and a grading form are attached. Then, the participants evaluate the system according to the given materials and record the result on the grading form under supervision. Finally, we collect all of the grading forms and analyze them to produce the corresponding matrices.

### 4) RESULTS

We analyzed all 60 of the grading forms and produced the statistics in Table 2. The first three rows of data in Table 2 are

**TABLE 2. The correctness and completeness of the data and the system of 3 categories of cases.**

| Category | $D_{cr}$ | $D_{cm}$ | $Sys_{cr}$ | $Sys_{cm}$ |
|---|---|---|---|---|
| OrderProcess | 0.95 | 0.99 | 0.95 | 0.97 |
| MedicalProcess | 0.97 | 0.98 | 0.96 | 0.98 |
| PaperReview | 0.98 | 0.89 | 0.95 | 0.98 |
| Average | 0.96 | 0.97 | 0.96 | 0.97 |

the average results of the grading forms related to each case. The final row shows the weighted average results of all of the cases.

### C. THREATS TO VALIDITY

#### 1) THREATS TO INTERNAL VALIDITY

relates to the quality of the inputs in our work. For the data part, the resolution of the form images decides the correctness of the recognition and the following phases. Since OCR is not the core process of the approach, we carefully choose the mature OCR engine to minimize this error. Also, the reasonable design of forms and BPMN is the fundamental of the correctness of the whole process. For example, if different data entities or tasks are entitled to the same name, the correctness of the generated service will decline.

#### 2) THREATS TO EXTERNAL VALIDITY

corresponds to generalizing the result. Our approach is domain-unrelated and can be applied in different disciplines. However, there are some restrictions for the BPMN as mentioned, so we do not assert that we support all variants of BPMN. And also there are some advanced or extensional elements in BPMN are not supported in our approach.

#### 3) THREATS TO CONSTRUCT VALIDITY

relates to the rationality of the experiments. We set 20 cases in 3 unrelated categories as samples to ensure the generalization of the experiment. The participants are of rich experience in web system development. And the grading forms provides no indications. Since there are no existing evaluation matrices, we designed the matrices to evaluate the correctness of the results, which can be a threat to construct validity.

## V. DISCUSSION AND RELATED WORK

The case studies show that CodingBlind is a fully automated cloud service generation method from the printed forms and BPMN model. The results of the evaluation of the experiment show that the generated cloud systems and related data are correct and complete with scores greater than 0.95. There are multiple reasons why our approach obtains high scores: (1) the structure template is abstracted from copious real-world business forms varying from contents to themes, which leads to high correctness and completeness in the data extraction. (2) A full analysis of the BPMN produces high completeness of the data and systems. (3) A semantic model is used to extract meaningful data in the data operation and state extraction, which ensures the correctness of the systems' functions. (4) An FSM is applied to control the logic in the systems, which also contributes to their correctness. However, there are still some limitations to the proposed approach:

- The proposed method provides a structural template to analyze printed forms. The template itself has been widely adopted. However, the data inside each part can be interpreted in various ways according to the semantic meaning, which influences the structure of the property tree and leads to a loss of data correctness.
- As mentioned in Section II-B, the BPMN models used in the method have a set of restrictions, and limited elements are currently supported. Therefore, a prerequisite of BPMN knowledge is required to apply this method. At this time, a system is generated from a single BPMN model. Multiple BPMN combinations are not currently supported.
- The method obtains operations in systems singly from BPMN models. Therefore, only basic functions that involve retrieving, creating, modifying and deleting can be extracted and established. Cases of advanced operations with complex logical interpretations, such as `sorting by` or including a set of operations such as `buying` including sorting, ordering, and paying, are not currently supported.
- There is no automated verification of the approach. To ensure or improve the correctness of the generated models, the users need to verify them manually.

Previous studies have been done in automated cloud services generation. Na-Lumpoon *et al.* [30] provided an approach that reuses the composition and execution of services to obtain new services. They obtained a service list in the Web Service Description Language (WSDL) [8] and composed it to build a BPMN model. Then, they used a BPMN engine to run the new service. Na-Lumpoon's work requires predefined atomic services to generate new services. However, to produce atomic services, enterprises must still depend on third-party services or their own services, which entails a large cost in money, time and manpower. Therefore, their approach is useful for enterprises with basic services who seek to offer new services.

Kwon and Tilevich [31] proposed an automatic approach to migrate existing services to the cloud. Kwon's approach [31] is a helpful tool to programmers and enterprises that already have local services. These authors transform local repositories into JAX-WS applications to produce web services in WSDL. This approach solves the automation problem from a local system to a cloud system. However, for an enterprise using the system but not maintaining it, this approach does not work.

There have also been automated systems that generate local services, where the process is similar to generating cloud services. Basso *et al.* [32] presented a model-driven web engineering approach to automatically design

[8] http://www.w3.org/TR/wsdl20-primer/

**TABLE 3. Comparison with related work.**

| Approach | Input | Feature | Similarity | Service Format | Intended users |
|---|---|---|---|---|---|
| This paper | Printed forms, BPMN | From 0 to 1 | - | WADL | Small and medium-sized enterprises with few programmers |
| Na-Lumppon's [30] | Existing services | 1 to more | BPMN-based controller | WSDL | Enterprises with certain services that hope to extend new services |
| Kwon's [31] | Local services | From local to cloud | - | WSDL | Enterprises with a self-developed local service system |
| Basso's [32] | Textual use cases, paper prototypes | From 0 to 1 (Local) | 3-layer architecture Model-driven approach | - | Start-up companies developing prototype systems |

a prototype information system. They extracted data and function requirements from use cases and paper prototypes and generated conceptual models to drive the system engineering. It is notable that the logic types of their use cases need to be predefined by the users, which is a strong challenge for users without programming experience. Furthermore, manual development is needed in the generation process for certain use cases (which are labeled as manually developed). Therefore, the approach is a semi-automated approach. However, because the development of artificial intelligence is limited to fully automated services, their work is a mature way to automatically design a local system instead of cloud services.

Compared to the previous studies, the innovation of our approach can be seen from three aspects as shown in Table 3. The inputs of our approach are graphic so that it is useful and friendly to users with no programming experience. Moreover, our approach can directly help users to construct a cloud application. But the other studies either need an existing system or service as a foundation or only construct local applications. Therefore our approach contributes to the SMEs with few programmers to quickly obtain their cloud applications.

## VI. CONCLUSION

In this paper, an automated cloud-services-generation process called CodingBlind is proposed. The process takes printed forms and the BPMN model as inputs and ultimately generates an independent service package that can be directly deployed in clouds. The generated cloud service system is modeled by quadruple models and follows the MVC design pattern. The model layer comes from the data services constructed according to the stateful ER model built from printed forms with the assistance of BPMN. The controller layer is composed of web services constructed on the basis of an FSM generated from the BPMN. The view layer is generated on the basis of the data services and web services that are provided in the other layers. Our approach provides an easy way to generate cloud services that is both time-saving and economical. In addition, this approach is a natural solution for the deployment and migration of cloud service systems. Furthermore, the approach gives flexible access for an enterprise to self-define its own cloud services, and the system provides a new way to achieve multi-tenant isolation in the business layer. A prototype tool and a set of experiments show that the proposed technique is feasible, correct and complete for widespread use.

In the future, we would like to improve the completeness of the system by considering the entities of men and the privileges in cloud services. We seek to improve the correctness of the entity identification by introducing linked data into the process. Moreover, we would work on the potential extendability of the system, such as making data services and cloud services plug and play at runtime.

## REFERENCES

[1] S. Tai, J. Nimis, A. Lenk, and M. Klems, "Cloud service engineering," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng. (ICSE)*, vol. 2. New York, NY, USA, May 2010, pp. 475–476.
[2] S. Sotiriadis and N. Bessis, "An inter-cloud bridge system for heterogeneous cloud platforms," *Future Generat. Comput. Syst.*, vol. 54, pp. 180–194, Jan. 2016.
[3] F. Tao, Y. LaiLi, L. Xu, and L. Zhang, "FC-PACO-RM: A parallel method for service composition optimal-selection in cloud manufacturing system," *IEEE Trans. Ind. Informat.*, vol. 9, no. 4, pp. 2023–2033, Nov. 2013.
[4] Y. Lu, X. Xu, and J. Xu, "Development of a hybrid manufacturing cloud," *J. Manuf. Syst.*, vol. 33, no. 4, pp. 551–566, 2014.
[5] Y. Choh, K. Song, Y. Bai, and K. Levy, "Design and implementation of a cloud-based cross-platform mobile health system with HTTP 2.0," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst. Workshops*, Jul. 2013, pp. 392–397.
[6] L. Xu, D. Huang, and W.-T. Tsai, "Cloud-based virtual laboratory for network security education," *IEEE Trans. Edu.*, vol. 57, no. 3, pp. 145–150, Aug. 2014.
[7] G. Laatikainen and A. Ojala, "SaaS architecture and pricing models," in *Proc. IEEE Int. Conf. Services Comput.*, Jun./Jul. 2014, pp. 597–604.
[8] S. Kibe, S. Watanabe, K. Kunishima, R. Adachi, M. Yamagiwa, and M. Uehara, "PaaS on IaaS," in *Proc. IEEE Int. Conf. Adv. Inf. Netw. Appl.*, Mar. 2013, pp. 362–367.
[9] M. Armbrust *et al.*, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, 2010.
[10] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at Google with borg," in *Proc. ACM 10th Eur. Conf. Comput. Syst. (EuroSys)*, New York, NY, USA, 2015, pp. 18:1–18:17.
[11] F. Basile, P. Chiacchio, and J. Coppola, "A hybrid model of complex automated warehouse systems—Part I: Modeling and simulation," *IEEE Trans. Autom. Sci. Eng.*, vol. 9, no. 4, pp. 640–653, Oct. 2012.
[12] T. N. Nguyen, "A unified model for product data management and software configuration management," in *Proc. 21st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Sep. 2006, pp. 269–272.
[13] V. B. R. V. Sagar and S. Abirami, "Conceptual modeling of natural language functional requirements," *J. Syst. Softw.*, vol. 88, pp. 25–41, Feb. 2014.
[14] J. S. Thakur and A. Gupta, "Identifying domain elements from textual specifications," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Aug. 2016, pp. 566–577.

[15] Á. Rebuge and D. R. Ferreira, "Business process analysis in healthcare environments: A methodology based on process mining," *Inf. Syst.*, vol. 37, no. 2, pp. 99–116, 2012.
[16] J. Liu, L. Shen, X. Peng, and W. Zhao, "A consistency detecting approach towards domain requirement and business process," *Small Miro Comput. Syst.*, vol. 34, no. 6, pp. 1270–1275, 2013.
[17] B. Wu, R. Lin, B. Wang, and J. Chen, "Automatic code generation for business process system based on artifact," in *Proc. IEEE Int. Conf. Services Comput.*, Jun./Jul. 2014, pp. 846–847.
[18] S. Meghzili, A. Chaoui, M. Strecker, and E. Kerkouche, "Transformation and validation of BPMN models to Petri nets models using GROOVE," in *Proc. Int. Conf. Adv. Aspects Softw. Eng. (ICAASE)*, Oct. 2016, pp. 22–29.
[19] X. Wang, J. Zhang, and W. Cui, "A process Web client interaction plug-in design and development based on jBPM," in *Proc. Int. Conf. Ind. Control Electron. Eng.*, Aug. 2012, pp. 738–741.
[20] S. Strobl, M. Zoffi, M. Bernhart, and T. Grechenig, "A tiered approach towards an incremental BPEL to BPMN 2.0 migration," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 563–567.
[21] J. Hirayama, H. Shinjo, T. Takahashi, and T. Nagasaki, "Development of template-free form recognition system," in *Proc. IEEE Int. Conf. Document Anal. Recognit. (ICDAR)*, Sep. 2011, pp. 237–241.
[22] R. Song, B. Z. Wu, Y. C. Wang, X. R. Nan, and J. X. Dong, "Research on quality knowledge learning oriented to bearing manufacturing process," *Adv. Mater. Res.*, vol. 215, pp. 159–162, Mar. 2011.
[23] T. Kasar, T. K. Bhowmik, and A. Belaïd, "Table information extraction and structure recognition using query patterns," in *Proc. IEEE 13th Int. Conf. Document Anal. Recognit. (ICDAR)*, Aug. 2015, pp. 1086–1090.
[24] G.-U.-D. Bulbun and H. M. A. Shahzada, "BPMN process model checking using traceability," in *Proc. 6th Int. Conf. Innov. Comput. Technol. (INTECH)*, Aug. 2016, pp. 694–699.
[25] L. Yan and Z. M. Ma, "Formal translation from fuzzy EER model to fuzzy XML model," *Expert Syst. Appl.*, vol. 41, no. 8, pp. 3615–3627, 2014.
[26] M. Aniche, G. Bavota, C. Treude, A. Van Deursen, and M. A. Gerosa, "A validated set of smells in model-view-controller architectures," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Oct. 2016, pp. 233–243.
[27] P. Y. Wong and J. Gibbons, "A process semantics for BPMN," in *Proc. Formal Methods Softw. Eng., Int. Conf. Formal Eng. Methods (ICFEM)*, Kitakyushu-City, Japan, Oct. 2008, pp. 355–374.
[28] G. A. Miller, "WordNet: A lexical database for English," *Commun. ACM*, vol. 38, no. 11, pp. 39–41, 1995.
[29] J. M. García, P. Fernández, C. Pedrinaci, M. Resinas, J. Cardoso, and A. Ruiz-Cortés, "Modeling service level agreements with linked USDL agreement," *IEEE Trans. Serv. Comput.*, vol. 10, no. 1, pp. 52–65, Jan./Feb. 2017.
[30] P. Na-Lumpoon, M.-C. Fauvet, and A. Lbath, "Toward a framework for automated service composition and execution," in *Proc. Int. Conf. Softw., Knowl., Inf. Manage. Appl.*, Dec. 2014, pp. 1–8.
[31] Y.-W. Kwon and E. Tilevich, "Cloud refactoring: Automated transitioning to cloud-based services," *Automated Softw. Eng.*, vol. 21, no. 3, pp. 345–372, 2014.
[32] F. P. Basso, R. M. Pillat, T. C. Oliveira, F. Roos-Frantz, and R. Z. Frantz, "Automated design of multi-layered Web information systems," *J. Syst. Softw.*, vol. 117, pp. 612–637, Jul. 2016.

**HAN YU** received the B.S. degree in software engineering from Shanghai Jiao Tong University, Shanghai, China, in 2016. She is currently a Postgraduate Student at the School of Software, Shanghai Jiao Tong University. Her research interests are in the areas of data recognition and analysis, data integration, Web services, and information systems. She received the Outstanding Thesis Paper Award from the School of Electronic Information and Electrical Engineering, Shanghai Jiao Tong University, in 2016.

**CONGCONG YE** received the B.S. degree in software engineering from Central South University in 2016. She received excellent grades and was recommended to Shanghai Jiao Tong University. She is currently a Postgraduate Student at the School of Software, Shanghai Jiao Tong University. Her research interests are in the areas of data mining, blockchains, and information systems.

**HONGMING CAI** (M'08–SM'15) received the B.S., M.S., and Ph.D. degrees from Northwestern Polytechnical University, China, in 1996, 1999, and 2002, respectively. He is currently a Professor at the School of Software, Shanghai Jiao Tong University, China. He received a reward for being a National Outstanding Scientific and Technological Worker by the China Association for Science and Technology in 2012. He is the Standing Director of the China Graphics Society, a Senior Member of ACM, and a Senior Member of the China Computer Federation.

**LIHONG JIANG** received the B.S., M.S., and Ph.D. degrees from Tianjin University, China, in 1989, 1992, and 1996, respectively. During 1992–1993, she was an Assistant Professor at the Department of Computer Science, Qingdao Ocean University, China. During 1996–1998, she was a Postdoctoral Research Fellow at the School of Management, Fudan University, China. She is currently an Associate Professor at the Software School, Shanghai Jiao Tong University, China.

**CHENG XIE** received the B.S. degree from Minzu University, Beijing, China, in 2009 and the M.S. and Ph.D. degrees from Shanghai Jiaotong University, Shanghai, China, in 2012 and 2017, respectively. During 2015–2016, he was a Visiting Scholar in the Data and Web Science Group, University of Mannheim, Germany. The Visiting Scholarship was appointed and sponsored by Deutscher Akademischer Austausch Dienst and Shanghai Jiao Tong University. He is currently a Lecturer at the School of Software, Yunnan University, China. His research interests include semantic technologies, linked open data, Web of ontology, and data science.

**BOYI XU** was born in Yantai, China in 1966. He received the B.S. degree in industrial automation and the Ph.D. degree in management science from Tianjin University, Tianjin, China, in 1987 and 1996, respectively. He is currently an Associate Professor at the College of Economics and Management, Shanghai Jiao Tong University, China. His research interests include enterprise information systems, electronic commerce, and business intelligence.