



# Business Objects - A New Business Process Modeling Approach

Xiaohui Shi, Jia Lin, Gaojia Hu, and Yang Yu (✉)

School of Data and Computer Science, Sun Yat-sen University,  
Guangzhou, China

{shixhui, linj39, hugj3}@mail2.sysu.edu.cn,  
yuy@mail.sysu.edu.cn

**Abstract.** The traditional process-centric modeling approach often focuses on tasks with a fixed execution order. This modeling approach is difficult to describe business processes in which the number of steps and cases is varied depending on the case state and determined dynamically at run time. Motivated by modeling requirements from complex collaborative business scenarios in the internet, such as crowdsourcing, a concept of Business Object (BO) and BO-oriented Process Modeling (BOOPM) approach are proposed. We introduce the role element into standard Harel statechart to explicitly describe the life cycle of a BO whose state changes as tasks processed by different business roles. We also extend event communication mechanism into the statechart to support collaboration among multiple BOs to achieve a business goal. Meanwhile, the BO provides a suitable foundation to control the granularity of the process model properly. After that, the ability of BOOWM for process specification is evaluated based on a set of workflow patterns. Finally, we illustrate BOOPM approach using a crowdsourcing example and realize a prototype system to support the execution of a business process modeling by BOOPM approach.

**Keywords:** Business Object · Process modeling · Statechart · Crowdsourcing

## 1 Introduction

A workflow modeling approach allows for explicit and intuitive representation of business processes. Traditional process-centric modeling approaches focus on the control-flow of an entire process, i.e., which tasks need to be executed and in what order, but ignore the data context. With regard to complex business processes that may deal with different data entities in different states, the process-centric modeling approach squeezes the control flows of different data entities into one process definition and result in a large flattened process model with high complexity. Meanwhile, without data context, the process-centric approach is different to capture dynamic business processes since there exist tasks depend on data which is not known until runtime.

These problems were discussed in the literature for some time leading to many proposals in which data context is the central focus (e.g. [10–13]). However, to a large extent, these proposals are extensions of process-centric modeling approach such that a process model is composed of control-flow and data-flow dependencies. We have never

seen a related research of data-centric modeling approach to model dynamic non-deterministic processes.

Crowdsourcing is a powerful mechanism to complete tasks on the Internet which is widely applied in real world recently [1, 2]. Compared with typical micro-tasks on the Amazon Mechanical Turk, tasks in life are very complex and take a lot of time to complete. They need technicians from different fields to work collaboratively. To make this happens, complex tasks need to be decomposed into sub tasks. The process of decomposition is very complex such that when and how should it be decomposed is fully decided and executed by crowd workers. The key features of this process are: (1) it is a complex and dynamic recursive process which involves collaboration among workers; (2) the number and execution order of tasks are not known until runtime. Any attempt to model this process using existing modeling approaches is usually met with resistance by these features.

Faced with these problems, we propose the concept of Business Object (BO) and BO-oriented Process Modeling (BOOPM) approach inspired by the concept of communicating finite-state machines [3]. With the help of the BOOPM approach, these dynamic collaborative business processes can be modeled flexibly and the granularity of a process can be controlled properly.

The rest of this paper is organized as follows. Related works are discussed in Sect. 2. Section 3 introduces the BOOPM framework. We then make an analysis of the BOOPM approach in terms of workflow patterns in Sect. 4. In Sect. 5 an example of crowdsourcing process is introduced to illustrate the improved expressive power and flexibility of the BOOPM approach. The conclusion and future work are provided at the end.

## 2 Related Work

Most existing notations and languages for business process modeling are based upon a process-centric modeling approach (e.g., [4–7]). These process-centric modeling approaches often focus on the control flow of the whole process to indicate the execution order of a set of activities, but ignore the data context [10]. Without data context, it is difficult to describe dynamic nondeterministic processes in which the number of tasks need to be executed depend on runtime factors.

Later, the object-centric approaches for modeling business processes have become an area of growing interest to support flexible workflow. The object-centric modeling approach decomposes a process into a hierarchy of modules. [8] proposes proclets for workflow specification that can support complex processes. The proclet is kind of sub-process based on Petri-net with no explicit emphasis on data flows. [9] proposes Object Coordinate Net (OCoN) for workflow specification. The OCoN uses variants of Petri-Nets to describe the different aspects of systems and interfaces in an object-oriented style in order to provide a seamless integration with UML structure diagrams. However, objects in these proposals are kind of sub process objects such that the backbone of process models are still activities and their control flows.

Recently, the data-centric process modeling approaches are proposed. The case handling modeling approach [10] addresses many problems of traditional workflow

management systems especially the handling of process context. [11] introduces the document-driven workflow system in which tasks are described using input and output documents. However, the issue of proper granularity for a business process is not actually addressed in these approaches.

Another popular data-centric process modeling approach is the artifact-centric approach. [12] develops an artifact-centric formal modeling framework that integrates both control flow and data flow. [13] presents nine operational patterns and develop a computational model for operational models based on Petri Nets. [14] studies artifacts with Guard-Stage-Milestone (GSM) lifecycles in which artifact interactions are based on conditions and events. GSM lifecycles are more declarative than the finite state machine. Business artifacts substantially store data information and state information pertinent to a business entity. In contract, BOOPM provides a high-level encapsulation for data entities that are involved in a business process based on object-oriented technique, so that more complex and dynamic processes can be supported. Meanwhile, BOOPM describes states and behaviors of a BO by means of standard Harel statechart, which is more intuitive and easy to be understood.

Another thread of related work is use of state machine to workflow specifications in the industry. Windows Workflow Foundation framework (Microsoft) and OSWorkflow engine (Open source) support using state machine to establish workflow model of event-driven business processes [15, 16]. However, they do not support communication among multiple statecharts. BOOPM extends communication mechanism of statechart.

### 3 Business Object-Oriented Process Modeling

The key abstraction of the BOOPM framework is that of BOs which are abstracted at the modeling level as BO types. BOs are business entities involved in the business process. Each BO encapsulates business data, tasks and object life cycle. A task corresponds to a human or automated operation being executed upon one certain BO. The object life cycle explicitly describes state changes of a BO as tasks processed by different business roles. The object life cycle is described using a standard Harel statechart [17] in which state transitions follow the Event-Condition-Action paradigm. Thus, the evolution of object life cycles of all BOs in the process from the initial state to the final state describes the processing of business.

#### 3.1 Formal Model of BOOPM

In this section, formal definitions of BOOPM framework are introduced.

**Definition 1 (BO type).** A BO type abstracts a group of BOs with their business data, tasks and object life cycle. A BO type  $T$  is a tuple  $(C, A, \Sigma, M)$  where,

- $C$  is a unique BO type name;
- $A = \{a_1, a_2, \dots, a_x\}$ ,  $a_i \in A$  is an attribute with a string or real number value or an undefined value;
- $\Sigma$  is a set of tasks that are associated with  $T$ ;
- $M$  is the object life cycle of  $T$ .

**Definition 2 (Task).** A task is tuple  $(n, t, \rho, m, E)$  where,

- $n$  is the name of the task;
- $t$  is the type of the task consisting of manual, automatic or semi-automatic type;
- $\rho$  is the business role to execute the task;
- $m$  is the application or the tool used by the role to execute the task;
- $E = \{e_1, e_2, \dots, e_x\}$ ,  $e_i \in E$  is a tuple  $(e, p)$  where  $e$  is the event name and  $p$  is parameters and  $e_i$  may be generated according to the result of the task.

**Definition 3 (Object Life Cycle).** An object life cycle is a tuple  $(S, st, F, \rho, \phi, V, E, C, A, L, T)$  where,

- $S$  is a finite set of states, and  $st \in S$  is the initial state,  $F \in S$  is the final state;
- $\rho: S \rightarrow 2^S$  is the hierarchical relationship of states,  $\rho(s)$  represents all direct children of composite  $s$ , and if  $s$  is a basic state, then  $\rho(s) = \emptyset$ ;
- $\phi: S \rightarrow \{XOR, AND\}$  is the function to define types for each composite state, for XOR-type, a composite state is made up of a set of mutually exclusive sub states, for AND-type, a composite state is made up of a set of parallel sub states,
- $V$  is a set of expressions. If  $p$  is a number, then  $p \in V$ ; Assume  $V_p \subseteq A$  is a set of basic variables; if  $v \in V_p$ , then  $v \in V$ ; if  $v_1 \in V_p$ ,  $v_2 \in V_p$ ,  $op \in \{+, -, *, /, \%\}$ , then  $v_1 \text{ op } v_2 \in V$ ;
- $E = \{e_1, e_2, \dots, e_x\}$ ,  $e_i \in E$  is a tuple  $(e, p, m, t, s)$  where  $e$  is the event name,  $p$  is parameters of  $e_i$ ,  $m$  is the propagation mode of  $e_i$  with a string value selected in the set  $\{\text{Broadcast, Multicast, Unicast}\}$ ,  $t$  is the target BO type that  $e_i$  is passed to and  $s$  is the current state of the target BO type;
- $C$  is a finite set of Boolean conditions.  $T(\text{true})$  and  $F(\text{false}) \in C$ . Assume  $C_p$  is a set of basic conditions; If  $c \in C_p$ , then  $c \in C$ ; If  $c_1 \in C_p$ ,  $c_2 \in C_p$ ,  $op \in \{\wedge, \vee, \sim\}$ , then  $c_1 \text{ op } c_2 \in C$ ; if  $c_1 \in V_p$ ,  $c_2 \in V_p$ ,  $op \in \{=, <, >, \neq, \leq, \geq\}$ , then  $c_1 \text{ op } c_2 \in C$ ;
- $A$  is a finite set of actions. An action is associated with state transition or entry action of a state. An action may be invoking a task (see Definition 2), creating new instance of a BO type, sending an event to other BOs;
- $L \subseteq E * C * A$  is a set of labels associated with transitions and written as  $e[c]/a$ , where  $e \in E$ ,  $c \in C$ ,  $a \in A$ ;
- $T \subseteq 2^S * 1 * 2^S$  is a set of transitions. For  $t = (x, l, y)$ ,  $x$  is the source state and  $y$  is the target state,  $l \in L$  is the label associated with the transition from  $x$  to  $y$ . If the event  $e$  is occurred when the source state is enabled, then the condition  $c$  is checked, if  $c$  is true the transition will be triggered and the action  $a$  will be taken in response to  $e$ .

### 3.2 Relationship Between BOs

Different from objects in the OO programming, there are three kinds of relationships between BOs. They are creation relationship, association relationship and composition relationship.

For creation relationship, it means a BO type T1 creates an instance of a BO type T2 and then they go on the processing of their own object life cycle respectively. In general, the creation relationship is contained in other relationships, such as the association relationship and the composition relationship.

For association relationship, it means a BO type T1 creates an instance of another BO type T2 when the business is well beyond the scope of the processing ability of T1. T1 transfers context and data to T2 and waits for the processing result of T2 in some state. When T2 sends an event to T1 to indicate the processing result of T2, T1 triggers some transition according to this event.

For composition relationship, it means a BO type T1 creates definite quantity instances of a BO type T2. Each instance of T2 must send a finish event to T1 when its object life cycle is completed. When the object life cycle of T1 is completed, all instances of T2 must be completed too. The composition relationship is a special case of the association relationship.

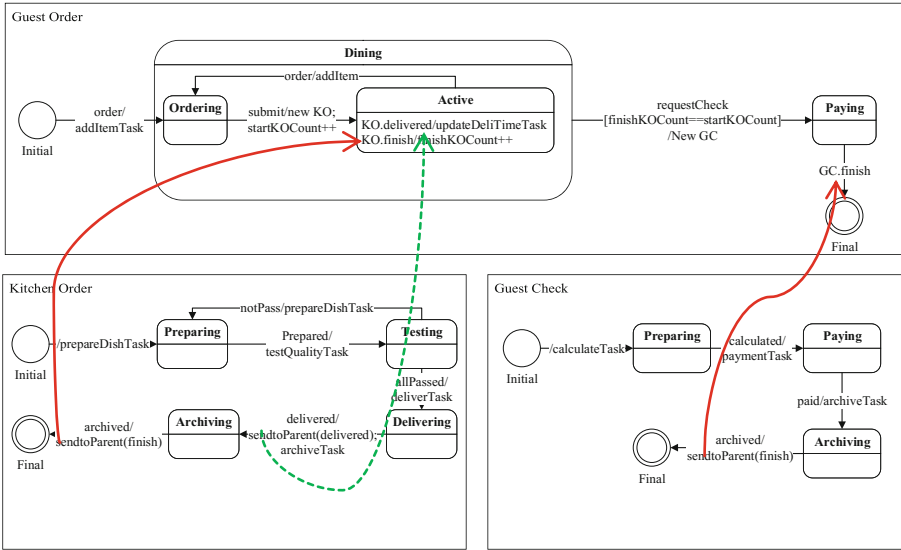
Hence the definition of BOO process model is given below.

**Definition 4 (BO-Oriented Process Model or BOOP model).** Let  $\Pi$  denote an BOOP model, and it is tuple  $(N, \lambda, \beta)$  where  $N$  is a set of BO types,  $\lambda: N \rightarrow 2^N$  is the association relationship between BO types, where  $\lambda(n)$  represents all BOs that  $n$  associates with during the processing of its life cycle.  $\beta: N \rightarrow 2^N$  is the composition relationship between BO types, where  $\beta(n)$  represents all BOs that  $n$  composes during the processing of its life cycle.

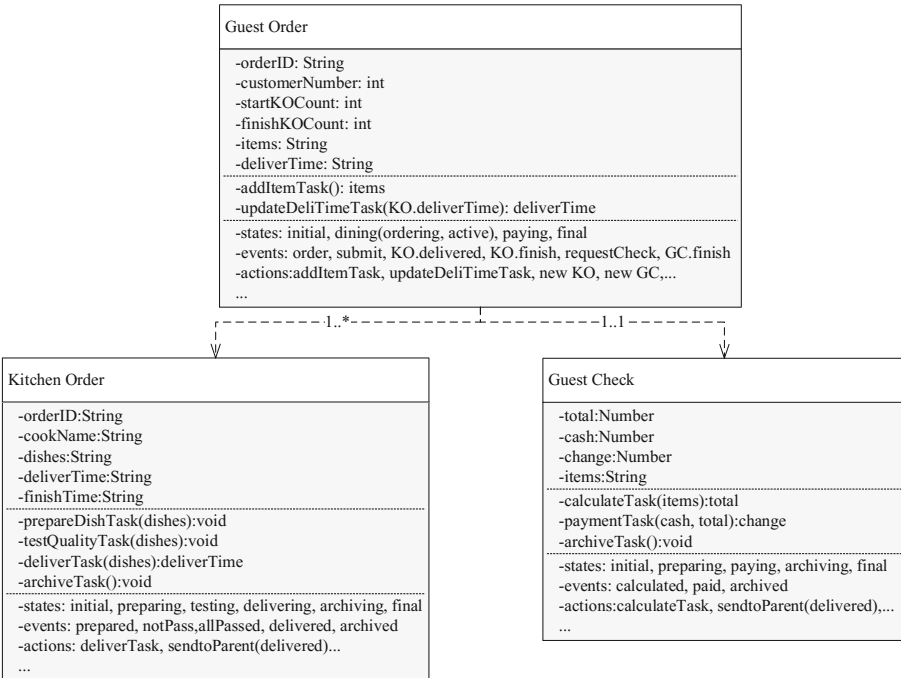
**Example 3.1.** We use a restaurant business process as an example to illustrate BOOPM approach which is described as an artifact-centric process model in [18]. BOs in this business include Guest Order (GO), Kitchen Order (KO) and Guest Check (GC). Figure 1 shows object life cycles for these three BO types. GO and KO maintain a one-to-many association relationship which means the customer can order several times over a meal time. GO and GC maintain a one-to-one association relationship which means one GO only owns one GC. Therefore Fig. 2 shows the business class diagram for the process model of the restaurant business process.

We give the xml schema of the BOO process model based on the formal model of BOOPM. The xml schema is the extension of W3C SCXML which is a generic state-machine based execution environment based on CCXML and Harel State Tables [19]. The xml schema in this paper extends the executable content of SCXML, namely action in the ECA paradigm. The extension labels are `<newbo>`, `<call>` and `<send>`. For detail, see <https://github.com/rinkako/RenWFMS/tree/master/XML>, the complete description of the BOO process model for Fig. 1.

Meanwhile, we developed a prototype system which can support reading and executing of the xml process model described using the xml schema [23]. The prototype system is based on Apache Common SCXML engine which is a java implementation of W3C SCXML standard [20].



**Fig. 1.** Object life cycles for BOs in the restaurant business process. (Color figure online)



**Fig. 2.** The business class diagram for the BOO process model of restaurant business process.

### 3.3 Communication of Business Objects

A BOOP model is made up of a set of connected BOs. Two BOs are connected if one is associated with another. Such two BOs maintain a supervision relationship during the processing of the business, which means the parent BO supervises the life cycle of the sub BO. When the life cycle of a sub BO is finished, it must report to the parent BO through a finish event.

The BO which is created at the beginning of the business process is the root BO. New BOs are created during the processing of the root BO. BOs supervised by the root BO and other BOs supervised by these supervised BOs in turn gradually form a BO instance tree structure in runtime. Each node in the tree is an instance of a BO type.

BOs in the tree collaborate with each other by means of event communication to achieve a business goal. The collaboration among BOs consists of synchronization pattern and asynchronization pattern. In the synchronization pattern, the parent BO must wait for finish events from all its sub BOs before taking the transition to the final state. In the asynchronization pattern, it means that processing of BOs are independently. The event between BOs is transmitted asynchronously, i.e., without waiting for a reply.

We extend the event propagation mode of the <send> label of SCXML to support event communication among multiple BOs. The core of collaboration is to determine the destination of the communication event according to the position of the current BO in the instance tree. Therefore the following extended attributes are necessary.

1. eventMode: the propagation mode of an event. Its value can be:
  - (a) Broadcast: The event will be sent to each BO in the tree;
  - (b) Multicast: The event is sent to a group of BOs according to the position of the current BO in the instance tree, such as to an arbitrarily specified BOs set, or directly to its offspring, children, siblings, ancestor;
  - (c) Unicast: The event is sent to a specified BO according to the identifier of the target BO, or directly to its parent.
2. targetType: This attribute shows that an event should be passed to what BO type in the tree.
3. targetState: Since state of the BO changes with time. This attribute indicates that an event should be sent to a BO whose current state equals to the value of the targetState attribute.

With the extensions above, the instance tree representing the relationship between BOs can be constructed in runtime and these BOs can communicate with each other to achieve a business goal.

**Example 3.2.** As shown in Fig. 1, the green dotted line means it is an asynchronization pattern. The red line means it is an synchronization pattern. When the business starts, the instance of GO is created firstly. When the transition from Ordering to Active is triggered, an instance of KO is created and it begins its life cycle. When GO is in the Dining state, the customer can order arbitrary times and thus corresponding number of KO instances are created. When the life cycle of a KO instance comes to the end, it must send a finish event to its parent. When the customer requests to pay the order, the

value of startKOCCount and finishKOCCount should be checked whether they are equal or not. If they are equal, GO triggers the transition to the Paying state and an instance of GC is created. When the instance of GC finishes its life cycle, it sends a finish event to its parent and GO triggers the transition to the Final state. Thus the business goal is achieved when all life cycles of BO instances come to the end.

## 4 Evaluation of BOOPM

This section provides evaluation of the expressiveness of BOOWM in terms of workflow patterns [21]. We represent each pattern in BOOWM approach in [22]. While for the limited space, only several common workflow patterns are discussed here. Meanwhile, this section provides several process patterns that are supported specifically in the BOOPM approach.

### 4.1 Evaluation of a Single BO

#### WP1: Multi-Choice and Structured Synchronizing Merge

The Multi-Choice pattern allows one or more parallel branches to be selected based on a selection mechanism. The Structured Synchronizing Merge pattern is used to merge parallel branches selected in the Multi-Choice pattern. The modeling challenge is that the number of parallel branches need to be merged is not known at the design time. It is totally depend on the Multi-Choice pattern.

The BOOPM approach describes the Multi-Choice pattern and the Structured Synchronizing Merge pattern as shown in Fig. 3. The Null state behind the initial pseudo-state in each parallel branch means no task is executed. One or both of the following transitions in parallel branches can be triggered depend on the evaluation of the conditions associated with each of them. There are two variables need to pay attention to. The variable sCount will be increased by one when one transition from Null to the following state is triggered. The variable fCount will be increased by one when one parallel branch is completed. Synchronization of parallel branches is executed when values of two variables become equal. Then the transition from MC to StateB is triggered. Thus, as shown in Fig. 3, any branches can be selected and synchronized.

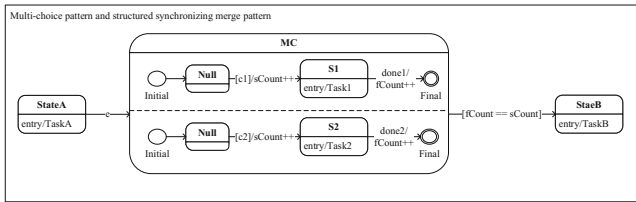


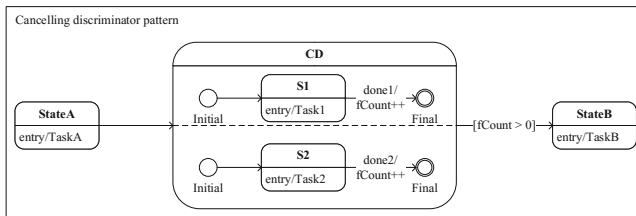
Fig. 3. Multi-choice pattern and structured synchronizing merge pattern.



## WP2: Cancelling Discriminator

The Cancelling Discriminator pattern differs from the Structured Synchronization pattern in that the subsequent process will continue once one parallel branch has been completed. All remaining branches are cancelled as a result.

The BOOPM approach describes behaviors of the Cancelling Discriminator as shown in Fig. 4. When entering CD, State1 and State2 will be entered simultaneously. Task1 and Task2 are executed in any possible order. If Task1 is completed first, it will generate an event done1. The first branch moves to the Final state with the variable fCount equals to one. Then the transition from CD to StateB is triggered since the exit condition of the concurrent state becomes true. The remaining task in State2 is cancelled.



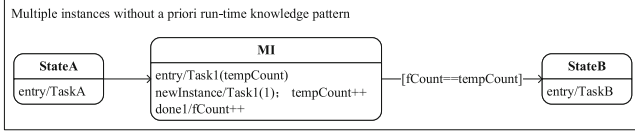
**Fig. 4.** Cancelling discriminator pattern.

This series of patterns describe multiple instances of the same task are created in a business process based on different situations, which are not included in [21].

## WP3: Multiple Instances Without a Priori Run-Time Knowledge

This pattern differs from other multiple instance patterns in that the required number of instances cannot be determined until the last instance of that task is completed. At any time, whilst multiple instances are running, it is possible for additional instances of the same task to be created.

The BOOPM approach utilizes the combination of external events and internal transitions to implement this pattern. As shown in Fig. 5, when entering MI, tempCount instances of Task1 are created such that the value of tempCount is determined by runtime factors. The BO stays in MI until the completions of these task instances. While in MI, if more instances are needed, an event named newInstance will be sent to the BO from external environment and an internal transition is triggered. The action of this transition is creating and dispatching an additional instance of Task1 as well as updating the value of the variable tempCount. Each time when a task instance is completed, an event named done1 is generated and another internal transition is triggered to increase the variable fCount by one. The guard condition associated with the transition from MI to StateB will be true only when all created instances are completed which means the value of fCount equals to tempCount.



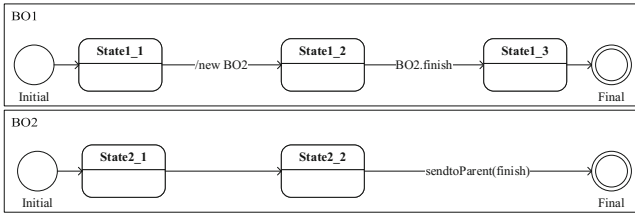
**Fig. 5.** Multiple instances without a priori run-time knowledge pattern.

## 4.2 Evaluation of Communicating BOs

We discover four more specific patterns that BOOPM supports due to event communication mechanism among multiple BOs, including nested pattern, similar synchronization pattern, simultaneous trigger pattern, and cyclic synchronization pattern. For simplicity, the business data and tasks are omitted and the following patterns focus on the event communication among BOs.

### SP1: Nested Pattern

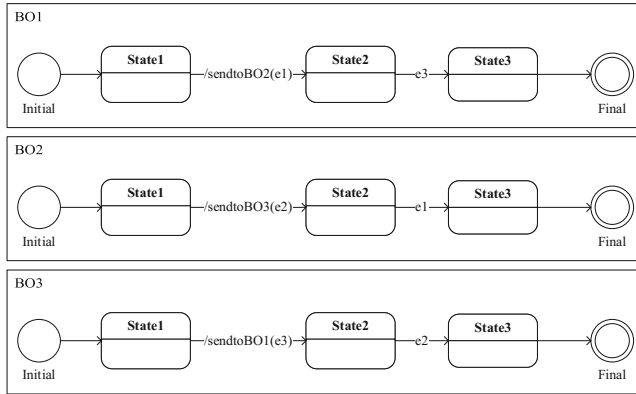
The Nested pattern describes a dependency relationship between two BOs. The parent BO creates a sub BO during processing of its life cycle and waits for the completion of that sub BO. As shown in Fig. 6, BO1 creates an instance of BO2 when triggering the transition from State1 to State2. A new instance of BO2 with a unique identifier starts its life cycle from the Initial state. BO1 must wait for the finish event sent from BO2 before exiting State1\_2.



**Fig. 6.** Nested pattern.

### SP2: Similar Synchronization Pattern

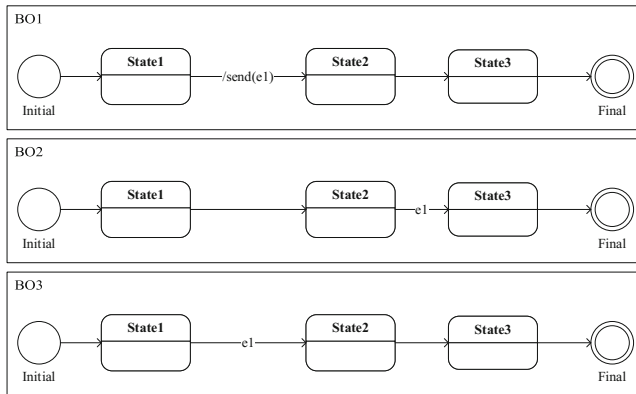
The Similar Synchronization pattern describes a constraint relationship of a set of BOs that each BO waits for an event sent from another BO in the set. As shown in Fig. 7, there are three BOs run concurrently and each of them sends an event when triggering the transition from State1 to State2. In State2, they all wait for the event sent from another BO.



**Fig. 7.** Similar synchronization pattern.

### SP3: Simultaneous Trigger Pattern

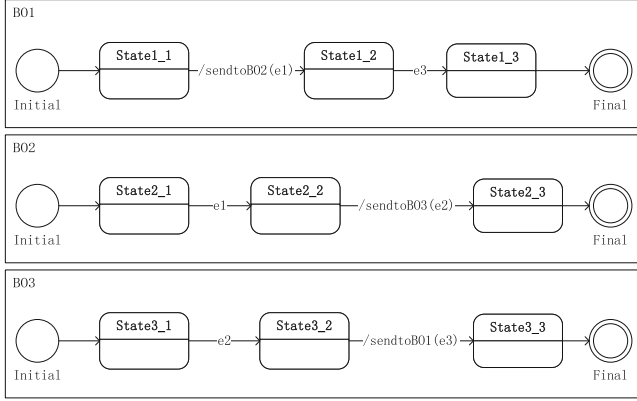
The Simultaneous Trigger pattern describes a situation such that two or more BOs are waiting for the same “wake-up” event from the same BO. If these BOs receive such a “wake-up” event, the corresponding transition is triggered in each BO concurrently. As shown in Fig. 8, BO2 and BO3 both wait for the event e1 sent from BO1.



**Fig. 8.** Simultaneous trigger pattern.

### SP4: Circular Synchronization Pattern

The Circular Synchronization pattern describes the circular dependency constraint among multiple BOs. There must be an entrance to enter the cycle and an exit to leave the cycle. As shown in Fig. 9, there is a Circular Synchronization pattern among BO1, BO2 and BO3. The transition from State1\_1 to State1\_2 is the entrance of the cycle such that BO1 sends e1 to BO2. The exit of the cycle is BO3 sends e3 to BO1 to trigger the transition from State1\_2 to State1\_3 to exit the cycle.



**Fig. 9.** Cyclic synchronization pattern.

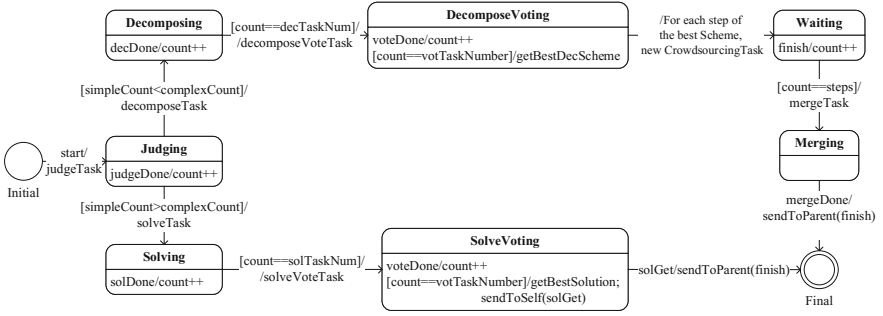
Overall, The BOOPM approach supports thirty six workflow patterns. Only a small set of workflow patterns cannot be supported due to the inherent semantics of statechart and the requirements of reasonability.

The semantic of the concurrent state is that it waits for the completion of all the parallel branches before triggering the transition to the next state. If there are additional guard conditions to force exiting the concurrent state, all the remaining uncompleted parallel branches must be cancelled and exited. As a result, the Multi-Merge pattern is not supported, in which the completion of each parallel branch will instantiate the tasks in the subsequent branch one time. Also, there is no support for Structured Discriminator pattern, Blocking Discriminator pattern, Structured Partial Join pattern and Blocking Partial Join pattern. Meanwhile, the life cycle of a BO must have a single Final state indicating the end of the life cycle. Semantically any state must have one path to that Final state. Hence a BOO process model allows no implicit termination states which means there are no transitions leaving from that state. Therefore, the General Synchronizing Merge pattern and Implicit Termination pattern are not supported.

However, the introduction of flexible event communication mechanism indeed enriches the expressiveness of BOOPM in the description of collaboration among multiple BOs. Thus, in addition to those thirty six workflow patterns, the BOOPM also supports four specific patterns. Other event-based models like BPMN, cannot support concurrency and uncertain number of sub processes.

## 5 Case Study

The ability of BOOPM approach to model control flows is shown in the previous section (see Fig. 1). In this section, we illustrate BOOPM approach using the crowd-sourcing example to illustrate the ability of BOOPM approach to model dynamic nondeterministic processes in which the number of steps and cases is determined dynamically at run time.

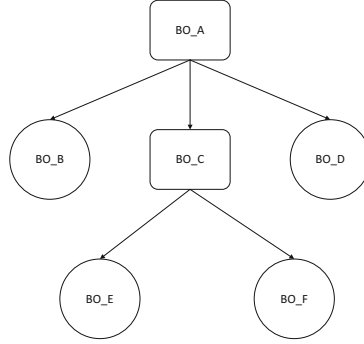


**Fig. 10.** The object life cycle of CrowdsourcingTask type.

Consider for example the crowdsourcing process of writing an introduction paper about crowdsourcing research. The goal of this process is workers collaborate with each other in the network to decompose the original complex task into multiple sub tasks, and then decompose these subtasks individually. Finally when all subtasks are simple tasks, solve these simple tasks and merge results from bottom up to obtain a complete paper. This is a dynamic and recursive process that contains uncertain number of recursive decomposition steps. Such unplanned situation is particularly challenging to capture using existing process modeling approaches since the best decomposition scheme of each task is determined by workers during runtime.

In the crowdsourcing process, the crowdsourcing task and its sub tasks are discovered BOs. They own the same BO type. Assume the name of the BO type is CrowdsourcingTask. CrowdsourcingTask stores business data related to the business and contains a set of tasks act upon it to deal with business logic. The object life cycle of CrowdsourcingTask specifies the possible ways an instance of this type might progress through the business. As shown in Fig. 10, tasks are mapped to actions associated with entry labels or transition labels. The internal transitions below entry labels and external state transitions follow the event-condition-action paradigm such that respond events from the environment.

When a requester posts a crowdsourcing task, an instance of CrowdsourcingTask type is created, for example named BO\_A. When the transition from the DecomposeVoting state to the Waiting state is triggered, sub BOs of BO\_A are created according to steps in the best decompose scheme such that the best decompose scheme is obtained automatically according to the voting results of crowd workers, for example what is crowdsourcing (named BO\_B), the related work of crowdsourcing (named BO\_C), and the future work of crowdsourcing (named BO\_D). BO\_A stays in state Waiting for the completion of these three sub BOs. Each of these sub BOs may be judged to be complex and then decomposed into several sub BOs, or judged to be simple and solved directly by workers. For example, for the second task, one possible decompose scheme may be search related work in recent five years (named BO\_E) and search related work in first five years (named BO\_F). Finally, the runtime BO instance tree structure of the crowdsourcing process is generated as shown in Fig. 11. Note that it is just one possible decomposition scheme of the crowdsourcing task.



**Fig. 11.** One possible BO instance tree structure of the crowdsourcing process.

All these BOs in the instance tree run individually. When a node (e.g. BO\_E) moves to the final state, which means this simple sub task gets the result, it sends a finish event to its parent (e.g. BO\_C), and then that parent BO checks whether all its sub BOs (e.g. BO\_E and BO\_F) have sent the finish event to it. If so, BO\_C moves to the Merging state and executes mergeTask which is an automatic task to merge the results of these sub BOs. When the mergeTask is completed, BO\_C sends a finish event up to its parent (BO\_A) and moves to the Final state. As this process goes on, the root BO (BO\_A) moves to the Merging state when it receives all finish events from sub BOs. After mergeTask of BO\_A completes, it moves to the Final state which means the crowdsourcing process completes and the requester gets the final result, an introduction paper about crowdsourcing.

## 6 Conclusion and Future Work

In this paper, we present a BOOPM approach. The BOOPM approach not only has the ability to model traditional control flows, but also has the ability to model dynamic processes. Meanwhile, the BO provides a foundation to control the proper granularity of a process model.

We have discovered three relations among BOs and they are enough for the current research. While the join of human factors, new relations are worth to explore. We expect our future work to discover more potential relations among BOs. Our future work also includes extending the business operation to support processes. The process is described in the process-centric modeling approach.

**Acknowledgements.** This work is Supported by the National Key Research and Development Program of China under Grant No. 2017YFB0202200; the National Natural Science Foundation of China under Grant No. 61572539; the Research Foundation of Science and Technology Major Project in Guangdong Province under Grant Nos. 2015B010106007, 2016B010110003; the Research Foundation of Science and Technology Plan Project in Guangdong Province under Grant No. 2016B050502006; the Research Foundation of Science and Technology Plan Project in Guangzhou City under Grant No. 2016201604030001.

## References

1. Yuen, M.C., King, I., Leung, K.S.: A survey of crowdsourcing systems. In: Privacy, Security, Risk and Trust and 3th IEEE International Conference on Social Computing, pp. 766–773. IEEE Press, Boston (2011)
2. Kulkarni, A., Can, M., Hartmann, B.: Collaboratively crowdsourcing workflows with turkomatic. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, pp. 1003–1012. ACM Press, Seattle (2012)
3. Brand, D., Zafriopulo, P.: On communicating finite-state machines. *J. ACM* **30**(2), 323–342 (1983)
4. Van der Aalst, W.M., Ter Hofstede, A.H.: YAWL: yet another workflow language. *Inf. Syst.* **30**(4), 245–275 (2005)
5. Business Process Modelling Notation. <http://www.bpmn.org>. Accessed 12 May 2017
6. Nüttgens, M., Fold, T., Zimmermann, V.: Business process modeling with EPC and UML: transformation or integration? In: Schader, M., Korthaus, A. (eds.) *The Unified Modeling Language—Technical Aspects and Applications*, pp. 250–261. Physica-Verlag, Heidelberg (1998)
7. Dumas, M., ter Hofstede, Arthur H.M.: UML activity diagrams as a workflow specification language. In: Gogolla, M., Kobryn, C. (eds.) *UML 2001*. LNCS, vol. 2185, pp. 76–90. Springer, Heidelberg (2001). [https://doi.org/10.1007/3-540-45441-1\\_7](https://doi.org/10.1007/3-540-45441-1_7)
8. van der Aalst, W.M.P., Barthelmeß, P., Ellis, C.A., Wainer, J.: Workflow modeling using proclets. In: Scheuermann, P., Etzion, O. (eds.) *CoopIS 2000*. LNCS, vol. 1901, pp. 198–209. Springer, Heidelberg (2000). [https://doi.org/10.1007/10722620\\_20](https://doi.org/10.1007/10722620_20)
9. Wirtz, G., Weske, M., Giese, H.: The OCoN approach to workflow modeling in object-oriented systems. *Inf. Syst. Frontiers* **3**(3), 357–376 (2001)
10. Van der Aalst, W.M., Weske, M., Grünbauer, D.: Case handling: a new paradigm for business process support. *Data Knowl. Eng.* **53**(2), 129–162 (2005)
11. Wang, J., Kumar, A.: A framework for document-driven workflow systems. In: van der Aalst, W.M.P., Benatallah, B., Casati, F., Curbera, F. (eds.) *BPM 2005*. LNCS, vol. 3649, pp. 285–301. Springer, Heidelberg (2005). [https://doi.org/10.1007/11538394\\_19](https://doi.org/10.1007/11538394_19)
12. Bhattacharya, K., Gerede, C., Hull, R., Liu, R., Su, J.: Towards formal analysis of artifact-centric business process models. In: Alonso, G., Dadam, P., Rosemann, M. (eds.) *BPM 2007*. LNCS, vol. 4714, pp. 288–304. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-75183-0\\_21](https://doi.org/10.1007/978-3-540-75183-0_21)
13. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling business contexture and behavior using business artifacts. In: Krogstie, J., Opdahl, A., Sindre, G. (eds.) *CAiSE 2007*. LNCS, vol. 4495, pp. 324–339. Springer, Heidelberg (2007). [https://doi.org/10.1007/978-3-540-72988-4\\_23](https://doi.org/10.1007/978-3-540-72988-4_23)
14. Hull, R., Damaggio, E., De Masellis, R., et al.: Business artifacts with guard-stage-milestone lifecycles: managing artifact interactions with conditions and events. In: Proceedings of the 5th ACM International Conference on Distributed Event-Based System, pp. 51–62. ACM, New York (2011)
15. State Machine Workflow in Windows Workflow Foundation. [https://msdn.microsoft.com/enus/library/ee264171\(v=vs.110\).aspx](https://msdn.microsoft.com/enus/library/ee264171(v=vs.110).aspx). Accessed 21 May 2017
16. OSworkflow. <https://java.net/projects/osworkflow>. Accessed 21 May 2016
17. Harel, D.: Statecharts: a visual formalism for complex systems. *Sci. Comput. Program.* **8**(3), 231–274 (1987)
18. Nigam, A., Caswell, N.S.: Business artifacts: an approach to operational specification. *IBM Systems Journal* **42**(3), 428–445 (2003)

19. SCXML. <https://www.w3.org/TR/scxml/>. Accessed 21 Nov 2017
20. Apache Commons SCXML. <http://commons.apache.org/proper/commons-scxml/>. Accessed 20 Dec 2016
21. Workflow Patterns. <http://www.workflowpatterns.com/patterns/control/>. Accessed 21 Mar 2017
22. <https://sysuworkflower.github.io/BOOWorkflow/>. Accessed 11 Nov 2017
23. <https://github.com/sysuworkflower/BOOWorkflow/releases>. Accessed 28 Jan 2018