Enhancing Business Process Automation by Integrating RFID Data and Events

Xiaohui Zhao[1], Chengfei Liu[1], and Tao Lin[2]

[1] Centre for Complex Software Systems and Services
Faculty of Information and Communication Technologies
Swinburne University of Technology
Melbourne, Australia
{xzhao,cliu}@groupwise.swin.edu.au
[2] Amitive Inc.
1400 Fashion Island Blvd
San Mateo, CA 94404, USA
tao.lin@amitive.com

Abstract. Business process automation is one of the major benefits for utilising Radio Frequency Identification (RFID) technology. Through readers to RFID middleware systems, the information and the movements of tagged objects can be used to trigger business transactions. These features change the way of business applications for dealing with the physical world from mostly quantity-based to object-based. Aiming to facilitate business process automation, this paper introduces a new method to model and incorporate business logics into RFID edge systems from an object-oriented perspective with emphasises on RFID's event-driven characteristics. A framework covering business rule modelling, event handling and system operation invocations is presented on the basis of the event calculus. In regard to the identified delayed effects in RFID-enabled applications, a two-block buffering mechanism is proposed to improve RFID query efficiency within the framework. The performance improvements are analysed with related experiments.

1 Introduction

RFID is a re-emerging technology intended to identify, track, and trace items automatically. Nowadays, the adoption of RFID is increasing significantly [1-3] in sectors of retailing, manufacturing, supply chain, military use, health care, etc. A bright forecast from IDTechEx [4] expects that the total RFID market value will rise from $4.96 billion in 2007 to $26.88 billion in 2017.

Attempts to apply RFID to facilitating the business running of real enterprises are often made by monitoring the material flows through reading events or collecting data from physical world. However, there is a distinct lack of business process management (BPM) support, and this lack barriers RFID-enabled applications from the benefits of on-site responses according to business logics, system agility against changing requirements, seamless integration with enterprise application systems, etc. Business process management blends business logics and related resources into reusable models for efficient transaction management. Once such business logics are incorporated into

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 255–272, 2009.

RFID systems, it can provide real-time information in a network consisting of different enterprises, applications, and business partners in collaboration scenarios, and enable automatic reactions and execution based on the captured real-time information while eliminating manual inputting, processing, and checking of information. As a milestone towards such integration, Boeing has already embarked on a new assembly paradigm that is focused on using RFID and collaborative processes for its 787 (formerly 7E7 Dreamliner) aircraft to reduce the assembly time [5]. The integration of RFID and business logics seeks methodological advances and facilitating architectures to merge the wire level deployment and business level applications under one umbrella.

With advancement of RFID technologies and the wider adoption of RFID applications, both research communities and industry companies are drawing attention to the fusion of RFID edge systems and application systems. By integrating business processes into RFID data management, we can effectively facilitate business process automation, and thereby improve the agility and efficiency of current business operations in the end. Towards this ultimate goal, this paper focuses on modelling and integrating RFID data and events according to existing business processes. This integration is expected to enable the awareness of RFID edge systems to both real-time context and business logic, and in turn the automation and orchestration of business processes can be well supported.

In this paper, we propose a framework for modelling business logics from an object-oriented and event-oriented perspective. This framework makes the following contributions to current RFID data management:

- Propose a formalised object-oriented and event-driven RFID data management model;
- Establish an event calculus based mechanism for designing event-based rules to capture the dynamics in the event-rich environment of RFID-enabled applications;
- Discuss the deployment of business logics and semantics to RFID edge system;
- Present a buffering mechanism for efficient query executions in conformity with the proposed data/event integration framework.

The remainder of this paper is organised as follows: Section 2 discusses the requirements and challenges for incorporating business logics into RFID edge systems by analysing an RFID-enabled application. Section 3 introduces the object-oriented modelling perspective and the event-driven mechanism, and then presents the event calculus based RFID data management model, and finally demonstrates the deployment of business logics to RFID edge systems with an example. Section 4 analyses the delayed effects in RFID query executions, and proposes a 2-block buffering mechanism with related experiment results. Section 5 reviews the related work and discusses the advantages of the proposed framework. Finally, we conclude the paper with an indication on future work.

2 Looking into RFID-Enabled Applications

In this section, we take a distribution centre for assembling shipments as an example of RFID-enabled application. As a typical chain of logistics processes, a distribution centre is often selected to discuss RFID deployment [6], since a distribution centre

assembles a large volume of shipments every day using pallets. Here we assume this distribution centre only packs pallets of two types of products and this distribution centre handles a large volume of shipments every day. We use this case to illustrate the nature of an RFID-applied environment.

Figure 1 shows the packing process at this distribution centre with a Business Process Modelling Notation (BPMN) [7] diagram. This packing process starts when receiving an order from customers, and thereafter pallets are transferred to the distribution centre, while the pickers are picking the ordered goods from the inventory and transferring them to the distribution centre. When receiving these goods, the packing station will periodically pack the goods at two packing lines, where line A can only do full pallet packing with the goods of the same type, and line B can do partial packing and mixed packing. After all ordered goods are packed, they will be sent for shipment. The details of sub process "Send Goods to Proper Lines" will be discussed in Section 3.4.

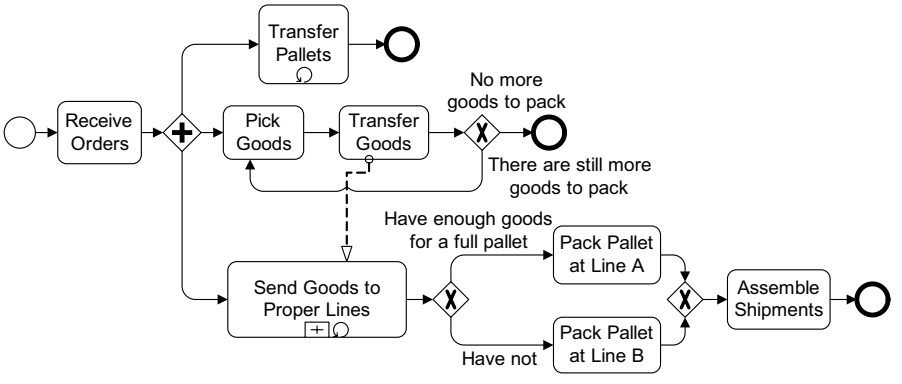

Fig. 1. Simplified business process diagram for the goods packing process

This distribution centre is an RFID "rich" environment. Not only the products and pallets, the equipments and tools are also tagged with RFID. The deployed readers in inventories, packing lines, transporting vehicles, etc., construct a network that monitors the movements of products, pallets, vehicles, etc. In this scenario, using RFID with the goods packing process in a distribution centre can bring the following benefits:

- Improve the visibility of the status of a business transaction. With RFID technology, which packing processes have started and which have not can be seen clearly. Therefore, early warnings for potential late shipments can be easily discovered.
- Improve the content accuracy for shipments.
- Record association hierarchy (association relationship of serialised cases with the corresponding pallet, pallets with the corresponding shipment). This information can be used for validating shipments.
- Recover recall or counterfeit goods.

In an RFID-applied environment, edge systems are responsible for most on-site operations, while monitoring the object movements [8]. If edge systems own the

knowledge of business logics, they can handle business operations intelligently, and as such the business efficiency and effectiveness can be improved. To this end, we pose the following questions to initiate the thinking on the integration of business logics in an RFID-applied environment:

- How to design event patterns and use them to elicit meaningful business information from low level tag events?
- How to blend the RFID events and event patterns into business logic modelling, so that the defined business rules are naturally sensitive to RFID events?
- How to characterise the real-time status of objects in an event-dominated context?
- How to track the real-time status of objects by sorting out related events, and can that be achieved in an efficient way?
- How to deploy the modelled business logics to RFID edge systems, and enable them to intelligently operate in response to run time dynamics of a real application environment?

Our work targets at incorporating business logics into RFID edge systems with a novel event calculus based data management framework. To cater for the event-driven and data-intensive execution mechanism of RFID applications, this framework defines rules ad RFID queries to elicit the business meaning from RFID tag read events. In addition, this model encapsulates the related classes, rules, event patterns into RFID contextual scenarios. Once such RFID contextual scenarios are deployed to the data-driven middleware systems, the business logics can be pushed down to RFID edge systems. Therefore, the RFID edge systems will become aware to both real-time object information and business logics. By modifying the deployed RFID application contexts, the same edge systems and infrastructures can change to support different applications. This feature enhances the flexibility and customisability of RFID application systems.

3 Object-Oriented RFID Data Management Framework

3.1 Object-Oriented Perspective and Event-Driven Mechanism

Classic business process management approaches specify the handling procedures and the involved business logic in form of graph-based workflow models. Such models emphasise the control flow between activities, and these activities execute along the pre-defined sequence, i.e., an activity's start-up is triggered by the completion of preceding activities. However, in the RFID-applied environment, RFID tag read events are the raw data to trigger operations and activities, and thereby control the business process execution. To effectively utilise such real-time object level information, we claim that the RFID data management framework should be fully object-oriented and event-driven. Further, the control of the event processing in the corresponding edge systems needs to be managed by a business process management system, yet the execution of the business logic will be conducted by the edge systems.

At technical level, we choose the event calculus as the tool for event and rule modelling. The event calculus is a logic-based formalism that infers what is true when given what happens, when and what actions do, and it is based on the supposition that "all change must be due to a cause, while spontaneous changes do not occur" [9]. The

event calculus particularly fits into the event based rule design and analysis in the event-rich environment of RFID-enabled applications. Compared with other event/state modelling approaches, such as UML state diagram and Event Process Chain (EPC) [10], event calculus owns advantages in modelling actions with indirect or non-deterministic effects, concurrent actions, and continuous changes, in a much concise representation.

3.2 Preliminaries of Event Calculus

The main components of event calculus include *events* (or action), *fluents* and *time points*. A *fluent* is anything whose numerical or boolean value is subject to change over time [9]. In this paper, we confine our attention to propositional fluents, i.e., Boolean fluents for simplification (note, this can be easily extended to situational fluents, i.e., the range of the fluents can be extended to be situations [9]). A scenario modelled by the event calculus constitutes predicates and axioms, which may refer to fluents, events, and time points as parameters. Table 1 lists the primary event calculus predicates.

In addition to these event calculus predicates, some *domain-specific predicates* may be used to describe special relations in a given scenario. For example, in the pallet packing scenario, we may use predicates *near_by*($pallet_1$, $pallet_2$), *contain*(*product*, *pallet*), etc., to specify the spatial relations between pallets and goods.

Table 1. Event calculus predicates

Predicates	***Explanation***
Initiates(e, f, t)	Fluent f starts to hold after event e at time t
Terminates(e, f, t)	Fluent f ceases to hold after event e at time t
$Initially_P(f)$	Fluent f holds from time 0, i.e., the initial time point
$Initially_N(f)$	Fluent f does not hold from time 0
$t_1<t_2$	Time point t_1 is before time point t_2
Happens(e, t)	Event e occurs at time t
HoldsAt(f, t)	Fluent f holds at time t
Clipped(t_1, f, t_2)	Fluent f is terminated between time points t_1 and t_2
Declipped(t_1, f, t_2)	Fluent f is initiated between time points t_1 and t_2

To guarantee its applicability, the event calculus introduces particular axioms to constrain the relationships between predicates, and solve the classic logic frame problem (please check reference [9] for details about the frame problem).

Domain-independent Event Calculus (EC) Axioms

(*EC*1) *Clipped*(t_1, f, t_4) $\leftrightarrow$ $\exists e$, t_2, t_3 [*Happens*(e, t_2, t_3) $\wedge$ $t_1<t_3$ $\wedge$ $t_2<t_4$ $\wedge$ *Terminates*(e, f, t_2)];

(*EC*2) *Declipped*($t1$, f, t_4) $\leftrightarrow$ $\exists e$, t_2, t_3 [*Happens*(e, t_2, $t3$) $\wedge$ $t_1<t_3$ $\wedge$ $t_2<$ t_4 $\wedge$ *Initiates*(e, f, t_2)];

(*EC*3) $HoldsAt(f, t_3) \leftarrow Happens(e, t_1, t_2) \wedge Initiates(e, f, t_1) \wedge t_2<t_3 \wedge \neg Clipped(t_1, f, t_3)$;

(*EC*4) $\neg HoldsAt(f, t) \leftarrow Happens(e, t_2, t_3) \wedge Terminates(e, f, t_1) \wedge t_2<t_3 \wedge \neg Declipped(t_1, f, t_3)$;

(*EC*5) $t_1 \leq t_2 \leftarrow Happens(e, t_1, t_2)$.

(*EC*6) $HoldsAt(f, t) \leftarrow Initially_P(f) \wedge Clipped(0, f, t)$;

(*EC*7) $\neg HoldsAt(f, t) \leftarrow Initially_N(f) \wedge \neg Declipped(0, f, t)$.

The first five axioms capture the behaviours of fluents once initiated or terminated by an event. To describe fluents' behaviours before the occurrence of any action which affects them, we axiomatise a general principle of persistence for fluents using (*EC*6) and (*EC*7), i.e., fluents change their values only via the occurrence of initiating and terminating actions. Here, we use symbol *EC* to represent the conjunction of these seven axioms, i.e., $EC = \bigwedge_{i=1}^{7} EC_i$.

Uniqueness-of-names (UNA) Axioms

To explicitly guarantee that there are no overlapping effects between events, it is needed to specify that the involved events are not identical. For example, UNA[*loaded*, *Alive*, *Dead*] indicates events *Loaded* ≠ *Alive*, *Alive* ≠ *Dead* and *Loaded* ≠ *Dead*. Here, we use symbol Ω to represent the conjunction of these UNA axioms.

3.3 RFID Data Management Model

Based on event calculus, we define an RFID data management model to characterise the behaviours of RFID objects.

Basic Definitions

Definition 1 (*RFID Class*)**.** An RFID class denotes a type of objects, which abstracts the common properties of these objects. Formally, an RFID class *c* is defined as tuple (*n*, *A*, *Q*), where

- *n* is the name of *c*;
- *A* is a set of attributes that *c* owns;
- *Q* is a set of fluent names, which can characterise the status of *c*'s instances.

Definition 2 (*Event Observation*)**.** The action of observing of an event raised by a reader or the system can be characterised as *Happens*(*e*, *t*), which denotes that event *e* occurs at time *t*.

In practice, events flow as a series, which consists of several event observations, such as, …, $Happens(e_1, t_1)$, $Happens(e_2, t_2)$, …, where $t_1<t_2$. We call such a flow of event observations as an *event series*.

Definition 3 (*Domain-dependent Rule*)**.** The domain-dependent logic is represented as the rules constituting the fluents and predicates introduced in the preliminary of event calculus. Syntactically, a domain-dependent rule *r* can be defined as $P \leftarrow \bigvee_{i=0}^{n} [(\bigwedge_{j=0}^{m} exp_{ij}) \vee exp_i]$, where

- $P \in \{Initiates(e, f, t), Terminiates(e, f, t), HoldsAt(f, t)\} \cup \{\text{domain-specific predicates}\}$;
- exp_{ij} and $exp_i \in \{Happens(e, t), HoldsAt(f, t)\} \cup \{\text{domain-specific predicates}\}$.

Definition 4 (*RFID Class Schema*)**.** An RFID class schema is a finite set Γ of RFID classes with distinct names such that every class referenced in Γ also occurs in Γ. i.e.,

$\forall$class $a \in \Gamma$, if a refers to class b, then $\exists b \in \Gamma$.

Definition 5 (*RFID Scenario*)**.** An RFID scenario S for an RFID schema Γ denotes an event calculus scenario with the RFID classes defined in Γ. Syntactically, S can be defined as tuple (Γ, R, EC, Ω)

- Γ is the RFID class schema;
- R is the set of rules, which are defined on the classes and fluents in Γ;
- EC and Ω represent the conjunctions of EC axioms and UNA axioms, respectively.

Definition 6 (*RFID Environment*)**.** An RFID environment *env* denotes the execution environment with actual events for an RFID scenario. An RFID environment *env* can be defined as tuple (I, Δ), where

- I denotes *env*'s initial settings, which are described using predicates $Initially_N$ and $Initially_P$;
- Δ denotes the received event series.

Definition 7 (*RFID Query*)**.** A query q_t over an RFID scenario S in a given RFID environment *env* can retrieve the real-time value of the fluents that are defined in S at a given time point t. Syntactically, a query q_t can be defined as $\leftarrow_{\Delta(t_0,t)\wedge I_{t0}} \rho_t$, where

- ρ_t denotes the target statement for the query in form of a conjunction of several $HoldsAt(f, t)$ predicates, i.e., $\rho_t = \bigwedge_i HoldsAt(f_i, t)$;
- $\Delta(t_0, t)$ denotes the set of events that are occurred from t_0 to t in environment *env*;
- I_{t0} denotes the initial settings at time t_0, i.e., the values of fluents at time t_0 in *env*;

In this paper, we confine that queries are all boolean ones, i.e., whether statement ρ_t is true or not at a given time point t according to RFID scenario S in environment *env*.

Definition 8 (*Operation Trigger*)**.** Driven by query results, an operation trigger *ot* can invoke corresponding operations of RFID edge systems in response to the dynamics of the RFID scenario. Syntactically, *ot* can be defined as

$|\leftarrow_{\Delta(t_0,t)\wedge I_{t0}} \rho_t| \Rightarrow invoke(op)$, where

- symbol "| |" denotes the boolean result of the query;
- $invoke(op)$ denotes the action of invoking operation *op* of the edge system, if the query returns true.

Figure 2 illustrates the relationship between the aforementioned notions. An RFID schema consists of a set of RFID classes, which specify the fluents and the attributes to be used in describing domain-specific rules in the RFID scenario. Such an RFID

scenario represents a self-contained system at build time, while an RFID environment represents the run time dynamics including the fluent values at the beginning time and the continuous event flow. Queries are used to retrieve the real-time fluent values, and the edge system can invoke proper operations in response to the query results via operation triggers.

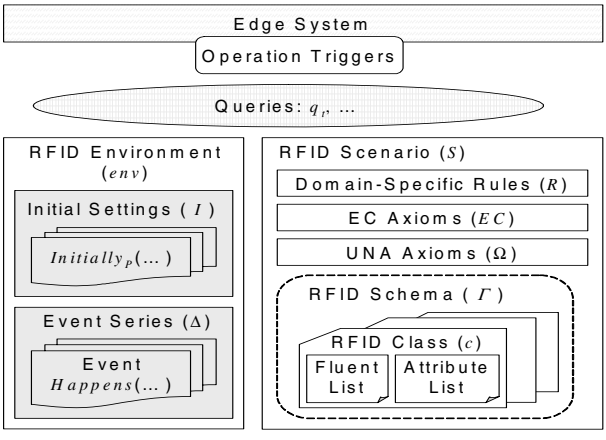

Fig. 2. RFID data management model architecture

The core part of our event calculus based model mainly relies on temporal predicates and rules, but this does not mean that our model is limited to temporal logics. Users can define domain-specific predicates, expressions and rules to represent complex business logics, as mentioned in Section 3.2.

3.4 Deployment of RFID Scenarios

We have conducted a pilot implementation of the proposed framework on a packing station at a distribution centre, which runs the same business process as shown in Figure 1. Here, we take sub process of "Send Goods to Proper Lines" as an example to evaluate if and how our approach can support business process automation in practice.

Figure 3 shows the details of this sub process in a BPMN diagram. In this scenario, the packing station has a temporary repository to store the received products. Once a product of type 1 comes, an event of "g1Arr" will be sent out, and be captured by task "Add the Number of Product Type 1", which is responsible for counting the products of type 1 in the repository. Similarly, task "Add the Number of Product Type 2" is responsible for counting the products of type 2 in the repository.

Event "stateCheck" is a periodical event, which triggers the execution of task "Check Repository". This task will check the number and types of the received products in the repository, and determines to send the products to which assemble line for packing. Here, line A can only pack full pallets of products of the same type, and line B can do partial or mixed pallets.

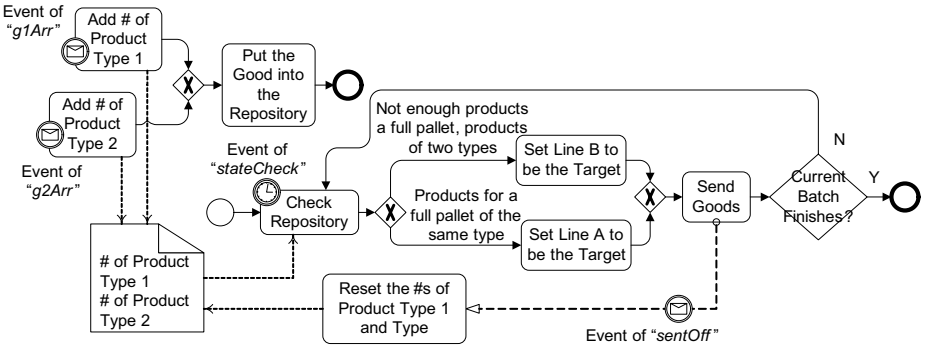

Fig. 3. Sub process "Send Goods to Proper Lines"

The current batch of products is considered to be over, if no more than a pallet of goods arrives in 4 seconds, and then the packing station will empty the repository and wait for the next batch. The main process covered by RFID technology was the tracking of the movement of goods from the band conveyer to the dispatcher. Tagging was done on individual product level.

According to the proposed model, we identify RFID classes *Dispatcher*, *Product Type* 1 and *Product Type* 2 in this scenario. With the classes, the following events are involved:

*g*1*Arr* –	a product of type 1 arrives to the dispatcher;
*g*2*Arr* –	a product of type 2 arrives to the dispatcher;
sentOff –	the deposited products are sent to a packing line;
stateCheck –	a periodical event to initiate the state checking of the dispatcher.

For class "Dispatcher", the following fluents are used to characterise a dispatcher's run time status.

Mixed –	the deposited products are of two types;
Full –	the repository has enough products for a full pallet;
Finish –	the current batch has been handled;
NotEmpty –	the repository is occupied;
Idle –	the dispatcher is standing by, rather than working.

In addition to these fluents, two variables, num_1 and num_2, are used to record the numbers of deposited products of type 1 and type 2, respectively.

The corresponding RFID scenario constitutes the RFID class schema including the three aforementioned classes, axioms *EC* and Ω, and the domain-dependent rule set *R* which comprises the following rules:

(R1) $[num_1{+}{+}, num_2{+}{+}] \leftarrow Happens([g1Arr, g2Arr], t)$;
(R2) $Initiates([g1Arr, g2Arr], NotEmpty, t) \leftarrow Happens([g1Arr, g2Arr], t) \wedge \neg HoldsAt(NotEmpty, t)$;
(R3) $Initiates(Mixed, t) \leftarrow Happens([g1Arr, g2Arr], t) \wedge (num_1 \neq 0) \wedge (num_2 \neq 0) \wedge \neg HoldsAt(Mixed, t)$;
(R4) $Initiates(Full, t) \leftarrow Happens([g1Arr, g2Arr], t) \wedge (num_1 + num_2 = MAX) \wedge \neg HoldsAt(Full, t)$;
(R5) $Terminates(Idle, t) \leftarrow Happens([g1Arr, g2Arr], t) \wedge HoldsAt(Idle, t)$;
(R6) $Terminates([Mixed, Full, NotEmpty], t) \wedge num_1 = 0 \wedge num_2 = 0 \leftarrow Happens(sentOff, t) \wedge HoldsAt([Mixed, Full, NotEmpty], t)$;
(R7) $Initiates(Idle, stateCheck, t) \leftarrow Happens(stateCheck, t) \wedge \neg HoldsAt(Idle, t) \wedge NoSentOff(t\text{-}4, t)$;
(R8) $Initiates(Finish, stateCheck, t) \wedge Terminates(Idle, stateCheck, t) \leftarrow Happens(stateCheck, t) \wedge \neg HoldsAt(Finish, t)\ ) \wedge HoldsAt(Idle, t) \wedge KeepsIdle(t\text{-}4, t)$;

Here, (R1) uses num_1 and num_2 to record the numbers of arrived products of *Product Type* 1 and *Product Type* 2, respectively. The square brackets denote a selective relation. (R2-5) adjust the values of fluents *NotEmpty*, *Mixed*, *Full* and *Idle* when a product arrives. (R6) resets the values of the mentioned fluents to be false once a "sentOff" event occurs. (R7-8) work on "*stateCheck*" events, were (R7) turns the dispatcher into "Idle" mode if no "sentOff" events occurred in last 4 seconds before the latest "stateCheck" event, and (R8) turns the dispatcher into "Finish" mode if the "Idle" mode has been lasting in last 4 seconds before a "stateCheck" event. The referenced predicates *NoSentOff* and *KeepsIdle* are defined as follows,

$$NoSentOff(t_1, t_2) = \bigwedge_{t_i \in [t_1, t_2]} \neg Happens(sentOff, t_i);$$

$$KeepsIdle(t_1, t_2) = \bigwedge_{t_i \in [t_1, t_2]} HoldsAt(Idle, t_i).$$

These two predicates in (R8-9) are subject to the event or fluent values prior to present time, and therefore they result in typical delayed effects in RFID queries. We will dedicatedly discuss about their influence to query execution in next section.

Once the RFID scenario is defined, it can be inputted into the edge system, i.e., the dispatcher. Thus, the dispatcher is empowered with the awareness about the products' arrivals and the business logics on where to send products for packing. Further, the following queries and operation triggers can be deployed to the dispatcher, and enable the dispatcher to intelligently react to real-time dynamics.

Query q_{1t}: $\leftarrow_{\Delta(t_0, t) \wedge I_{t0}} HoldsAt(Full, t) \wedge \neg HoldsAt(Mixed, t)$;
Query q_{2t}: $\leftarrow_{\Delta(t_0, t) \wedge I_{t0}} HoldsAt(Full, t) \wedge HoldsAt(Mixed, t)$;
Query q_{3t}: $\leftarrow_{\Delta(t_0, t) \wedge I_{t0}} HoldsAt(Idle, t) \wedge \neg HoldsAt(Full, t) \wedge HoldsAt(NotEmpty, t)$.

Trigger 1: $| q_{1t} | \Rightarrow$ invoke operation "send to Line A";
Trigger 2: $| q_{2t} | \vee | q_{3t} | \Rightarrow$ invoke operation "send to Line B".

An RFID environment contains the initial settings and the real-time event series of a concrete execution environment for an RFID scenario. In this example, the RFID environment may consist of the following content:

I={ $Initially_N(NotEmpty)$; $Initally_N(Full)$; $Initally_N(Mixed)$; $Intially_N(Idle)$; $Initally_N(Finish)$; MAX=4; num_1= num_2=0. };

Δ={ $Happens(g1Arr, t_1)$, $Happens(g2Arr, t_2)$, $Happens(g2Arr, t_3)$, $Happens(g1Arr, t_4)$, … (t_1< t_2< t_3< t_4<…)}.

Please note that MAX=4 denotes that a full pallet contains four products.

4 Efficient RFID Querying in Event Calculus Context

According to the proposed model, RFID queries are running to monitor the changes of the RFID scenario, and thereby invoke proper operations of edge systems to enable the business process automation. From our deployment practice, we find that the event and fluent dependencies influence a lot on the query execution performance. In this section, we are to investigate such dependencies.

4.1 Event/Fluent Delayed Effects in Queries

From Definition 7, we can see that for each query execution it needs to calculate the events occurred from t_0 up to the current time point. As time goes, the number of events increases towards infinite, and in turn this will reluctantly increase the query execution time towards infinite. To optimise the RFID query execution, we intend to cut off the event series for each query from the endless to a limited scope by eliciting useful information from previous query results. This intention is based on the fact that RFID-enabled applications always run queries continuously to monitor the real-time variations.

To optimise the event series, we have to consider the delayed effects caused by the rules referring to different time points. Take (R7) in last section as an example, fluent *Idle* changes to be true only if there are no "*sentOff*" event occurred in recent 4 seconds. We classify such delayed effects caused by events as *event delayed effects*, and the ones caused by fluent as *fluent delayed effects*, for example, fluent *Finish* in R(8) is subject to the values of fluent *Idle* of recent 4 seconds.

As shown in Figure 4, a series of events occur along the timeline. At time t_i, query q_{ti} is executed, and we can save the retrieved fluent values as a *fluent snapshot* for time t_i. When another query q_{tj} is to execute at time t_j, we intend to calculate the events from t_i rather than t_0 with the knowledge of the fluent snapshot at t_i. However, due to the delayed effects, some events occurred before t_i and some fluent values before t_i are also needed to when calculating for q_{tj}. Once q_{tj} is worked out, the fluent snapshot at t_j will be stored for the execution of later queries.

To specify the event delayed effects and fluent delay effects in RFID queries, we have regulated the form for general RFID queries, and proved this form can cover all possible cases.

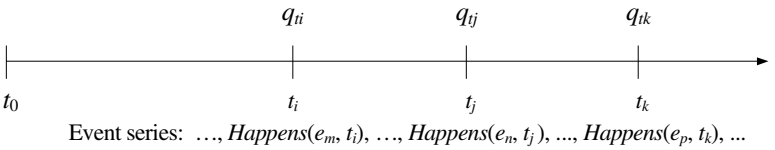

Fig. 4. Historical queries and subsequent queries

Equivalent Query Transformation

Suppose RFID scenario S has rule set $R=\{P\leftarrow \bigvee_{i=0}^{n} \{[\bigwedge_{j=0}^{m} Happens(e_{ij}, t\text{-}\Delta t_{ij})]\wedge[\bigwedge_{k=0}^{h} HoldsAt(f_{ik}, t\text{-}\Delta t_{ik})]\wedge exp_i\}\vee exp^{\circ}\}$, where variables i, j and k start from zero to cover all possible combinations. In S, Query $\leftarrow_{\Delta(t_0,t)\wedge I_{t0}} \rho_t$ is equivalent to $\leftarrow_{(\Delta(t_1,t)+\Delta')\wedge(I_{t0}+\Delta I)} \rho_t$, where

– $\Delta'=\{Happens(e_{ij}, t') \mid Happens(e_{ij}, t')\in\Delta(t_0, t_1), t_1\text{-}\Delta t_{ij}<t'<t_1$, where e_{ij} and Δt_{ij} are referred to by rules in $R\}$ (1)
– $\Delta I=\{HoldsAt(f_{ik}, t_1\text{-}\Delta t_{ik}) \mid f_{kj}$ and Δt_{ik} are referred to by rules in $R\}$ (2)

Details of the proof can be referred to report [11].

4.2 Two-Block Buffering Mechanism

Technically, the introduced query transformation deploys a specific space to buffer the previous events and fluent snapshots that may own delayed effects to later queries. This event buffering is similar to the slide window control in data stream processing, yet we only keep the events having delayed effects for future queries. The fluent snapshot buffering keeps the fluent values at historical time points by running extra queries. However, we need to note that such extra queries may refer to the fluent snapshots of earlier time points, which will require more queries. As such, an update to the snapshot buffer may result in a series of recursive queries, which will seriously damage the query execution performance. To avoid such recursive querying, we propose a two-block buffering mechanism. First, we define the data structure for a buffer.

For a given RFID scenario S and a given RFID environment *env*, the data structure of a buffer b can be defined as tuple $(t, \Delta', I, \Delta I)$, where

– t, the time point for buffer b;
– Δ', the buffered events that occurred before t, i.e., Δ' of formula (1) for S and *env*;
– I, the fluent snapshots at time point t;
– ΔI, the buffered fluent snapshots that occurred before t, i.e., ΔI of formula (2) for S and *env*.

The two-block buffering mechanism runs two such buffers to record the data with delayed effects for later queries. The one with an earlier time point is called *back*

buffer (*bb*), while the other is called *fore buffer* (*fb*). As Figure 5 shows, each buffer reserves the related past events and fluent snapshots, which are indicated by the shadowed rectangle left to the buffer. The buffer size, which is measured by the time period that the buffered data span over, is subject to the rule set R of RFID scenario S, and is therefore bounded. In addition, the two buffers must be guaranteed to be non-overlapped with each other. The initial content of *bb* can be obtained by calculating from time t_0, while *fb*'s content can be done by calculating from *bb.t* with *bb*'s buffered content.

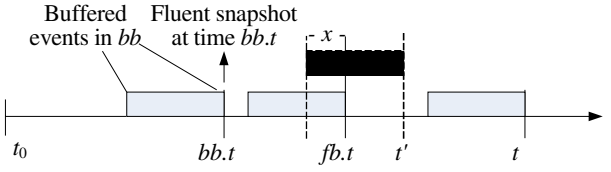

Fig. 5. Two-block buffering

If a query q_t runs at time t (t>*fb.t*) and all the required historical data occurred after *fb.t*, these historical data can be obtained by calculating from *fb.t*, with *fb*'s buffered content. If a query q_t' runs at time t' (t'>*fb.t*), where this query requires some historical data of the time earlier than *fb.t*, it indicates that some past events and fluent snapshots occurred in time period x, as shown in Figure 5, have delayed effects to query q_t'. For the fluent snapshots taken during period x, they cannot be calculated out from *fb*'s buffered content, and will require more queries over earlier historical data, which may cause query recursions. We prevent such recursions by using *bb*'s buffered content instead of *fb*'s to calculate out q_t'.

This buffer updating mechanism guarantees that each query only needs to calculate the events occurred from *bb.t* or *fb.t* to current time plus the buffered events, rather than the whole event series from t_0. This mechanism is independent with queries, and therefore can support multiple heterogeneous queries to run over the same RFID scenario in an RFID environment.

The details of this 2-block buffering mechanism is given in report [12].

4.3 Experiment Results

To test the efficiency of the buffering mechanism, we have implemented a simulation environment on Cygwin (Linux Emulation for Windows) with IBM Discrete Event Calculus (DEC) Reasoner [13], relsat 2.02 [14] as the propositional satisfiability solver. The event input stream is simulated by a large event log file which can be randomly generated by programs. The simulation was performed on a personal computer with a Core 2 6300 CPU at 1.86 Ghz, 1 GB memory, and Windows XP OS.

The rule set for testing is based on the one discussed in the deployment section with slight modifications to adapt to DEC reasoner's encoding format. Queries are executed every 5 seconds, and we take the execution time as the main performance indicator.

To demonstrate the performance improvement, we execute the same queries under three mechanisms.

- Naive mechanism. This mechanism calculates all occurred events from time point 0s without any buffering.
- Periodical buffering mechanism. This mechanism periodically stores the fluent snapshots and discarding some old events. In this test case, to prevent the delayed effects, we set the interval to be 8 seconds (4 seconds for fluent delayed effect and 4 seconds for event delayed effect as indicated by (R7-8)), i.e., every 8 seconds it will record the current fluent snapshot and discard the events occurred 8 seconds before. This mechanism follows the idea of the fixed period partition mechanism proposed by Siemens Lab in [15].
- The proposed 2-block buffering mechanism. Here, we set the length of fore buffer and back buffer to be 4 seconds according to (R7-8).

Table 2. Comparison of query execution times

Query time point (seconds)	Naive (s)	Periodical (s)	2-Block (s)	Periodical * (s)
5	1.6	1.6	1.6	3.1
10	3.4	3.4	2.9	4.9
15	5.4	5.4	3.6	5.8
20	7.7	3.5	2.4	5.0
25	11.2	2.6	2.7	4.1
30	13.0	4.1	3.0	5.6
35	16.0	3.4	3.3	4.9
40	19.6	4.8	2.5	6.3

Table 2 lists the query execution time under these three mechanisms. Due to the limit of the simulation platform, the simulation results are only used to show the performance difference between buffering mechanisms, yet not indicate the practical execution time on real systems.

Table 3. Extra time cost for updating fluent snapshots in the periodical buffering mechanism

Query time points (s)	8	16	24	32	40
Execution time	2.8	2.4	2.4	2.5	2.4

Compared to the other two buffering mechanisms, the periodical buffering has to run extra queries to update the fluent snapshots on certain time points, i.e., time points $t \times 8$ (t=1, 2, 3, …). Table 3 lists the time cost for these queries, and from these data we can work out the mean extra time cost for each query is (2.8+2.4+2.4+2.5+2.4)/8 ≈1.5s. Take into account the extra query cost, we change the query execution time by adding the mean time cost 1.5s, and put the new data in the column titled "Periodical *" in Table 2.

The query execution time of the periodical buffering mechanism fluctuates as the query time point changes. This indicates the execution time is influenced by the distance between the query time point and the latest buffering point. As the time for updating fluent snapshots follows the pattern of $t \times 8$ (t=1, 2, 3, …), at time point 15s, the latest updating is at time point 8s, and events from 0s to 8s are stored in the buffer. Therefore the calculation starts from 0s to 15s (range of 15s) with the fluent snapshot at time point 8s. At time point 25s, the latest updating is at time point 24s, and the calculation starts from 16s to 25s (range of 9s) with the fluent snapshot at time point 16s. The different event calculation ranges result in the variation of query execution time.

In contrast, the query execution time for the 2-block buffering mechanism keeps very smooth around 2.7s. This is because the 2-block buffering mechanism only updates fluent snapshots at the time of executing users' queries. For example, at time point 15s, the back buffer (*bb*) and the fore buffer (*fb*) were created at 5s and 10s, respectively, where *bb* buffered the events from 1s to 5s with the fluent snapshots at time point 5s, and *fb* did the events from 6s to 10s and with fluent snapshot at time point 10s, respectively. When a users' query is run at time point 15s, it calculates with the events occurred from 6s to 10s and from 11s to 15s, with fluent snapshot at time point 10s. The query result will be used to update the fluent snapshot at time point 15s. In addition, this updating mechanism does not involve extra queries cost like the periodical one does. This experiment clearly shows that our 2-block mechanism stands out with higher performance.

5 Related Work and Discussions

Major software vendors, like IBM (WebSphere RFID), Sun (Java RFID System), BEA (BEAWebLogic RFID), SAP (AII), etc., attempt to create RFID integration platform and deploy business process management to RFID-enabled applications. For instance, SAP's Auto-ID Infrastructure (AII) project aims to facilitate the connection between RFID devices, middleware systems, and business application systems [16, 17]. Components of the association data management (ADM) and the action & process management (APM) are responsible for managing the contextual information and the received events, and navigating the activity handling, respectively.

To better describe business logics in data (event)-intensive business scenarios, the object-oriented (or artifact-oriented) perspective has been proposed recently as a new modelling method [18-20]. Compared with traditional business process modelling approaches, the object-oriented modelling approach focuses on the business contexture and behaviours with declarative logics, rather than the sequencing of activities. In this way, the object-oriented modelling enables business actors to be aware of what can be done instead of what should be done.

A lot of research efforts have been put to tackle RFID complex event processing, yet most of them mainly focus on data cleansing and filtering. SASE [21, 22] has defined a SQL like complex event language to aggregate RFID events. The implemented SASE system uses a persistence storage component to support querying over historical data and to allow query results from the stream processor to be joined with stored data. In addition, the extended sliding window control and indexing techniques

have been adopted by work [23] and [24] to improve the performance of continuous query processing over RFID event flows. Hu, Misra and Shorey have addressed the query issue from the perspective of energy efficiency in [25]. Wang, Liu, and Bai have investigated the temporal management of RFID data [15]. They have adapted traditional database query techniques to the temporal relationships of RFID data, and thereby defined a set of temporal complex event constructors in [26]. Two partitioning mechanisms have also been proposed in their work to support efficient queries. However, none of the mentioned works have provided an explicit solution on how to handle the delayed effects in event management, or how to integrate business process automation into RFID event management.

Our work aimed to integrate the business rules and RFID event data management together, and thereby empowered the RFID edge systems with the awareness to both business logics and real-time object-level information. A formal RFID data management framework was presented to model the involved business logics, RFID scenarios, queries and so on, on the mathematical basis of the event calculus. Particularly, the following features distinguished our RFID data management framework from others:

- Concise representation

With time dependent fluents, our RFID data management model is more elegant in representation and more powerful in expressivity, in comparison with traditional event/status modelling approaches, like UML state diagram and EPC, which has to awkwardly employ a large number of intermediate states to represent all possible status and delayed effects.

- Integration of business rules to RFID data management

The proposed model well composes the business rules, operation invocations, into the RFID scenario modelling to realise the design of self-contained and autonomous RFID systems. The customisable rules also enable the re-configurability of RFID edge systems.

- Edge system level business logic deployment

Our approach focused on deploying the modelled business scenarios to RFID edge systems. With the injected business logics, RFID edge systems can respond to the received events on site according to the defined rules and operation triggers. This feature is practically important for real-time handling applications.

- RFID query optimisation

In regard to the delayed effects in RFID queries over continuous event flows, our proposed 2-block buffering mechanism reused historical data to shorten the event series for calculus. The experiment results proved our mechanism is with better performance than others.

6 Conclusion

This paper looked into the business logic modelling for RFID-enabled applications. By establishing a formal RFID data management model, we proposed a novel method to specify the RFID classes, domain-dependent rules, real-time event series, queries and operation triggers, as well as inter-relations among these notions. This method

fully catered for the features such as event-rich communications, large event volumes, event and fluent delayed effects, etc., in the RFID-applied environment. In compliance with this method, a query optimisation scheme with a two-block buffering mechanism was discussed for improving RFID query performance.

Our follow-up work is to further refine the proposed RFID data management model to support more complex business transactions, and enrich the optimisation mechanism with more techniques for different query cases.

References

1. Banks, J., Pachano, M., Thompson, L., Hanny, D.: RFID Applied. John Wiley & Sons, Chichester (2007)
2. Koh, S.C.L.: RFID in Supply Chain Management: A Review of Applications. In: Kumar, S. (ed.) Connective Technologies in the Supply Chain, pp. 17–40. Taylor & Francis, Abington (2007)
3. Liu, C., Li, Q., Zhao, X.: Challenges and Opportunities in Collaborative Business Process Management. Information System Frontiers 11, 201–209 (2009)
4. IDTechEx (2007), http://www.IDTechEx.com
5. Gillette, W.: Boeing Presentation at the Aerospace & Defense Summit (2004)
6. Bottani, E.: Reengineering, Simulation and Data Analysis of an RFID System. Journal of Theoretical and Applied Electronic Commerce Research 3, 12–29 (2008)
7. OMG: Business Process Modeling Notation (2009), http://www.omg.org/spec/BPMN/1.2/
8. Glover, B., Bhatt, H.: RFID Essentials. O'Reilly, Sebastopol (2006)
9. Shanahan, M.: The Event Calculus Explained. Artificial Intelligence Today, pp. 409–430. Springer, Heidelberg (1999)
10. Scheer, W.A.: Business Process Engineering, ARIS-Navigator for Reference Models for Industrial Enterprises. Springer, Berlin (1994)
11. Zhao, X.: Report: Proof of RFID Query Equivalent Transformations (2009), http://www.ict.swin.edu.au/personal/xzhao/proof.pdf
12. Zhao, X.: Report: 2-Block Buffering Mechanism for RFID Query Optimisation (2009), http://www.ict.swin.edu.au/personal/xzhao/algorithm.pdf
13. IBM Discrete Event Calculus Reasoner, http://decreasoner.sourceforge.net/
14. relsat 2.02, http://www.bayardo.org/resources.html
15. Wang, F., Liu, P.: Temporal Management of RFID Data. In: The 31st International Conference on Very Large Data Bases, Trondheim, Norway, pp. 1128–1139 (2005)
16. Bornhövd, C., Lin, T., Haller, S., Schaper, J.: Integrating Automatic Data Acquisition with Business Processes - Experiences with SAP's Auto-ID Infrastructure. In: The 13th International Conference on Very Large Data Bases, Toronto, Canada, pp. 1182–1188 (2004)
17. Götz, T., Safai, S., Beer, P.: Efficient Supply Chain Management with SAP Solutions for RFID. Galileo Press (2006)
18. Nigam, A., Caswell, N.S.: Business Artifacts: An Approach to Operational Specification. IBM Systems Journal 42, 428–445 (2003)
19. Liu, R., Bhattacharya, K., Wu, F.Y.: Modeling Business Contexture and Behavior Using Business Artifacts. In: The 19th International Conference on Advanced Information Systems Engineering, Trondheim, Norway, pp. 324–339 (2007)

20. Hull, R.: Artifact-Centric Business Process Models: Brief Survey of Research Results and Challenges. In: Meersman, R., Tari, Z. (eds.) OTM 2008, Part II. LNCS, vol. 5332, pp. 1152–1163. Springer, Heidelberg (2008)
21. Gyllstrom, D., Wu, E., Chae, H.-J., Diao, Y., Stahlberg, P., Anderson, G.: SASE: Complex Event Processing over Streams. In: The 3rd Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, pp. 407–411 (2007)
22. Wu, E., Diao, Y., Rizvi, S.: High-performance Complex Event Processing over Streams. In: The ACM SIGMOD International Conference on Management of Data, Chicago, Illinois, USA, pp. 407–418 (2006)
23. Bai, Y., Wang, F., Liu, P., Zaniolo, C., Liu, S.: RFID Data Processing with a Data Stream Query Language. In: The 23rd International Conference on Data Engineering, Istanbul, Turkey, pp. 1184–1193 (2007)
24. Park, J., Hong, B., Ban, C.: A Continuous Query Index for Processing Queries on RFID Data Stream. In: 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, Daegu, Korea, pp. 138–145 (2007)
25. Hu, W., Misra, A., Shorey, R.: CAPS: Energy-Efficient Processing of Continuous Aggregate Queries in Sensor Networks. In: The 4th Annual IEEE International Conference on Pervasive Computing and Communications, Pisa, Italy, pp. 190–199 (2006)
26. Wang, F., Liu, S., Liu, P., Bai, Y.: Bridging Physical and Virtual Worlds: Complex Event Processing for RFID Data Streams. In: The 10th International Conference on Extending Database Technology, Munich, Germany, pp. 588–607 (2006)