RC22406 (W0204-087) April 24, 2002
Computer Science

IBM Research Report

FlexFlow: Workflow for Interactive Internet Applications

Rakesh Mohan, Mitchell A. Cohen, Josef Schiefer
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

LIMITED DISTRIBUTION NOTICE: This report has been submitted for publication outside of IBM and will probably be copyrighted if accepted for publication. Ithas been issued as a Research Report for early dissemination of its contents. In view of the transfer of copyright to the outside publisher, its distributionoutside of IBM prior to publication should be limited to peer communications and specific requests. After outside publication, requests should be filled only by reprints or legally obtained copies of the article (e.g, payment of royalties). Copies may be requested from IBM T. J. Watson Research Center , P. O. Box 218, Yorktown Heights, NY 10598 USA (email: reports@us.ibm.com). Some reports are available on the internet at http://domino.watson.ibm.com/library/CyberDig.nsf/home

FlexFlow: Workflow for Interactive Internet Applications

Rakesh Mohan, Mitchell A. Cohen and Josef Schiefer
{rakeshm, macohen, josef.schiefer}@us.ibm.com

IBM T.J. Watson Research Center
PO Box 704
Yorktown Heights, NY 10598

Abstract

Traditional workflow systems are not suited for highly interactive online systems. We present a state-machine based workflow system, named FlexFlow, *which formally describes Internet applications using state charts. The FlexFlow engine uses these descriptions to directly control the execution of the applications. FlexFlow helps in generating controls for user interactions on web pages. Different versions of an application can be generated by visually editing its FlexFlow description, with minimal incremental effort in rewriting application software or related web pages. FlexFlow, thus, provides an efficient way to customize online systems to variations in business processes or to support different versions of a business processes in the same e-business system for different sets of industries, organizations, users, or devices. We demonstrate FlexFlow's use for rapid prototyping of business processes. We describe how we have used FlexFlow in commercial platforms for B2B e-commerce.*

Keywords: Interactive applications, e-business systems, workflow management, business process management, e-commerce platforms.

Approximate number of words: 8000

1. Introduction

In business systems, abstraction of the process logic from the embedded task logic enables the business processes to be modified independent of application code. The implementation of an e-commerce platform at a company often requires a customization of processes, such as an order process or a Request for Quotes, to the existing environment of that company. Workflow technology is prevalent for the modeling, analysis and execution of business processes [19][7].

Business process management is critical in a three- or multi-tier environment of e-business systems. Business rules and process information is extracted from the business logic tier and is presented in a workflow-based environment, which manages the execution of the business processes. Consequently, this approach greatly simplifies the application logic at each step. Business rules become explicit, visible, and rapidly changeable. There is also a need for stimulating and supporting communication about system changes to be made between the development team and the business, and between the business and its partners i.e., the customers and suppliers.

Business processes vary with a company's business model, and its industry sector. In e-commerce systems, trading mechanisms, such as auctions and negotiations are varied to suit particular business partners, product categories or market conditions. Business processes are customized to the role of the user and the terms and conditions of a contract with the user's organization. For example, the registration process for an administrator may be different from that of a buyer, and whether payment precedes or follows order confirmation may depend on the terms of a contract. E-commerce platforms thus need to provide both an easy way to modify business processes and to maintain variations of business processes. The separation of process and task logic allows both the easy customization of business process and reuse of the task logic in the variants of a business process.

In most current e-commerce systems, the steps of a business process, or the actions a system takes in response to user actions in such a process, are not made explicit, but are buried in software code for both the dynamic pages and the application server. This makes the

modification of implemented business processes extremely difficult and fragile. For example, to change the ordering of the process steps requires substantial rewriting of the software for the application and the web pages for the user interface. For e-commerce platforms made to be used by different companies, this presents a big problem as most companies' business processes differ from those of other companies to a small or large extent. Thus, deploying such e-commerce platforms at each different company incurs a large overhead in terms of time and money required to rewrite the business processes. Often, this overhead actually forces companies to adjust their business processes to conform to an e-commerce system instead of modifying the system to match their preferred processes.

In this paper, we present an approach for managing web-based business processes based on state machines that is more suited for the interactive nature of online systems than traditional workflow systems. We introduce a system which facilitates communication about system change with a descriptive model in which "as-is" and "as-to-be" models represent business processes. Since the e-business environment is so dynamic, change often overtakes models before delivering any significant results. Business people, rather than information technology experts, must be able to develop and extend the business process model. Hence, tools are required that facilitate business experts in communicating their vision and insights via a descriptive model.

First, we will discuss the contribution of this paper in respect to the existing related work. Then, we give an overview of the FlexFlow system, introduce the FlexFlow process model, and explain how defining business processes with FlexFlow can drive e-commerce development. We wrap up with our real world experiences using FlexFlow and where we can go with it next.

2. Contribution and Related Works

Business Process (Re-)Engineering [8] is an important driving force for workflow management. It aims at increasing the efficiency of business processes and making them easily and quickly adjustable to the ever changing needs of customers. In contrast to specifications of business processes, workflow specifications serve as a basis for the largely automated execution of processes. Workflow specifications are often derived from business process specifications by refining the business process specification into a more detailed and more concrete form. Automated and computer-assisted execution means that a workflow management system

(WfMS) [7][14][18] controls the processing of activities, which have to be performed in the workflow. Some activities may have a manual or intellectual part, to be performed by a human. But the workflow management system is in charge of determining the (partial) invocation order of these activities. In contrast to business process specifications, this requires a formal specification of control flow and data flow.

Workflow specifications based on script languages contain control flow and data flow constructs which are specifically tailored to workflow applications. Such script languages are popular in current WfMS products. They provide a compact representation and are therefore easy to use. A drawback of most script languages is their lack of a formal foundation. Their semantics is mostly 'defined' by the code of the script interpreter used.

Leymann argues in [16] that state transition nets are a good choice when a graphical visualization of workflow specifications has high priority. In state transition nets, activities are represented by nodes, and control flow is represented by edges. In fact, almost all WfMS products provide means for graphical specifications similar to state transition nets.

Considering only net-based methods with a formal foundation, we have to restrict ourselves more or less to state charts [9] and Petri nets [6][22]. Variants of Petri nets, especially predicate transition nets, are used in a number of research prototypes as well as in several WfMS products [5][21]. Some workflow management systems use variants of Petri nets for the internal representation of the workflow engine, e.g. [23]. State charts [9][12] have received little attention in workflow management, but they are well established in software engineering, especially for specifying reactive systems. In the MENTOR project [29], state charts are used as a formal foundation for workflow specification.

Event-Condition-Action-Rules (ECA) rules, are used in active database systems and have been adopted by a number of projects in the workflow area (e.g., [15]). ECA rules are used to specify the control flow between activities. Like for other methods that are not based on nets, the graphical visualization of sets of ECA rules is a non-trivial task. Large sets of ECA rules are hard to handle, and a step-wise refinement is not supported [25]. In terms of their formal foundation, ECA rules are typically mapped to other specification methods, especially variants of Petri nets or temporal logic.

Van der Aalst and Basten discuss in [1] inheritance concepts for changing business processes to manage ad-hoc changes and evolutionary changes. They argue that today's workflow management system have problems dealing with both types of changes, because they do not support dynamically changing processes or the process are supported in a rigid manner, i.e. changes are not allowed or handled outside of the workflow management system.

2.1 Contributions

Workflow is the dominant technology for modeling and controlling the execution of business processes [19]. The Workflow Management Coalition defines workflow as "The automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [30].Workflow treats processes as an assembly-line of task steps. Let us consider a simple insurance business process to illustrate workflow. The process starts with a field agent filing and insurance claim. Next, the claim is placed in the in-basket of a claim adjuster. The claim adjuster goes through claims in her in-basket one by one. Once she processes a claim she sends them to the in-basket of the reimbursement handler. Workflow systems are designed primarily for such processes where a set of documents goes through a number of processing steps. Each process has a worklist, similar to the "in-box" of our example. The processing stages may be actual people or software applications, such as a credit check system. The workflow is captured in executable "scripts" and business processes can be modified by changing their scripts. Usually workflow systems have visual modeling tools for defining the process and generating the scripts.

The pattern of user interaction with e-commerce business processes is very different from that of traditional workflow systems. Online business systems are highly interactive. Internet applications follow the request-response model (mirroring HTTP). In online business systems, a user takes an action, such as clicking a submit button on a web page. This results in the form data on that page being sent to the system, the system acting on it and presenting another page to the user. For example, a user goes to a shopping web site, fills out the login page and clicks the submit button. This results in her user name and password being sent to the system, which confirms that she is a legal user and sends her the catalog page. Then the system waits until the user selects products to fill the shopping basket. This interactive, or conversational, pattern of

system action based on a user action and then waiting for the user to initiate the next step is not well modeled by existing workflow systems. We believe that is a major reason why online e-business applications do not use workflow systems.

Another problem is that workflows systems are large, complex and expensive. For online applications, this both increase the cost of deployment and the responsiveness when servicing a large number of concurrent requests. Microflows [4] have been proposed to address this drawback of workflow systems. Microflows are small footprint workflow systems crafted for a particular class of applications. They provide minimal or no support for services provided by full workflow systems such as transaction management, guaranteed messaging and worklists. Microflows provide the benefits of abstracting process logic from task logic while at the same time improving the responsiveness and reducing the cost as compared to industrial strength workflow systems.

State machines are widely used for implementation of network protocols to describe the conversation/interaction between a sender and receiver. Business processes for e-Commerce platforms also interact frequently with each other. Examples are negotiation scenarios between buyers and sellers, where each party has to maintain the current state of the negotiation to decide the next actions. State machines have been also used to model negotiations [26]. They have been used for real-time systems; a system reacts or responds to events with a quick, nearly instantaneous response [27]. Thus, there is strong evidence to support that state machines would be useful for interactive, conversational and responsive online business systems.

In this paper, we show how to employ the formal method of state charts [9][12] for the specification of processes for e-commerce platforms. By using state charts as our specification method, we are able to model business processes which can be automatically executed by a workflow engine. Our contribution is the introduction of process state diagrams, which use the state starts notation for modeling business processes. Furthermore, we introduce the FlexFlow system, which supports the formal specification of process state diagrams, including the simulation and execution of these diagrams.

UML is widely used in software engineering practice so more developers are conversant with state diagrams then workflow models. A basic understanding of finite-state-machines (FSMs) is even more prevalent, and state charts extend the modeling available in FSMs.

In this paper, we present FlexFlow which is both suited for interactive applications and is light-weight. FlexFlow helps creation and alteration of business processes. It uses state machines to (a) describe the actions that can be taken by particular user, based on her roles at particular points in a process, (b) to enforce the validity of user actions, (c) to track the execution of actions within an instance of the business process, (d) to provide the user interface with a list of actions available to a user working on an instance of the business process, (e) to provide coordination between state machines, and (f) to allow different organizations to have varied business processes.

3. FlexFlow - Overview

Figure 1 shows the lifecycle of business processes in the FlexFlow system. A visual modeling tool is used to design business processes as process state diagrams. The visual modeling tool generates from the process state diagrams a XML representation, which is a full description of the business process. It contains all the information required by the FlexFlow engine to control the execution of the business process. This XML description is compiled and loaded into the FlexFlow system database tables. An addition table is used to store the current state of each instance of a business process running at a given time in the business system. These tables are used by the FlexFlow engine to (a) control the execution of the business process and (b) to control the user interface.

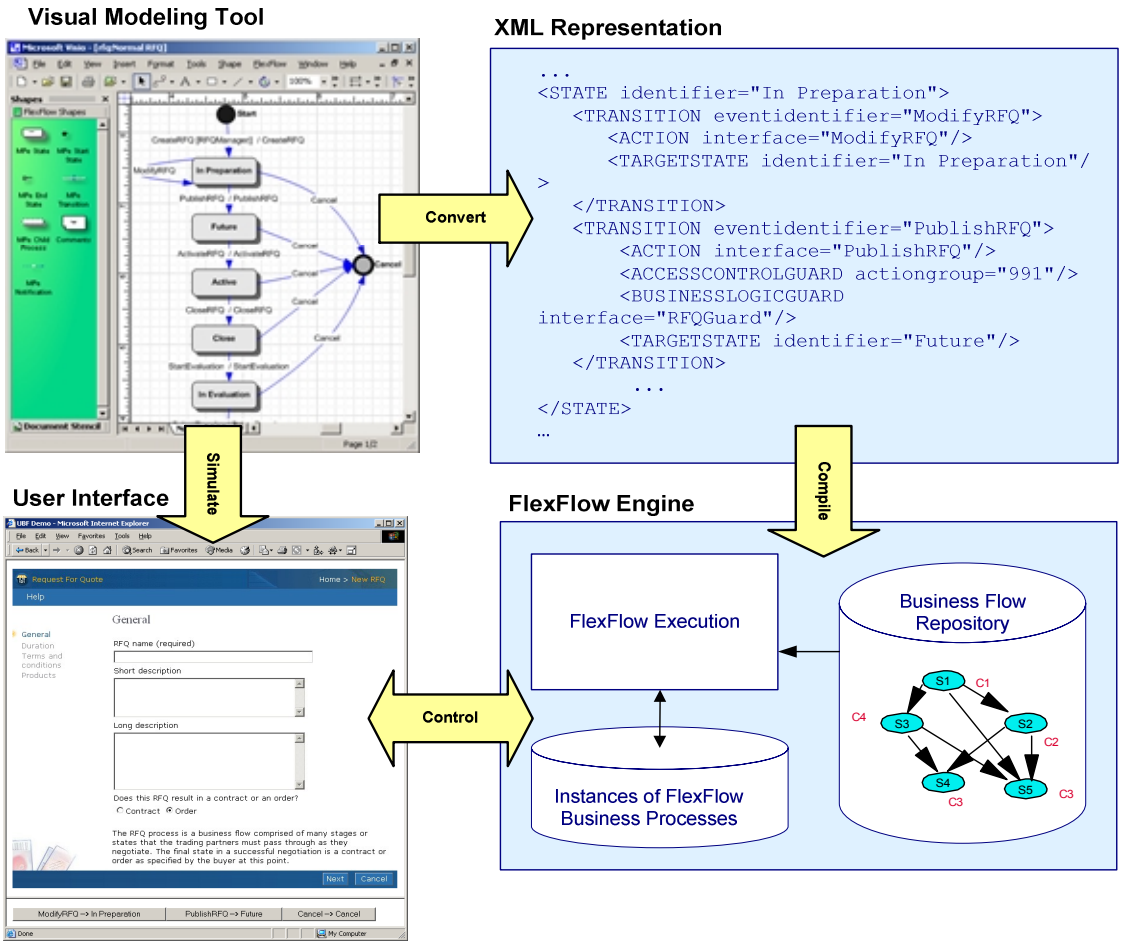


Figure 1: FlexFlow - Lifecycle of a Business Process

The visual modeling tool is also used to modify business processes definitions. Different versions of the business process are stored in the business flow storage. Business processes can be changed with limited change task logic or computer programs that are implementations of the business processes. A commerce function is reconfigured simply by reconfiguring its corresponding state machine.

The FlexFlow system also includes a component for simulating business processes. Thereby, users and developers can explore process variations to reach a common vision of how the user might interact with the system to perform a task.

In Section 8 we describe how business processes are coordinated by allowing an instance of one state machine to trigger an action in one or more instances of the same or different state

machines. For example, it may be desired that the closing of an auction by a seller in an auction business process disable the ability of buyers to increase their bids in the bidding business process.

4. The FlexFlow Process Model

FlexFlow models e-commerce business processes as Unified Modeling Language (UML) state diagrams [28], which are an adaptation of Harel's statecharts [9][12]. UML uses state diagrams to describe the behavior of objects, whereas, FlexFlow uses statecharts describe processes. We adopt the UML state diagram notation for the FlexFlow process models.

Finite State Machines have serious limitations such as there is no support of hierarchical states, inheritance and delegation of messages. To address these shortcomings Harel proposed state charts [9]. State charts have been adapted in the Universal Modeling Language (UML) [28] and thus are well known and widely used in the software engineering community.

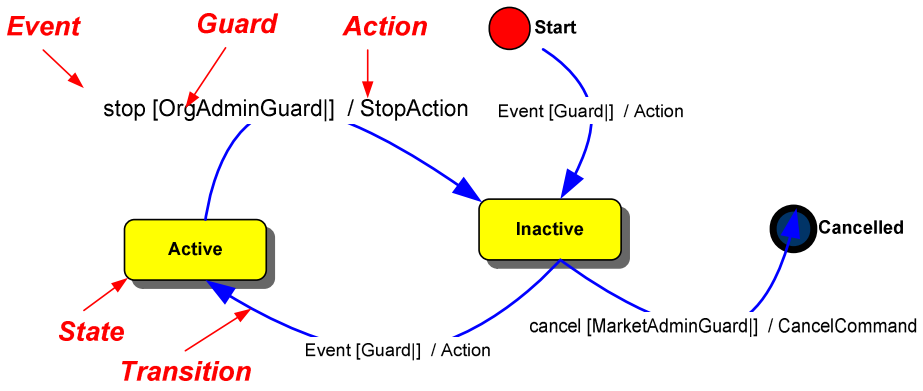


Figure 2: FlexFlow State Charts

UML state charts are directed graphs with nodes called *states* and the directed edges between them called *transitions* (see Figure 2). FlexFlow models interactive online business processes as state charts. However, unlike UML, where state charts describe the behavior of objects, FlexFlow state charts describe processes. In addition to the functionality of Harel state charts and UML state diagrams, FlexFlow adds three key features: the concept of roles, the coordination of interactions of multiple parties, and the ability to allow different organizations to use different versions of the business process. Business processes are versioned as different state charts.

Versions can be selected based on membership at the organization level. Versions can also be selected based on other factors including the mode of interaction, such as device, browser, and messaging method. The UML notation used by FlexFlow consists of states, transitions, events, guards, and context.

Actions

Actions correspond to task logic being executed at the application server. For FlexFlow they are atomic units of business work. Actions can appear in states and transitions. An action can be used to interface to an external system, such as a workflow system handling its own set of functionality. An action can be a conglomeration, or sequence, of pre-defined internal commerce actions. All actions caused by the processing of an event are run within the same transaction.

States

States correspond to stages in a business process. A state identifies a precise point within the process. In a given business process at a given state, the actions that can be taken by various parties is completely defined by the set of outgoing transitions. A state may have an *entry action*, an action that is executed upon entering the state, and an *exit action*, an action that is executed upon leaving the state. In FlexFlow, *entry actions* are allowed to trigger new events which in turn get processed by FlexFlow.

Transitions

A transition represents a change of the process state. It connects two states, a source state it exits and a target state it enters. A transition corresponds to an *action* that is taken in response to an *event*. The transitions may further have *guards* on them. These guards are checked, and the transition is taken only if they are true. Only one transition out of a state is taken in response to an event. In UML state diagrams, the actions on the transitions are assumed to be instantaneous. In FlexFlow most of the processing activity happens on the transition actions. Given the interactive nature of the applications, these usually take a very short time, but are not instantaneous.

Events

An event is a named message needing to get processed. In Internet applications, an event is usually a HTTP client request generated by a user pressing a hyperlink, button, etc. on a web page. It can also be an incoming Simple Object Access Protocol (SOAP) request or a Java Message Service (JMS) message. It can also be an event generated by another process such as a scheduler or another FlexFlow process. It can even be an event generated in the same FlexFlow process by a transition or a state entry action.

Guards

A guard is a set of conditions that need to be true before the action can be taken. Conditions are Boolean computations on the *context* of the business process and/or the parameters of the event. In general, the guards can be rules. In our implementation, an access control condition is always present in the guard. Thus, the action on a transition is taken only if access is allowed. If no access control policy is explicitly specified, the default access control mechanism is used.

Context

Context is data associated with a business process. It consists of

- The session information that includes information about the user including roles and permissions, and
- The data submitted by the user such as form entries and the data stored in the form such as

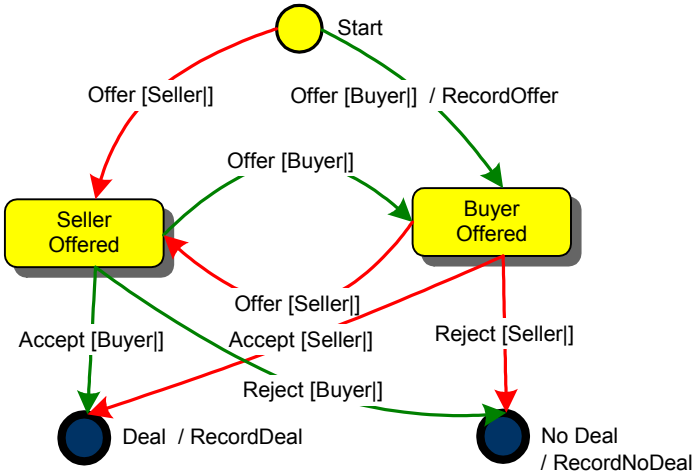


Figure 3: A simple statechart for bilateral negotiation

the identification of the process and the identification of the business object. For example, if a user is submitting a bid for an auction, the context would contain the username and roles, the amount of the bid. The event would include the number of the auction on which the bid is made. Also included in the context is more general information about the process such as auction start and end time, number of bids, etc.

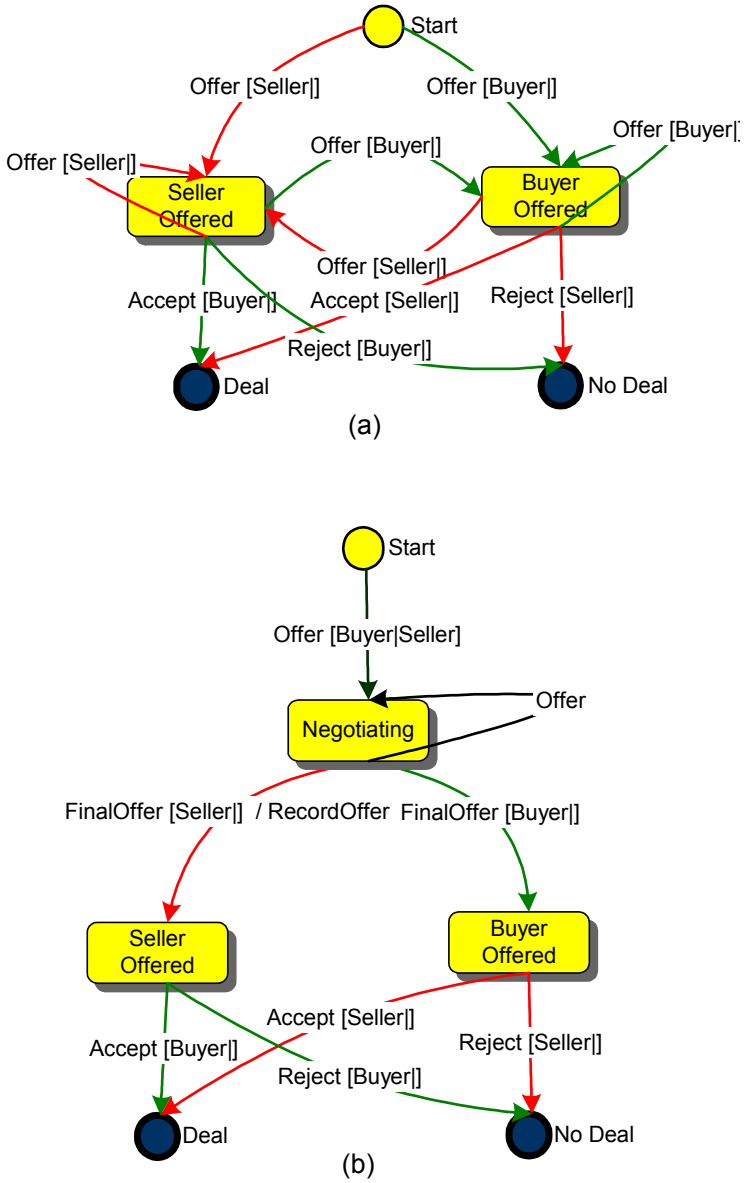


Figure 4: Two variations of the bilateral negotiation process.

This context can be referenced in guards as well as read and updated in actions.

Figure 3 shows a FlexFlow model for a simple bilateral negotiation process between a buyer and a seller. The top right transition shows that on the event "Offer" the action "RecordOffer" is taken. The guard checks that the user making the offer is the "Buyer". As the action for the other "Offer" transitions is "StoreOffer" we do not show it here for simplicity. There is no action corresponding to the "Accept" or "Reject" events. On entry to the final state "Deal" a "RecordDeal" action is taken.

Figure 4 shows two variations of the bilateral negotiation process shown in Figure 3. The process in Figure 3 forces the buyer and seller to alternate their bids, i.e. once a participant makes and offer, she has to wait for the other party to make a counter offer. In Figure 4 (a), the parties can improve their offers without waiting for a counter offer. In Figure 4 (b), the parties can make a final offer which forces the other party to either accept of reject the offer but does not allow them to counter offer. As is obvious from the process diagrams, the three variants of the business process reuse the code for just three actions "RecordOffer", "RecordDeal" and "RecordNoDeal".

The FlexFlow system particularly addresses following modifications of business processes:

Business Rules. Business managers can change the business rules of the processes by changing the business logic guards on the transitions. For instance, a business logic guard can check whether a particular RFQ needs approval or not.

Control Flows. If business managers change the transitions between the states or add new transitions they modify the control flow of the process. This way, business managers can for instance implement loops, skip activities or include new actions in the process.

Access Policies. The transitions include access control guards which check whether a user can perform an action. These access control guards allow business managers to determine what actions of the process a user can perform.

Approval Support. The FlexFlow system inherently supports the approval of process activities. It includes templates for several approval processes which can be used within a process state diagram. Furthermore, new approval process templates can be specified (for instance a template, which describes the process for 3 levels of approval).

The FlexFlow system also facilitates modifications of the web-application's user interface without changing the underlying business logic. The presentation layer of a web-application communicates by events with the FlexFlow engine. Therefore, this communication mechanism is independent from the rendering of the web-pages.

5. FlexFlow Process Execution

The FlexFlow system has an engine to manage the execution of business processes. When an event arrives, an event dispatcher figures out to which business process instances the event applies and invokes the engine for each. The engine, on invocation, retrieves the process instance or creates a new one if instance does not yet exist.

The engine selects the set outgoing transitions from the current state of the process that correspond to the event. For this set of transitions, it selects those where the guard evaluates to true. If more than one transition survives, the transition with the highest priority is selected. In case of ties, one of the transitions is chosen at random. The engine then executes the action corresponding to the transition selected (if any). If the action executes successfully, the engine updates the process instance's state based on the state machine transition traversed. If the action fails, an exception is raised.

An event can cause multiple actions. In addition to the action on the transition, there can be an exit action on the old state and an entry action on the new state. The FlexFlow engine processes all actions from one event within a single transaction scope. In other words, the system is left with either the effect of all the actions executed, or none. If any action fails, the whole transaction is aborted and the effects of the previous actions whose execution was initially or subsequently caused by the incoming event are rolled back. Only when all the actions succeed is the transaction committed. This prevents the process instance from ending up in an "unnatural" state.

5.1 Event Creation

Events can be created in the following two ways (both shown in Figure 6):

- **Interaction Controllers** handle external interactions with different types of clients including web browsers, mobile devices, and message queues. For example, a buyer requests on a web form for an RFQ to be aborted. This HTTP request is received by the interaction controller which converts it into an event.

- **Internal System Actions** can trigger events. For example, the RFQ abort action might create an RFQ Canceled event as the first step in informing all the quotes for the RFQ of the need for invalidation. Another example if of the scheduler sending an RFQ close event. The processing of these events run in the same transaction as the system action, but in a different thread of the engine.

5.2 Cascaded flows

Business flows can be sequenced or cascaded. For example, an RFQ process can be followed by bilateral negotiation on each of the selected responses. Implementation of cascaded flows is handled by used of blocks and different versions of FlexFlows. Format conversion commands are used between cascaded flows.

5.3 Coordination of FlexFlow Processes

Often, related business processes need to be synchronized. In FlexFlow, business processes coordinate with one another by triggering events. That is, an instance of one state machine can trigger an event to be processed in one or more instances of the same or different state machines. Events enable coordination of multiple parties. For example, it may be desired that the closing of an auction by a seller in an auction business process disable the ability of buyers to increase their bids in the bidding business process.

Another example for the need of process interaction is with a request for quotes (RFQ) where a buyer drafts and sends out an RFQ. A number of sellers then create responses (quotes) for the RFQ. Since the quote process itself is involved (the quote may in draft sate, awaiting approval, or being modified and resubmitted etc.), considering the buyer's process and the seller's process as different business processes simplifies their modeling and customization. This allows us to handle scenarios where one seller organization requires approvals before they submit quotes, while another does not. However, these different business processes need to be coordinated. For

example, a quote cannot be submitted before the RFQ is active. In addition, once the RFQ is closed, the sellers are not allowed to modify nor retract quotes.

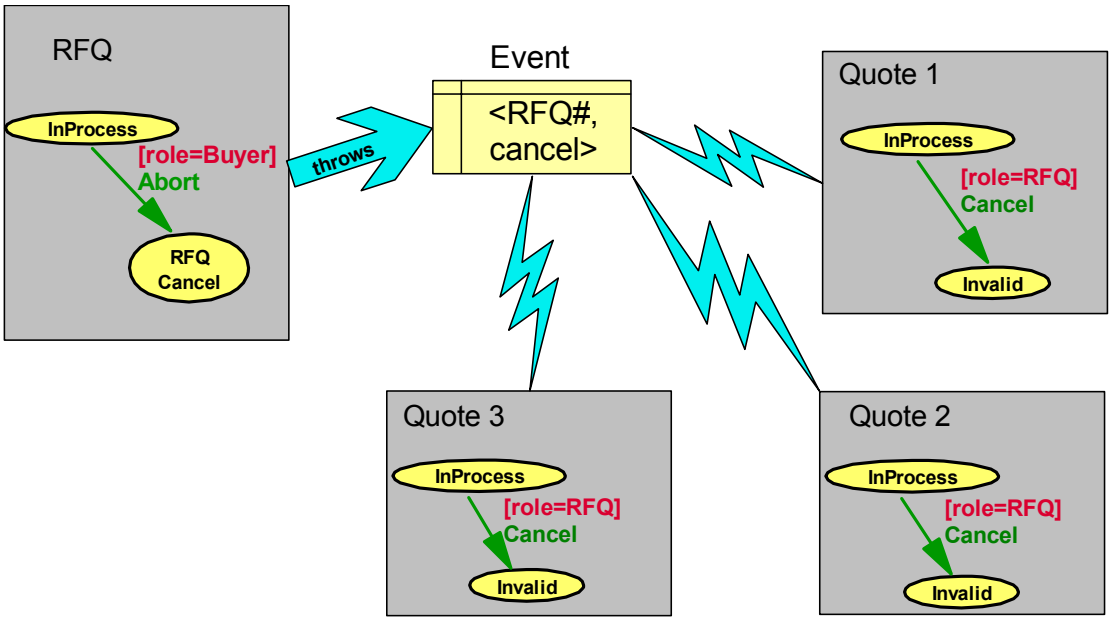


Figure 5: Coordination of RFQ Abort by Buyer with Seller's Quotes

Figure 5 shows the interaction of one RFQ machine instance with three corresponding quote machine instances. The coordination of the FlexFlow process instances is handled by a message sent from a source to any targets listening to that source. When a process is created, it registers to listen to other related processes. For example, when a quote for an RFQ is created, it is known for which RFQ it is a response. The quote registers to listen to the RFQ events. So, when the RFQ is canceled, its corresponding quote processes receives the cancel event and transitions the quote processes from the "InProcess" state to the "Invalid" state (i.e. the quote is no longer valid as the request for quotes has been cancelled).

5.4 The FlexFlow Event Handler

All events arrive first at the FlexFlow event handler. From the event handler's perspective, there are two classes of events (both of which are shown in Figure 6):

- Events triggered by interaction controllers simply get routed to the FlexFlow engine. This includes events working on existing instances and those creating new ones.

- Events triggered by existing process instances that need to be processed by all process instances listening for it. For example, all the quotes listening to their parent RFQ need to process events coming from the RFQ. To determine which instances are listeners, the router reads a Flow Instance Event Registry where quote process instances are registered to listen to the RFQ process to which they belong. The FlexFlow event handler will duplicate the event for all the listeners, routing each to the Flex Flow engine.

5.5 The FlexFlow Engine

The FlexFlow Engine, the heart of the FlexFlow run-time, receives targeted events from the event handler and executes the necessary actions. Figure 6 shows how the engine interacts with the other parts of the system to accomplish this processing.

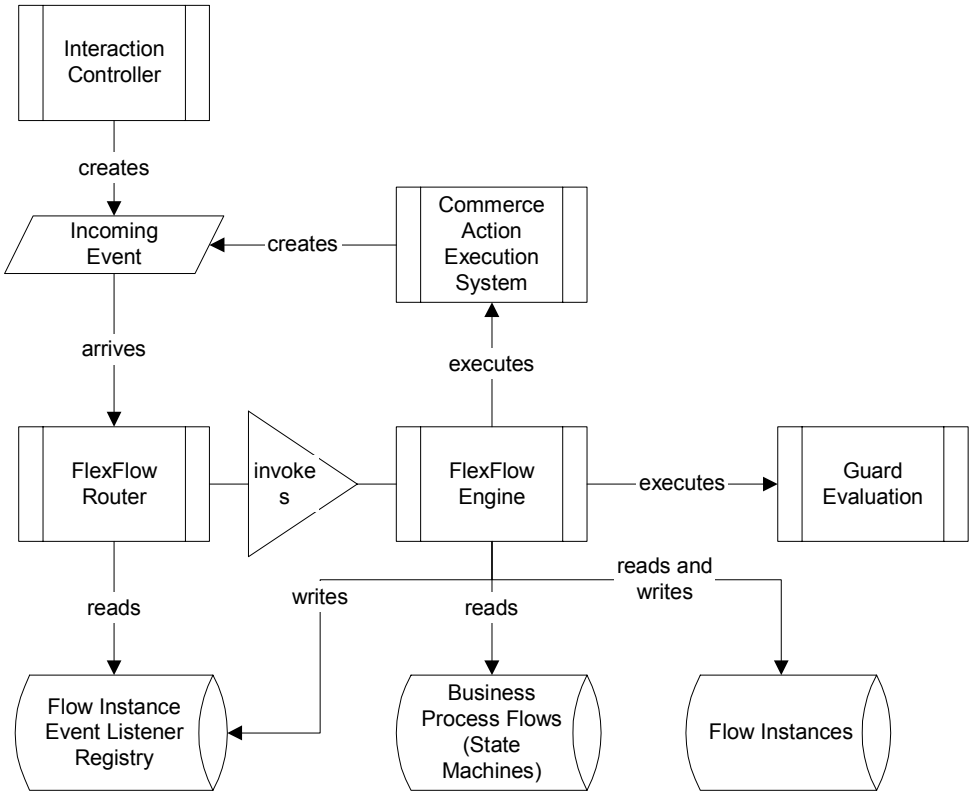


Figure 6: FlexFlow event handler and engine.

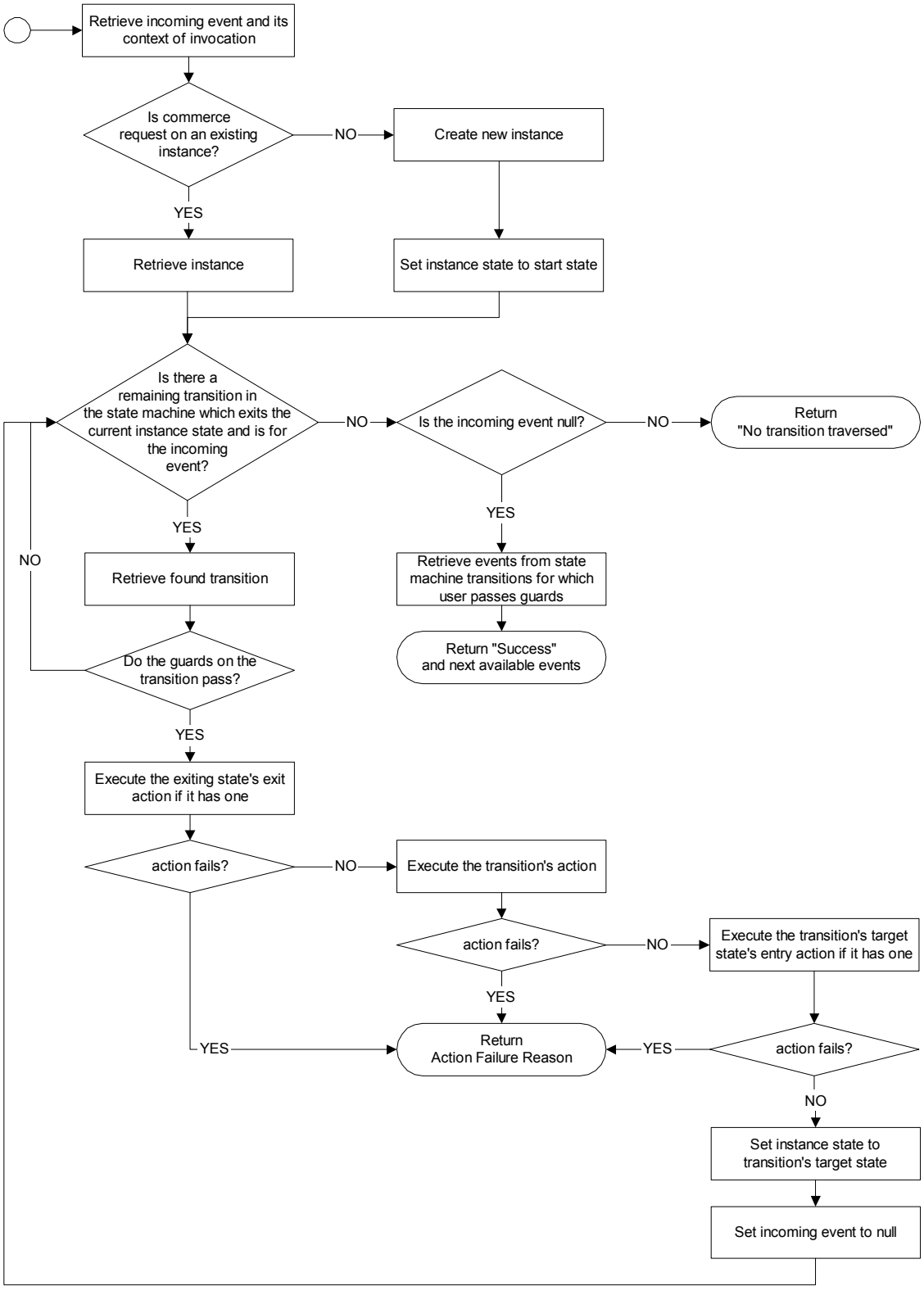


Figure 7. Flow chart showing engine execution of an incoming event.

When receiving an event, the FlexFlow Engine takes the following steps (as shown in Figure 7):

1. The incoming event is retrieved along with its context. The context is generated from the event including retrieving and marshaling incoming parameters and deriving of user and role information.

2. The engine determines whether the event applies to a new instance or an existing instance. Events with a target instance identified use an existing instance that the engine retrieves. For example, when an RFQ Abort event comes in it includes an identifier for the particular RFQ for which it is intended (i.e., the RFQ being aborted). Events without a target instance identified cause the creation of a new instance. The engine determines the state machine for which it creates an instance based on the business process being started. Once this new instance is created, it is set to the start state of its underlying state machine. The engine registers the instance for events from those instances from which it needs to know outgoing messages. For example, a quote instance is registered to listen to its parent RFQ.

3. The engine looks for a transition to take. First, it gathers those transitions in the instance's underlying state machine that:

    - exit the current state of the instance

    - have an event matching the event being processed

    - have guards which can be passed through; the engine calls the guard evaluation to check if a transition's guards are satisfied

4. If no transitions were found in the previous step, then:

    - If this was the initial pass through the previous step (i.e., the engine is not looking for null event transitions), the engine returns control to the caller with an error stating that no transition was traversed.

    - If this was not the initial pass through the previous step (i.e., the engine is looking for null event transitions), the engine retrieves all the transitions exiting the current state. For each, it calls guard evaluation to determine which the user can take next. The

engine returns control to the caller noting the processing succeeded and listing the next available events.

5. If the engine has come this far without returning, it has a transition that can be traversed. The first step is to execute the exit action on the current state if there is one. If this action fails to complete successfully, the engine returns control to the caller with the reason for failure.

6. The engine executes the action on the found transition. If this action fails to complete successfully, the engine returns control to the caller with the reason for failure.

7. The engine looks for an entry action on the transition's target state. If one exists, the engine executes it. If this action fails to complete successfully, the engine returns control to the caller with the reason for failure.

8. The engine updates the instance's current state to the transition's target state.

9. In order to process any automatic (null event) transitions exiting the new current state of the instance, the engine then sets the incoming event to null and returns to step 3.

During these steps, as shown in Figure 6, the engine uses data from the state machines (also know as business process flows) and the instances and updates data for the instances as well as the registry of which instance are listening to which.

6. User Interaction with FlexFlow

We have observed that a common practice for designing web sites, such as e-commerce sites, is to first mock-up the flow of web pages for user interactions, and then to use this flow to drive the development of application logic. This practice works when the business process is simple and when only one party (the user) is interacting with the system. However, this design practice does not scale to complex business processes, especially where multiple parties are participating in the business process, such as two users in a bilateral negotiation or a buyer and multiple sellers in an RFQ, along with schedulers for timeouts etc. Another drawback of this design practice is that process logic gets embedded both in web pages and application code further complicating any modification of the business process.

6.1 Process Reflection

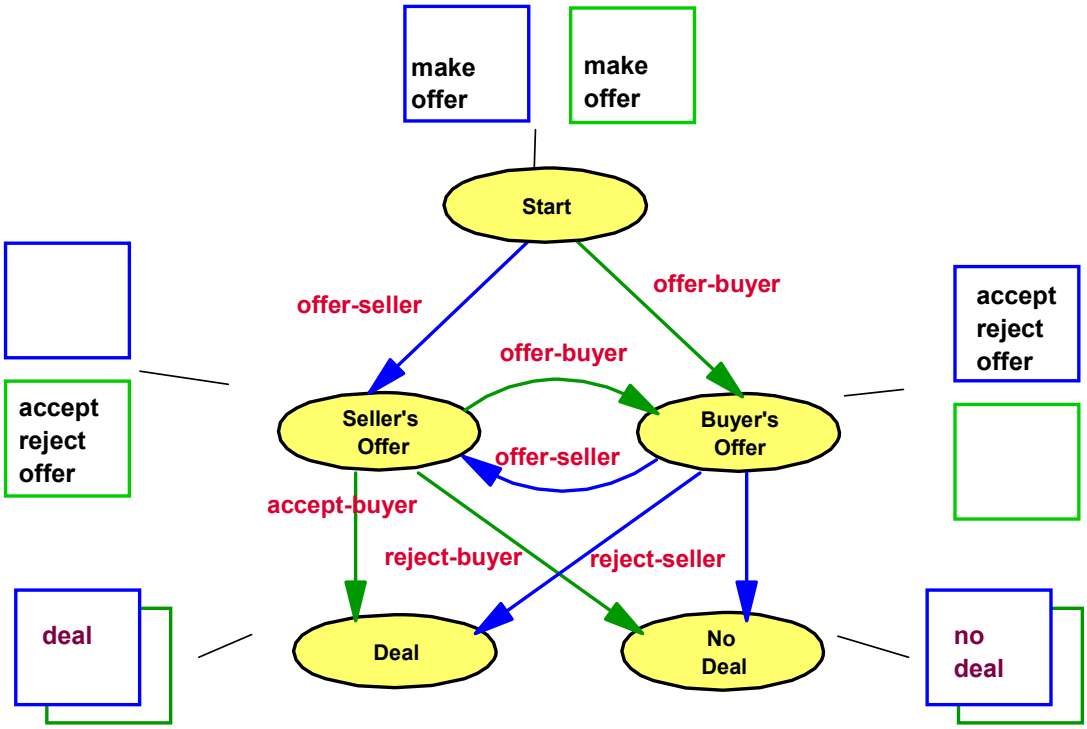


Figure 8: Controls on web forms for user interactions are created using FlexFlow. The blue outlined page is for the seller, the green for the buyer. The text in black corresponds to buttons on the forms.

The FlexFlow process model has sufficient information for deriving user interactions from the statechart. The process reflection mechanism of FlexFlow allows clients to discover or query process information at run time. This mechanism can be used to drive the user interface or the future user interaction. Thus, with FlexFlow, the design practice is to first design the process and then to automatically derive the flow of user interactions. As the user interaction information is added dynamically to the web pages at run time, the modifications of the business process get automatically reflected in the web pages.

Process reflection allows users to query a list of actions that are valid for a given user role at the current state of the business process. At each given state, the FlexFlow system knows the next possible set of actions a particular user can perform using the guards on all the outgoing transitions. Thus, FlexFlow can provide relevant information for the rendering of the user output (i.e it can determine whether buttons should be enabled or disabled). If web-designers use this

reflection mechanism, web-pages can to be shared among different process versions and it reduces the effort for modifying FlexFlow processes [13].

We illustrate this in Figure 8 for a simple bilateral negotiation. There is web page for each state for each user where the user can take an action. The seller's pages are outlined in blue and the buyer's pages are outline in green. At the start state, either party can make an offer to start the negotiation so the both the buyer's and seller's page show a button (or other control) for making an offer. If the seller makes an offer, the process moves to the "Seller Offered" state and the page for the seller will show no buttons (corresponding to this instance of the bilateral negotiation) while the buyers page will display the options to make a counter offer or to accept or reject the current offer. As the controls are generated dynamically via reflection on the process model, when the process is changed, for example as shown in Figure 4, the controls on the web pages will show the correct set of actions without any rewriting.

End users may interact with the system using a web. Actions requested by end users are passed to the FlexFlow engine. The engine processes client inputs depending on the process state machine, the current state of the process instance and the role of the user. The FlexFlow system also provides as output to the user interface system, the list of actions that are valid for a given role at the current state of the business.

Each state corresponds to a view for each of the participants in the process (when the participants are software agents, these views would be outgoing messages). The FlexFlow statechart clearly defines for each state in the process the next legal set of actions for each participant. For a user (with a set of roles) of a business process, the set of outgoing transitions from the current state of the process, which have an access control guard which will compute to true for a role of the user, completely define the legal set of events that the user can send to the process.

The action corresponding to a transition specifies the view to be used. We allow for relationships between states and views displayed to users. The state may specify a default view to be used, which can be specified in the process state diagram. If the process state machine enters a state and the executed action does not specify a view, then the default view of the state is being used.

If the client requests a view directly, the client view action can correspond to different transitions depending on which state the process is in. For example, a "query order status" request returns either a "display pending order" or a "display completed order" view name, which causes one of two different pages to be displayed to the end user. This could be alternately be implemented by the “view” action correspond to a “display completed order” transition at the completed order state and to the transition “display pending order” at the pending order state.

7. Visual Modeling

The FlexFlow engine uses a XML representation of the process definition. To allow business managers easily create and change FlexFlow processes, we extended popular COTS modeling tools. Since state charts are a part of the popular UML notation, a number of graphical tools are available. For managing FlexFlow processes, we have added extensions to both Microsoft Visio® and Rational Rose®. Therefore, business managers can use a familiar modeling environment, which provides following key functionalities:

- Easy-to-use modeling interface for creating or modifying business processes by changing, adding, and/or removing states and transitions from the process state diagram.
- XML generation of the process definition based on the process state diagram.
- Import / Export of the XML process definition
- Management of different versions of process state machines.
- Simulation of the FlexFlow processes

The states and transitions of FlexFlow state diagrams have additional attributes like response views, additional guard properties or priority settings. Business managers can import and export XML process definitions via a file or a web-service.

Different versions of a business process can be maintained based on membership at the organization level. Versions can also be selected based on the mode of interaction, such as device, browser, and messaging. Figure 9 shows the default version of a RFQ process. By specifying new flows, the modeling tool allows to manage several variations of a RFQ process. This way, business managers can model and maintain several RFQ processes (for instance a

"Normal RFQ" process, and a "Fast RFQ" process, which is a more compact version of the normal RFQ process).

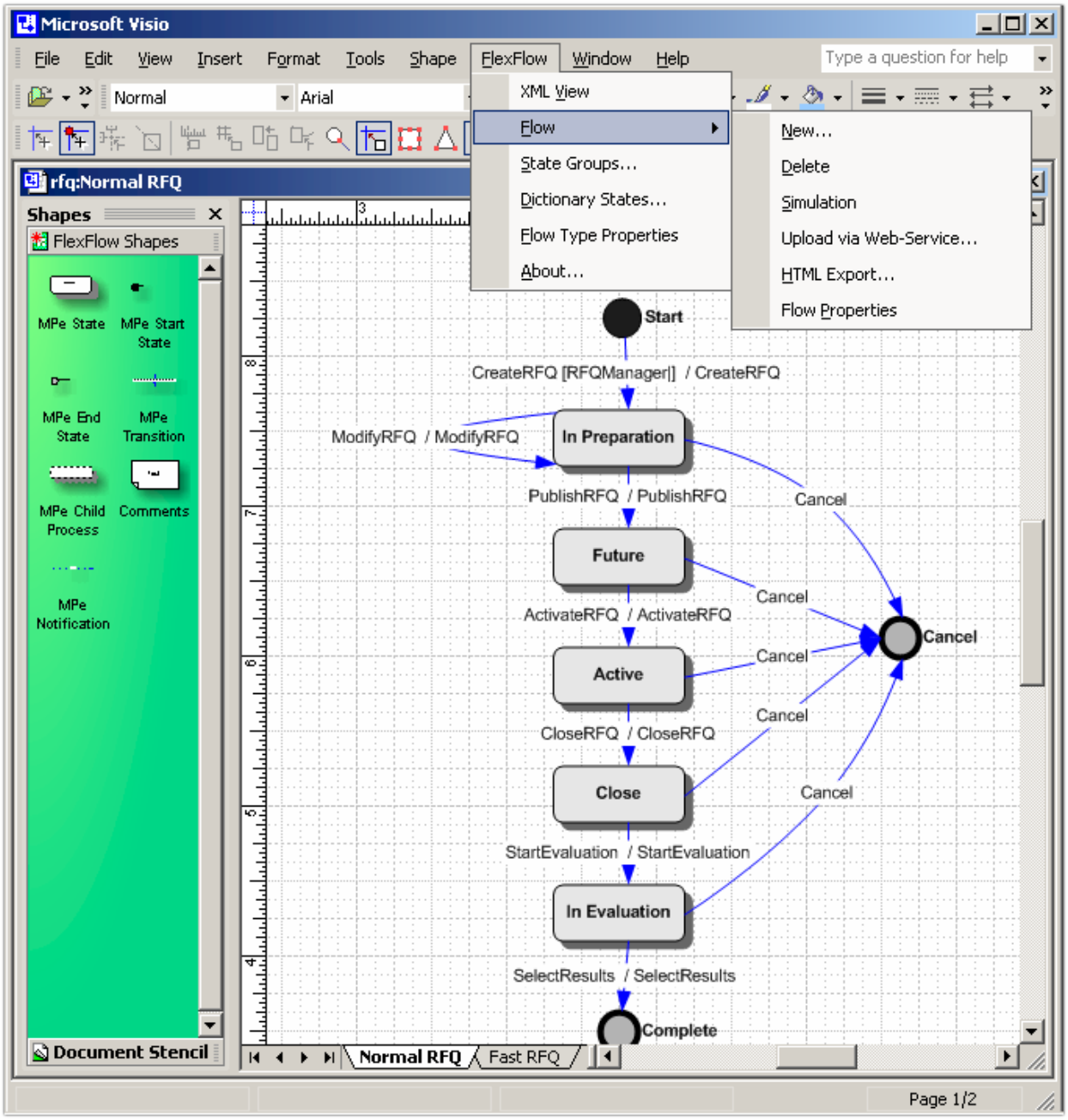


Figure 9: Visio Modeling Tool for FlexFlow

Versions can be selected based on the membership at the organization level or on the mode of interaction, such as device, browser, and messaging.

The modeling tool gives the business manager the possibility to simulate the business processes with existing web-pages or mock-up pages.

8. Process Simulation

In typical web-application users can navigate to a limited number of web-pages based on the action she takes at the active input location. The number of possible navigation paths can be very large in a complex graphical user interface, but the number is finite and the options usually are known. User interfaces also must stay in sync with the underlying business process. Therefore, process state diagrams reflect the navigation paths of the user at a high level of abstraction.

Process state diagrams can be used to explore hypothetical process models and user interface concepts based on the understanding of the requirements. Users and developers can study a process state diagram to reach a common vision of how the user might interact with the system to perform a task. The business process, business rules and the user experience can be incrementally and iteratively optimized by simulating the business process with user scenarios without implementing the business logic. This way, conflicts between the business process and the user interface can be easily discovered.

Process state diagrams capture the essence of the user-system interactions and task flow without getting one bogged down too soon in specifying the details of web-pages or data elements. Users can trace through a process state diagram to find missing, incorrect, or superfluous transitions, and hence missing, incorrect, or superfluous requirements.

The FlexFlow modeling tool includes a simulation component which allows developing horizontal prototype which displays the facades of user interface screens from the web-application, possibly allowing some navigation between them, but they do not show real data or contain little or no real functionality. The information that appears in response to a client requests is faked or static, and report contents are hard-coded. Nevertheless, the simulation component allows a process-oriented navigation through the web-application. It allows users to change the status of the current process by selecting one of the available actions of the simulation panel. For the simulation, we can include web-pages of existing web-solutions or new web-pages, which can be instantly created and modified. Figure 1 shows the simulation of the RFQ process. The

buttons in the simulation panel at the bottom of the screen show the available navigations paths based on the RFQ process state diagram.

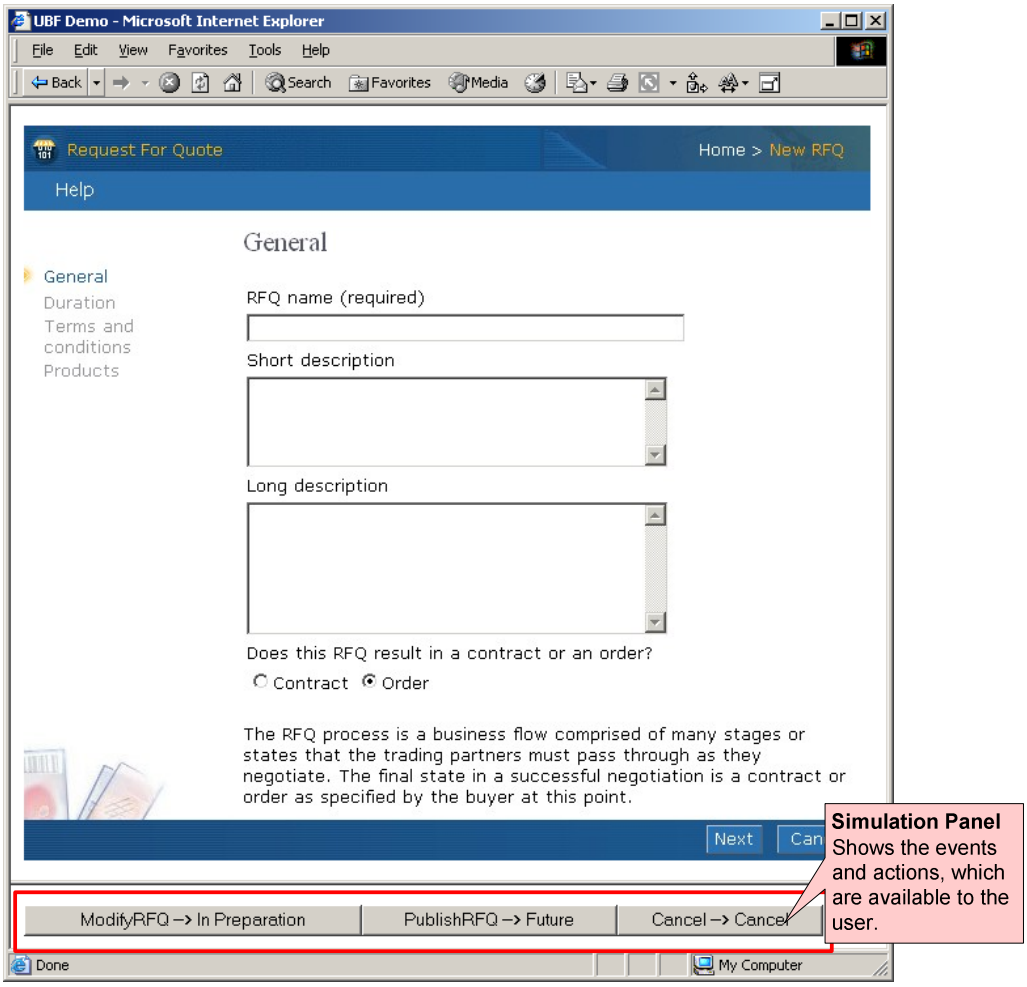


Figure 10: Simulation of FlexFlow Processes

Note, that not all page flows are represented by the control flows of a process. For instance wizards like in Figure 10, or other UI facilitators which have a predefined sequence of processing steps, are implemented solely in the presentation layer and have no impact the process itself. UI components of the presentation layer use the process reflection mechanism to determine functionality, which should be available to the user.

This type of simulation is often enough to give the users a feeling for the web-application and lets them judge whether any functionality is missing, wrong, or unnecessary. The simulation

prototypes represent the concepts to the developers of how the business process might be implemented. The user's evaluation of the prototype can point out alternative courses for a business process, new missing process steps, previously undetected exception conditions, or new ways to visualize information.

By modeling and executing business processes as state machines, these processes can be modified with minimal changes to the underlying implementation of the business processes. A commerce function can be modified simply by reconfiguring its corresponding process state diagram.

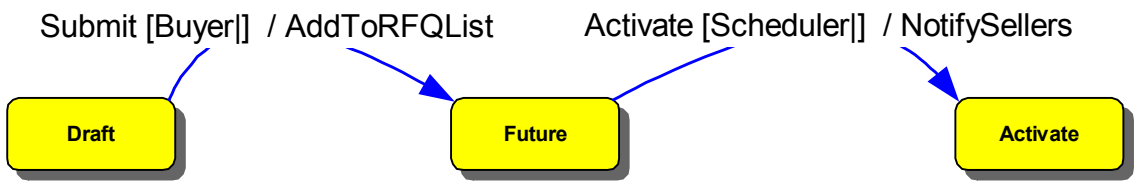
9. Experiences with implementation

Over the past two years, we have deployed two generations of the FlexFlow system in commercial B2B e-commerce systems, first in IBM's WebSphere Commerce Suite MarketPlace Edition ® (WCS MPe), and then in IBM's WebSphere Commerce Business Edition ® (WCS BE).

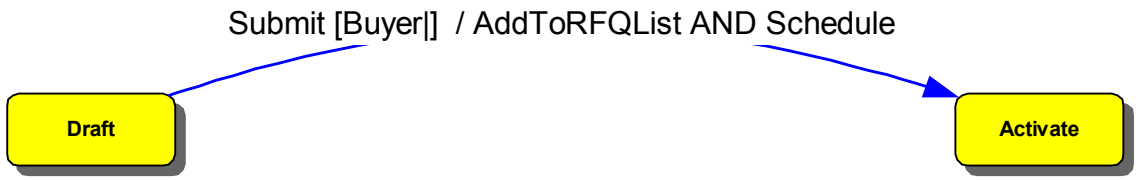
From our experience from the first version several enhancements went into the follow-up. We found the need for multiple actions to ease customization as well as to simplify modeling. Integrating with an access control system further improved flexibility while enhancing richness while caching the business processes in memory provided performance improvements.

9.1 Multiple Actions

In WCS MPe each user request could only cause the traversal of one transition, as we did not implement UML's *null* events. This omission hindered customization as simplifying the business process meant rewriting actions. This implication is best shown by example. In a piece of a Request for Quote (RFQ) process, a buyer first submits an RFQ with a particular start time. The scheduler will then activate the RFQ when the start time is reached. This would like look like:

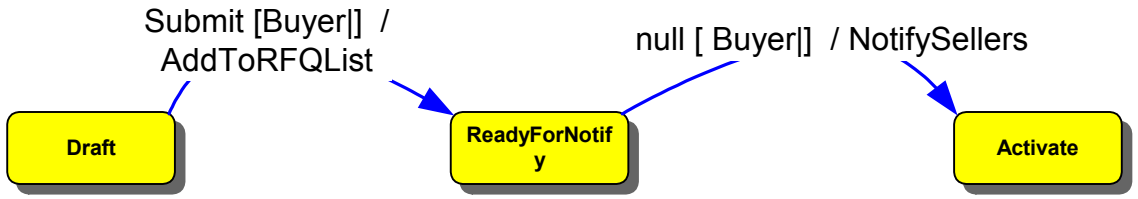


However, if the company using this business process decides it wants to remove the notion of RFQs that start in the future and, instead prefers all RFQs start when they are submitted, they would need to change the state machine to something like:

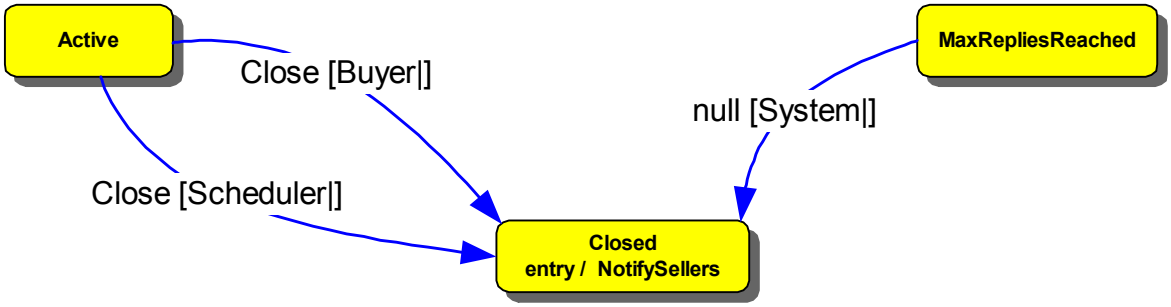


Notice that the two actions AddToRFQList and Schedule cannot be put on the same transition, so a new action needs to be written that will perform both actions. The ability to re-configure a business process without re-writing the application logic is lost.

Implementing *null* events in WCBE alleviated the need to re-write. A transition with a *null* event is automatically taken if its guard is satisfied. *Null* events enable us to simplify the business process using the original actions:

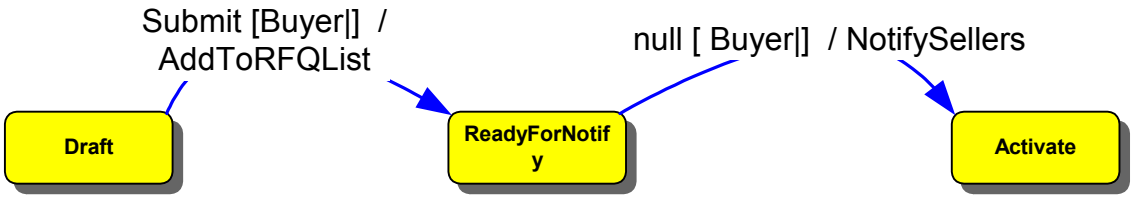


Another improvement made in our second version of FlexFlow is the use of UML entry actions and exit actions. Commonly, reaching a certain state requires an action no matter how the state is reached. For instance, sellers need to be notified when an RFQ is closed. The notification needs to be sent out whether a buyer decides to close an RFQ, the Scheduler closes an RFQ when its end time has been reached, or the RFQ is closed automatically because a threshold on the number of replies has been satisfied. While we can model the notifications with *null* events, it is extremely simplified with one entry action.



With one entry action, we can send out a notification no matter what caused us to reach the *closed* state.

Allowing an incoming event to cause multiple actions through entry and exit actions as well as *null* events raised a new issue. In the initial FlexFlow implementation, any error occurring in an action caused the action to be undone and the transition not to be taken. Leaving processes in their initial state enabled user to retry them. Now, with multiple actions, if we only rolled back one the transition with the error, we could end up in an "unnatural" state. For example, let's revisit the section of an RFQ process:



If the submit event transition completes successfully but the *null* event transition does not, the process must not be left in the ReadyForNotify state. So, the FlexFlow engine ensures the atomicity of the processing of all actions from one event. In other words, the system is left with either the effect of all the actions executed, or none. The engine accomplishes this by executing all the actions within the same database transaction. If any action fails, the whole transaction is aborted and the effects of the previous actions whose execution was initially or subsequently caused by the incoming event are rolled back. Only when all the actions succeed is the database transaction committed.

9.2 Access Control

Originally, FlexFlow used roles to identify those users who were allowed cause a transition in th process to be taken. In its second release, FlexFlow was integrated with an access control system. A transition guard now identifies an action group to which the transition belongs. The administrator of an e-commerce system then defines access control policies that specifies which users can perform a particular action group on a particular business object based on their relationship to that object (e.g., creator).

In addition to a greater richness for access control, this integration provided a new layer of abstraction. Formerly, administrators were forced to create access control policies at the action level. For example, they might say users with the buyer role can cancel an RFQ. However, a different policy on a business object may be required based on the current state of the business process or by the state in which an action will put the process. By allowing administrators to specify action groups on transitions, all these requirements are met. Now, canceling an auction can be limited to sellers when the auction is in the “Future” state and not the “Active” state.

9.3 Caching Business Processes

During development of e-commerce systems, business processes change often. However, once an e-commerce system is running, the processes change quite infrequently. By caching the business processes in memory, FlexFlow now avoids reading database records to determine the details of the business process.

9.4 Workflows in FlexFlow

FlexFlow allows workflow type processing. A typical example is a business process for multilevel approvals. Workflow type processing is exhibited as at each level of approval, a new set of users become approvers. Figure 11 is the state diagram from such a process. After FlexFlow handles a submit event, the process will be in a Pending state, unless the value is less than $100 in which case it is automatically approved. The CreateRequest sets up the first level of approvers. Each time an approver approves a request, the ProcessApproval action will increment the approval records level and set up the approvers for that level. Once the correct approval record is reached, the request becomes approved.

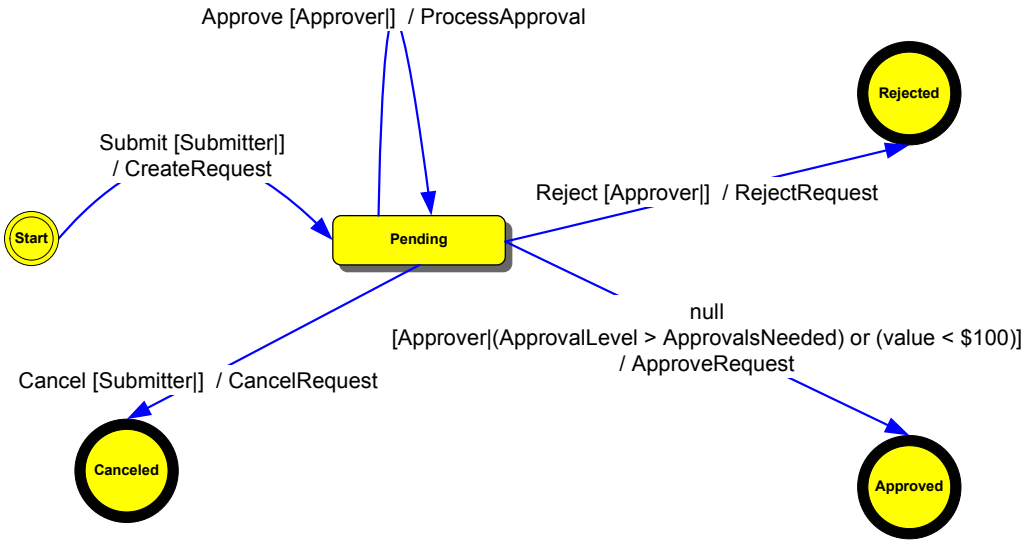


Figure 11: Multilevel Approval

The approval system is a special business sub-process. One can model the approval system with its own set of one or more state transition diagrams, or directly within the business processes that use approvals. To model with the former, one can use messages between the business process and the approval sub-process (see "Coordination of FlexFlow Processes"). When modeling with the latter, transitions and states, such as those shown in Figure 11, are put into that process' FlexFlow state diagram.

One could code the actions and model the process to be integrated with an external approval system such as a Lotus Notes-based approval system. Similarly, comprehensive workflow systems can be integrated within the state machine. Messages coming from external systems get translated into FlexFlow events. Actions within FlexFlow can send messages to these external systems.

10. Conclusions

Web-applications are difficult to build with traditional workflow management systems. In this paper, we presented an approach for managing web-based business processes. We introduced a state machine based model for the specification of business processes. Since e-commerce

environment are very dynamic and change frequently, we argued that a descriptive model in which business processes are represented “as-is” and “as-to-be” models are advantageous compared to workflow management systems, where separate models are used. We have shown the FlexFlow system which supports the modeling, simulation and execution of process state machine.

Further problems we want to consider in the future include the management of hierarchical states as well as the concurrent execution of FlexFlow processes:

- FlexFlow state machines can be denoted as super-states, whereby each super-state corresponds to a state machine. We want to extend our model to allow states to be nested an arbitrary number of times. Nested states would also allow a notional simplicity for handling duplicate transitions and interrupts.
- Concurrent process state machines sometimes need to be synchronized with each other. Web-applications with many business processes demand a possibility to start business processes together, run them independently until a certain state and finally re-synchronize them. Forks and joins will allow us to specify more complex transitions to allow this kind of synchronization.
- Besides forks and joins, we want to include synch vertex in our process model in order to synchronize concurrent regions in a process state machine. A synch vertex is different from a state in the sense that it is not mapped to a Boolean value (active, not active), but an integer. It is used in conjunction with forks and joins to insure that one region of a state machine leaves a particular state or states before another region can enter a particular state or states.

References

[1] Wil van der Aalst, T. Basten, Inheritance of Workflows - An approach to tackling problems related to change, http://wwwis.win.tue.nl/~wsinwa/wf_inh.ps, 2001.

[2] Ahmed K. Elmagarmid and Weimin Du. Workflow Management: State of the Art vs. State of the Market. In *Advances in Workflow Management Systems and Interoperability*, pages 1-17, August 1997.

[3] Grady Booch, Ivar Jacobson, and James Rumbaugh, The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[4] Dragos A. Molescu, Ralph E. Johnson, A Micro Workflow Framework for Compositional Object Oriented Software Development, OOPSLA, 1999.

[5] Ellis, C.A., Nutt, G.J., Modeling and Enactment of Workflow Systems, 14th International Conference on Application and Theory of Petri Nets, 1993

[6] Genrich, H.J., Predicate/Transition Nets. In: Advances in Petri Nets, 1986, Springer, LNCS 254

[7] Georgakopolaus, Diimitiros and Hornik, Mark, An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure, Distributed and Parallel Databases, 3, 119-153, 1995.

[8] Hammer, M., Champy, J., Reengineering the Cooperation, A Manifesto for Business Revolution, New York, 1993

[9] Harel D., Statecharts: A Visual Formalism for Complex Systems, Science of Computer Programming, Vol. 8, 1987.

[10] D. Harel et. al., "STATEMATE: A Development Enivronment for Complex Reactive Systems," IEEE Transactions on Software Engineering, April 1990.

[11] Harel D., Politi, M., Modeling Reactive Systems with Statecharts, McGraw-Hill, 1991.

[12] Harel, D., On Visual Formalisms, Communications of the ACM Vol.31 No.5, 1988

[13] Ian Horrocks, Constructing the User Interface with Statecharts, Addison-Wesley, 1999.

[14] Jablonski, S., Bussler, C., Workflow-Management, Modelling Concepts, Architecture, and Implementation, International Thomson Computer Press, 1996

[15] Kappel, G., Lang, P., Rausch-Schott, S., Retschitzegger, W.: Workflow Management Based on Objects, Rules, and Roles, IEEE Bulletin of the Technical Committee on Data Engineering, Vol. 18/1, March 1995, pp. 11-17.

[16] Leymann, F., Altenhuber, W., Managing Business Processes as an Information Resource, IBM Systems Journal Vol.33 No.2, 1994

[17] Raul Medina-Mora, Harry K.T. Wong, and Pablo Flores. ActionWorkflow as the enterprise integration technology. Bulletin of IEEE Technical Committee on Data Engineering, 16(2):49-52, 1993.

[18] Mohan, C.: State of the Art in Workflow Management Research and Products, SIGMOD, Montreal, Canada, 1996

[19] Mohan, C., Recent Trends in Workflow Management Products, Standards and Research, NATO, 1997.

[20] Peter Muth, Jeanine Weissenfels and Gerhard Weikum, What Workflow Technology can do for Electronic Commerce, Current Trends in Database Technology, Idea Group Publishing, 1998

[21] Oberweis, A., Modeling and Execution of Workflows with Petri-nets, Teubner, 1996

[22] Reisig, W., Petri Nets: An Introduction, Springer, 1985

[23] Reuter, A., Schwenkreis, F., ConTracts - A Low-Level Mechanism for Building General-Purpose Workflow Management Systems, IEEE Computer Society, Bulletin of the Technical Committee on Data Engineering, 18(1):4-10, 1995

[24] Marek Rusinkiewicz and Amit Sheth. Specification and Execution of Transactional Workflows. In W. Kim, editor, Modern Database Systems: The Object Model, Interoperability, and Beyond. Addison-Wesley, 1994.

[25] Simon, E., Kotz-Dittrich, A.: Promises and Realities of Active Database Systems, International Conference on Very Large Data Bases, Zurich, 1995,

[26] J. Sprinkle, C.P. van Buskirk and G. Karsai, Modeling Agent Negotiation, Proceedings of the IEEE Systems, Man, and Cybernetics Conference, October 2000.

[27] Tsai J.J.P., Yang, S., Bi, Y., Smith, R., Distirbuted Real-Time Systems, John Wiley and Sons Inc., 1996

[28] Unified Modeling Language Specification, version 1.4, http://www.omg.org/technology/documents/formal/uml.htm, 2001

[29] Weißenfels, J., Wodtke, D., Weikum, G., Kotz Dittrich, A., The MENTOR Architecture for Enterprise-wide Workflow Management, Workflow and Process Automation in Information Systems, 1996

[30] Workflow reference model. Technical report, Workflow Management Coalition, Brussels, Brussels, 1994.