On Demand Business-to-Business Integration

Liangzhao Zeng[1], Boualem Benatallah[1], and Anne Ngu[2]

[1] School of Computer Science & Engineering, University of New South Wales
Sydney, NSW 2052, Australia
{zlzhao, boualem}@cse.unsw.edu.au
[2] Telcordia Austin Research Laboratory
Telcordia Technologies,
106 East Sixth Street, Littlefield Building, Suite 415, Austin, TX 78701 U.S.A
angu@research.telcordia.com

Abstract. In order to support global competitiveness and rapid market responsiveness, virtual enterprises need to efficiently integrate different organization's business processes to provide customized services. Currently, most of the integration solutions are case-based, which have high setup cost and involve time consuming low level programming. Cross-enterprise workflow that is able to streamline and coordinate business processes across organizations in dynamic Web environment provides a low cost and flexible solution. We develop an agent-based cross-enterprise Workflow Management System (WFMS) which can integrate business processes on user's demand. In our approach, business processes are wrapped by service agents. Based on the requirements of users, the integration agent contacts the discovery agents to locate appropriate service agents, then negotiates with the service agents about the task execution. A cost model is proposed which allows the integration agent to update execution plan and integrate service agents dynamically.

1 Introduction

The ability to efficiently and effectively integrate business processes(e.g., order procurement, customer relationship management, finance, human resources, supply chain and manufacturing) on the Web is a critical step towards the development of the new on-line economy driven by B2B E-commerce. Existing enterprises would form alliances and integrate their processes to share costs, skills and resources in offering *Virtual Enterprises* [Geo99]. Leading organizations are embracing business-to-business integration(B2Bi) today, realizing the enormous competitive advantages that this model provides through faster time to market, reduced cycle times and increased customer service. The return on investment(ROI) for B2Bi is high–often 5 to 15 times the investment or more, with a pay back period measured in months, not years. An example of a B2Bi is a financial management system that uses payroll, tax preparation, and cash management as components. The component processes might all be outsourced from business partners.

However, current B2Bi solutions are still ad-hoc, time-consuming and require enormous effort of low-level programming. This problem is aggravated by the added degree of dynamism, unpredictability, and distribution of the Web. B2B E-commerce requires more flexible integration techniques. More importantly, the fast and dynamic integration of business processes is an essential requirement for organizations to adapt their business practices to the dynamic nature of the Web. B2Bi requires the ability to efficiently discover and exploit business services in a dynamic and constantly growing environment. It also requires the capacity to dynamically establish relationships among business processes. Moreover, where there are multiple organizations, there are likely to be multiple IT groups, multiple pools of users, multiple decision makers and independent set of goals and requirements. No matter how important B2Bi is to these organizations, the need to maintain autonomy and independently control over their own business processes is likely to be of even greater importance.

With respect to meeting the requirements of B2Bi, we identified two relevant emerging technologies: *workflow* and *agent* technologies. Workflow Management Systems (WFMSs) have achieved considerable improvements in critical, contemporary measures of performance, such as cost, quality of service, and speed by coordinating and streamlining complex business processes within large organizations (e.g., health-care, education, banking and finance, manufacturing, and communication). Indeed, an increasing number of organizations have already automated their internal process management using workflows and enjoyed substantial benefits in doing so. Since the success of using workflow technology in automating internal business processes, there is a trend to apply workflow technology to automate external business processes between the enterprises and their trading partners. Agents have emerged recently as an important paradigm for organizing many classes of distributed applications [HS97]. Agents are sophisticated software entities with a high degree of autonomy. They are charged to achieve certain tasks in collaboration with their peers. Agents may be incorporated into an existing application in order to add new functions or customize the execution of existing functions according to specific requirements, operating conditions or observations of user behavior. Agents' abilities on conducting semantic conversation support more flexible and complex interactions among trading partners. An agent-based framework will cover properties such as autonomous, pro-activeness, cooperation, adaptiveness and multi-party negotiation. These properties make agent technology one of the most important candidates for providing interoperability and interactions in volatile, dynamic and cooperative environments.

In order to address the problems of B2Bi, the *AgFlow* prototype has been developed [ZNBO01]. Our approach is mainly motivated by the fact that for dynamic and constantly changing business processes, there is a need to integrate business processes just in time on user's demand. The philosophy behind our agent-based approach is that B2Bi craves the workflow technology, but current workflow technology is unable to cope with pro-activeness, dynamic interactions

(negotiation), conversational semantics and component autonomy that is what agent-based systems can provide.

The remainder of this paper is organized as follows. In Section 2, we give a brief overview of the AgFlow system. In Section 3, we discuss how to select a workflow execution plan. In Section 4, we discuss dynamic B2Bi. In Section 5, We present our current implementation. A scenario of using the AgFlow is reported in Section 6. Finally, we discuss related work and conclude the paper in Section 7.

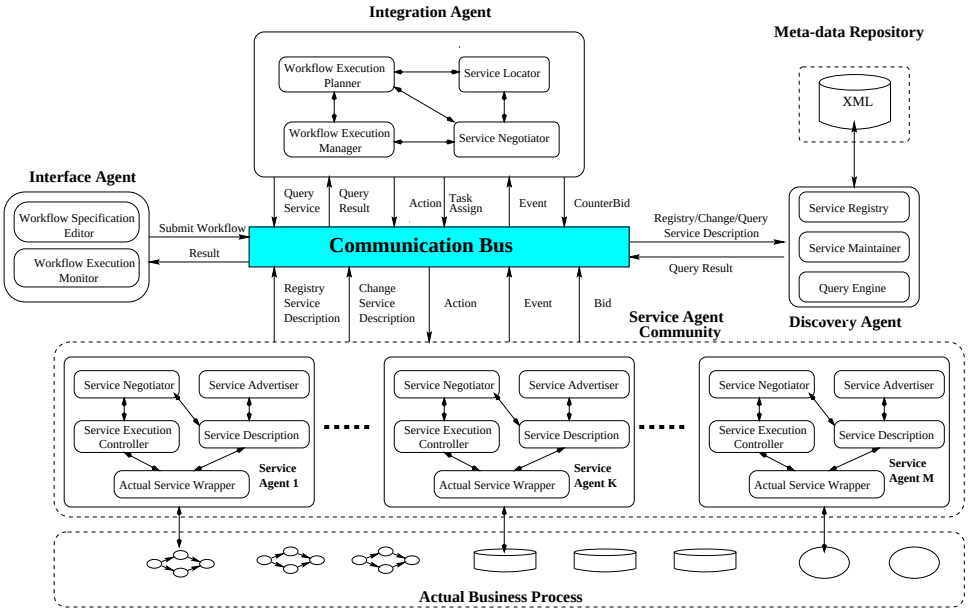

Fig. 1. AgFlow System Architecture

2 The AgFlow: An Overview

The detailed architecture diagram of the AgFlow is presented in Figure 1. There are four types of agents in the system, namely *interface agent*, *service agent*, *discovery agent*, and *integration agent*. In the AgFlow, B2Bi is regarded as cross-enterprise workflows which involve multiple enterprises' business processes. For each workflow instance, these four kinds of agents dynamically construct an agent community to execute the workflow.

Interface Agent The interface agent provides an interface which allows users to access the AgFlow. There are two types of users in the AgFlow: process composer and end user. Process composers can define workflow schemas. End

user can create and start workflow instances, control and monitor execution of workflow instances. In the AgFlow, a workflow schema is defined by a UML statechart diagram. The interface agent provides tools which allow process composers to draw UML statechart diagrams. Basically, there are two scenarios to define workflow schemas: defining workflow schemas from scratch; reusing existing workflow schemas to compose more complex ones. In the first case, we propose the following methodology:

1. **Defining Tasks.** Firstly, all the tasks in workflow schema are identified. Task is the basic element for workflow (it is the same as activity in WFMC model); it might be a fully automated business process such as booking a flight ticket online. A task is defined as a statechart which has input data, output data, states and transitions.
2. **Composing Tasks into Workflows.** In this step, tasks are regarded as states to be assembled into a workflow statechart. These can be done by specifying control flow among the tasks using transitions, fork or join.

In the second case, existing workflow schema statecharts are regarded as tasks to build more complex workflow schemas. After process composers define a workflow schema, end users can generate an XML document based on the workflow schema. In the XML document, end users provide values of the parameters that are needed to select service agents and execute the workflow. The XML document will be submitted to the integration agent to execute the workflow.

Service Agent In order to abstract business processes from their physical organizations, the service agents are developed to wrap them using a uniform process model. In the uniform process model, a UML statechart diagram represents a business process. Statechart diagrams can be described in XML. Figure 2 gives an example of XML document. Although being wrapped by service agents, the actual business processes do not need to be modified and still maintain their autonomy.

Discovery Agent In a Web-based environment, the service space is large and highly dynamic. An important issue to tackle is how to efficiently manage the potentially vast amount of available services. For this purpose, we use a discovery agent. The discovery agent allows each service agent to register its location, properties and service description in a meta-data repository. The meta-data repository contains meta-data that describe, among others, the meaning, type, content, capability and location of the service agents. The meta-data is used to locate service agents. It is generated when the discovery agent receives advertisement messages from service agents. Service agents construct the advertisement messages in XML. Figure 3 shows an example of advertisement message. Note that every service agent must advertise itself before participating the AgFlow. By searching the repository, the discovery agent can answer query about which service agents can execute a particular task.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE BusinessProcess SYSTEM "business_process.dtd">
<BusinessProcess>
  <ServiceID ID="001" Name="Flight Ticket Booking"/>
  <Service>
    <States> Request,Reserved,Booked </States>
    <Transitions>
      <Transition>
        <From>Request</From>  <To>Reserved</To>
        <TransitionString>
          <Event> Reserving</Event>
          <Condition> Avail(Ticket) == true</Condition>
          <Action> Reserve(FlightTicket) </Action>
        </TransitionString>
      </Transition>
      <Transition>
        <From>Reserved</From> <To>Booked </To>
        <TransitionString>
          <Event> Booking</Event>
          <Condition> Avail_Credit >= Ticket_Price </Condition>
          <Action> Booking(FlightTicket)</Action>
        </TransitionString>
      </Transition>
      </Transitions>
  </Service>
</BusinessProcess>
```

Fig. 2. FlightTicketBooking.xml.

Integration Agent The real challenge in our work is how to integrate existing service agents to execute a global workflow? In the AgFlow, we propose an integration agent to construct an agent community for each workflow instance. The integration agent gets the workflow specification from the interface agent and integrates service agents to execute the workflow. It contacts the discovery agent to locate appropriate service agents. It negotiates with service agents about task execution. One of the other responsibilities of the integration agent is to react to the changes during the execution of workflow. This paper focuses on the integration agent. The detailed scenario on integrating service agents is given in next two sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<Message>
   <MessageType> advertise</MessageType>
   <Repository> yellowpage</Repository>
   <AgentIdentity>
     <AgentName>FlightAgent</AgentName>
     <AgentAddress>203.23.3.27</AgentAddress>
     <AgentType>service</AgentType>
   </AgentIdentity>
   <AgentCapabilities>
     <SupportDTD>
        http://agflow.cse.unsw.edu.au/business_process.dtd
     </SupportDTD>
     <SupportService>
        http://FlightAgent.cse.unsw.edu.au/FlightBooking.xml
     </SupportService>
   </AgentCapabilities>
</Message>
```

Fig. 3. Advertisement.xml

3 Workflow Execution Planning

In this section, we discuss how the integration agent coordinates with other agents to select a workflow execution plan. Assume that the business processes have been wrapped by service agents and the service agents have registered with the discovery agent. We start from the interface agent submitting a workflow specification to the integration agent.

3.1 Locating service agents

When receiving the workflow specification from the interface agent, the integration agent parses the workflow specification into a set of tasks W[1]:

$$W = \{t_1, t_2, ..., t_n\} \tag{1}$$

For each task t_i, the integration agent sends a query to the discovery agent to search all the service agents that can execute the task t_i. The search result is a set of service agents A_i. After finishing the queries for each task t_i in W , the integration agent gets a set of 2-tuples C:

$$C = \{< t_1, A_1 >, < t_2, A_2 >, ..., < t_n, A_n >\} \tag{2}$$

$$A = \bigcup_{i=1}^{n} A_i = \{a_1, a_2, ..., a_m\} \tag{3}$$

In (3), the set A represents all the of service agents that can execute certain task in the workflow W. For each a_i in A, searching the tuples in C and putting all the tasks that the service agent a_i can execute into a set, we can have C':

$$C' = \{< a_1, T_1 >, < a_2, T_2 >, ..., < a_m, T_m >\} \tag{4}$$

Where T_i denotes a set of tasks, the 2-tuple $< a_i, T_i >$ denotes that the service agent a_i can execute all the tasks in T_i. After grouping service agents by tasks that they can execute, we can have C^*:

$$C^* = \{< A'_1, T_1 >, < A'_2, T_2 >, ..., < A'_l, T_l >\} \tag{5}$$

$$\mathbb{T} = \{T_1, T_2, ..., T_l\} \tag{6}$$

In (5), the 2-tuple $< A'_i, T_i >$ denotes that $\forall a \in A'_i$, the service agent a can execute all the tasks in T_i. In (6), $\forall T_i \in \mathbb{T}$, there is at least one service agent that can execute T_i.

3.2 Negotiation between the integration agent and service agents

For each 2-tuple $< A'_i, T_i >$ in C^*, the integration agent starts a negotiation session N_i to negotiate with every service agent in A'_i about executing the tasks T_i. Currently, we adopt the negotiation protocol which is shown in Figure 4. Note that other protocols such as auctions can also be used. We intend to include them in the future. Below we briefly introduce the negotiation protocol of Figure 4.

Call-For-Bid. The negotiation session starts with the integration agent sending a call-for-bid to each service agent in A'_i. The call-for-bid includes specification of the tasks T_i and deadline for responding to the call-for-bid.

[1] Here, we abstract a workflow as a set of tasks. The control flow will be considered during execution of the workflow.

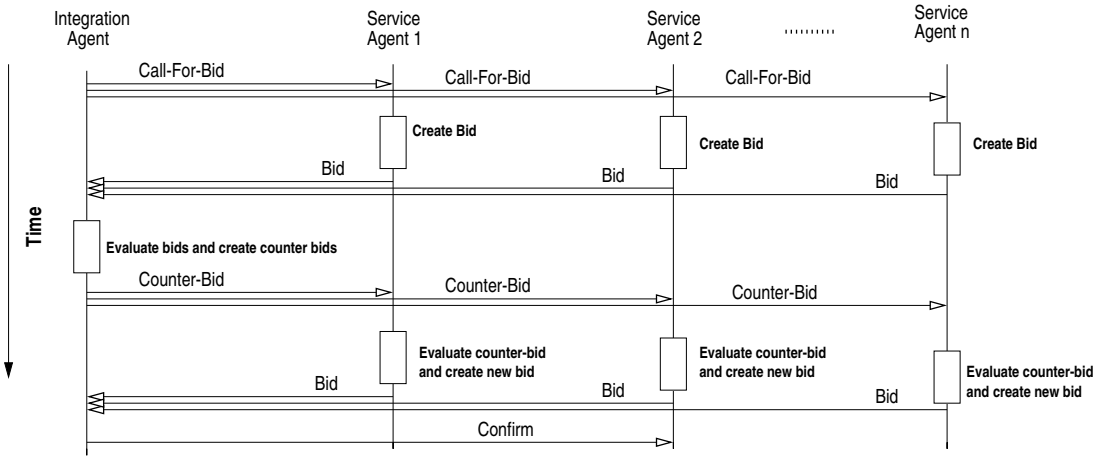

Fig. 4. Negotiation Protocol

Bid. Once a service agent inspects the call-for-bid from the integration agent, it will decide whether or not to respond with a bid, depending on the task specification, its resources and workload. If a service agent chooses to respond to the call-for-bid, it will send a bid in XML. Figure 5 shows an example of bid. If the bid is for one task(i.e.,$|T_i| = 1$), the service agent will indicate the cost it charges for executing the task and the duration of task execution. If the bid is for a set of tasks(i.e.,$|T_i| > 1$), then the cost and duration will be specified for each task in T_i. Here, we assume that the bid can not be partially accepted. The integration agent either accepts the whole bid or rejects it.

Counter-bid. After the integration agent receives bids, it will send a counter-bid to service agents. In a counter-bid, the integration agent bargains the value of the execution cost and the work duration of a task. The integration agent uses following bargaining strategies:

- **User-dependent Tactics.** Process composers can define ECA(Event-Condition-Action) rules [Han92] to specify the counter-bids. For example, the following ECA rule can be used to create a counter-bid with 200 dollars and 3 hours when there are more than four service agents which send bids for the task of hotel room booking.

  $\{Receive\_Bid(b)\}[Num(bider, Hotel\_Room\_Booking) > 4]/Send\_Counter\_Bid(200 \text{ dollars}, 3 \text{ hours})$

- **Experience-dependent Tactics.** For this type of tactics, the predominant factors used to decide which parameters to bargain is based on the past negotiation experiences. The integration agent logs all the bids in past negotiation sessions so that it can generate counter-bids from past experiences. For example, the integration agent can search the bids for the same

task in past negotiation sessions, then selects the bid that asks for both minimal value of execution cost and work duration. If such bid does not exist, the latest accepted bid can be selected as counter-bid. Also, the integration agent can generate the counter-bid based on currently available bids. If there is a bid that asks for both minimal value of execution cost and work duration, the counter-bid will be created by applying a discount function(e.g., eighty percent) on both value of execution cost and work duration. If such a bid does not exist, then the counter-bid's bargaining value of execution cost(respectively, work duration)is the minimal value of execution cost(respectively, work duration) in all currently available bids. This type of tactics is also specified by a set of ECA rules.

In absence of the user-dependent tactics, the integration agent will use the experience-dependent tactics to create counter-bids. When a service agent receives a counter-bid, it will decide whether to generate a new bid based on the counter-bid. Service agents can continue sending bids until deadline has passed. The integration agent will confirm the bid when it requests the service agent to execute the tasks.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE Bid SYSTEM "bid.dtd">
<Bid>
  <TasksBid>
    <TaskBid>
      <Task>Flight_Ticket_Booking</Task>
      <Cost>500 dollars</Cost> <WorkDuration> 8hours</WorkDuration>
    <Output>
      <FlightTicket Departure="Sydney" Destination="HongKong"
                    DepartureDate="13:00 21/05/2001" FlightClass="Business">
    </Output>
  </TaskBid>
  <TaskBid>
     <Task>Hotel_Room_Booking</Task>
     <Cost>400 dollars</Cost> <WorkDuration> 12 hours</WorkDuration>
     <Output>
       <HotelRoom Class="5 star" CheckinDate="21/05/2001" Duration="5 day">
     </Output>
   </TaskBid>
 </TasksBid>
 <AcceptDeadline> 22/07/2001 </AcceptDeadline>
</Bid>
```

Fig. 5. bid.xml

3.3 Generating Execution Plans

Once bids are available in negotiation sessions, the integration agent can select the bids, integrate the service agents to execute the workflow. Before giving details about selecting service agents, we define the following concepts:

Definition 1: Execution Schema. $s = \{T_1^*, T_2^*, ..., T_m^*\}$ is an **execution schema** of the workflow $W = \{t_1, t_2, ..., t_n\}$ if:

1. $T_i^* \subseteq W$,
2. $T_i^* \bigcap T_j^* = \emptyset$, when $i \neq j$,
3. $\bigcup_{i=1}^{m} T_i^* = W$, and
4. $T_i^* \in \mathbb{T}$.

In this definition, all the tasks T_i^* in s are mutually disjoint subsets. The combination of all the tasks T_i^* is the workflow W, and each T_i^* can be executed by different service agents, respectively. So, all the tasks will be executed only once. In fact, the execution schema indicates how to decompose workflow into a set of tasks. For example, $W = \{t_1, t_2, t_3, t_4, t_5\}$, $s = \{T_1^*, T_2^*\}$ is an execution plan of W, if $T_1^* = \{t_1, t_3\}$, $T_2^* = \{t_2, t_4, t_5\}$ and there are at least two service agents that can execute T_1^* and T_2^* respectively.

Definition 2: Execution Plan. $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_m^*, b_m, a_m >\}$ is an **execution plan** of the execution schema s if:
For each 3-tuple $< T_i^*, b_i, a_i >$ in p, the service agent a_i sends the bid b_i for the tasks T_i^*.
In fact, an execution plan indicates which service agents can be selected to execute the workflow.

The integration agent use the Set Packing algorithm[2][Ski97] to generate a set of execution schemas S from W, C^* and $\mathbb{T}$(see the definitions in Section 3.1):

$$S = \{s_1, s_2, ..., s_n\} \tag{7}$$

Bases on the available bids, for each s_i in S, the integration agent will generate a set of execution plans P_i:

$$P_i = \{p_1, p_2, ..., p_k\} \tag{8}$$

$$P = \bigcup_{i=1}^{n} P_i \tag{9}$$

Where P is the set of all the execution plans that can be used to integrate service agents.

[2] The Set Packing algorithm is a graph algorithm which is used in applications such as network planning, scheduling, etc. The description of this algorithm is outside the scope of this paper.

3.4 Selecting Execution Plans

In this section, we first discuss the criteria which will be considered when selecting execution plans. Then the details about the selection process is given.

Evaluating Criteria. The following criteria will be used to select execution plans:

(i)Cost and Time. The total cost and the total execution time are two basic criteria that we use to select a desired execution plan. For an execution plan p_i, the total execution cost $c(p_i)$ and total execution time $t(p_i)$ are calculated as follows:-

$$c(p_i) = \sum_{k=1}^{m} c\_bid_k \tag{10}$$

$$t(p_i) = CPT(t\_bid_1, t\_bid_2, ..., t\_bid_m) \tag{11}$$

In (10), total execution cost is the sum of the execution cost(i.e., c_bid_i) in every bid from the execution plan p_i. In (11), t_bid_k denotes the work duration in every bid from the execution plan p_i. The function CPT uses the Critical Path algorithm[3][Smi89] to calculate the execution plan's total execution time. Ideally, the workflow is expected to be executed in minimal time with minimal cost. Unfortunately, it is not always possible to have such an execution plan. We will explain later how to handle this situation.

(ii) Size of agent community. For different execution plans, the corresponding agent communities could have different size Z_p(i.e., the number of agents in community), since some service agents can execute one task, while other service agents can execute a set of tasks. Bigger size agent community will have more communication overhead.

(iii) Reliability of service agent. Service agents' reliability is an important issue to be considered when selecting execution plans. Basically, the reliability has twofold meanings: whether the service agent can finish tasks and whether it can finish tasks on time as specified in its bid. Here, R represents the service agent's reliability:

$$R = \begin{cases} \frac{\sum_{i=1}^{n}\left(\frac{t\_bid_i}{t\_actual_i}\right)}{n} & n \neq 0, \\ 1, & n = 0 \end{cases} \tag{12}$$

In (12), assume that the service agent has executed some tasks for n times. t_bid_i is the task execution duration as specified by the service agent in the bids for

[3] The Critical Path algorithm is a graph algorithm which is used in project scheduling application. The description of this algorithm is outside the scope of this paper.

a given task, t_actual_i is the actual execution duration that service agent spent on the execution of a given task each time. In case service agent can not finish the task, t_actual_i is ∞. So, R is calculated based on bids in past negotiation sessions and task execution results in past workflow instances. These data are logged by the integration agent. It should be noted that the larger R is, the more reliable the service agent is. R will be used to estimate the service agent's actual execution duration $t\_estimate$ based on its current bid:

$$t\_estimate = \begin{cases} \frac{t\_bid}{R}, & R \neq 0, \\ \infty, & R = 0 \end{cases} \tag{13}$$

For example, service agent a has been assigned to execute the task t_i for five times. The corresponding bids and the execution results are listed in Table 1. In

Table 1. Service agent a's bids and execution results

	Execution Time in Bid t_bid	Actual Execution Time t_actual
1	20 hours	21 hours
2	15 hours	17 hours
3	17 hours	16 hours
4	15 hours	15 hours
5	15 hours	14 hours

this, $R \approx 0.9937 (R = \frac{\frac{20}{21}+\frac{15}{17}+\frac{17}{16}+\frac{15}{15}+\frac{15}{14}}{5})$. If now service agent a indicates that it can finish the task in 20 hours in its bid, we can estimate that the service agent's actual execution duration is $\frac{20}{0.9937} \approx 20.13$ hours.

Based the reliability of service agents, total execution time of a plan p can be estimated as $t_e(p)$, where:

$$t_e(p) = CPT\ (t\_estimate_1, t\_estimate_2, ..., t\_estimate_m) \tag{14}$$

Selection Process. Execution plans are selected by the integration agent in two phases. In the first phase(pruning phase), execution plans are selected using a set of ECA rules that are defined by process composers. These ECA rules use criteria which are defined above. For example, in (15), the ECA rule excludes the execution plans which have more than seven service agents, while in (16), the ECA rule excludes the execution plans with execution cost over 700 dollars.

$$\{Evaluating\_ExecutionPlan(p)\}[Community\_size(p) > 7]/Exclude(p) \tag{15}$$

$$\{Evaluating\_ExecutionPlan(p)\}[c(p) > 700 \text{ dollars}]/Exclude(p) \tag{16}$$

Assume that the set of execution plans P' is the result of the pruning phase. In the second phase(weighting phase), an execution plans will be selected from

P'. The follows formula will be used:

$$V(p) = W_c \times \left(\frac{c(p)}{C_{planned}}\right) + W_t \times \left(\frac{t_e(p)}{T_{planned}}\right) + W_s \times \left(\frac{Z_p}{Z_w}\right) \quad (17)$$

Where $C_{planned}$ (respectively, $T_{planned}$) represents the process composers expected total cost (respectively, work duration). Z_p is the size of the agent community, while Z_w is the number of the task in the workflow. W_c,W_t and W_s $\in [0, 1]$, are the weights of cost, time and the size of agent community. Process composers are responsible for providing values of $C_{planned}$, $T_{planned}$, W_c, W_t and W_s.

In (17), process composers can balance cost, time and size of agent community to select a desired execution plan by adjusting the value of W_c, W_t and W_s. Specifically, if the cost is the most important factor, the weights can be set as: $W_c = 1$, $W_t = 0$, and $W_s = 0$. The integration agent will choose the execution plan which has the minimal value of V(i.e., $min(V)$). If there are more than one execution plans which have same minimal value of V, then a plan will be selected randomly. We say the selected one is an optimized execution plan.

4 Dynamic Business-to-Business Integration

In this section, we discuss how the integration agent coordinates with service agents to implement dynamic B2Bi.

4.1 Execution Plan Evaluation

The integration agent starts the workflow instance based on the selected execution plan. It updates the execution plan based on the changes (e.g. a service agent fails when it is executing a task) which may happen during the execution of the workflow instance. The integration agent adopts a late binding strategy, which means it confirms the service agent's bid to execute task at the execution time. In this way, the integration agent can update the execution plan without withdrawing any accepted bids. Bellow, p_c is the current execution plan. p' contains the service agents which are executing tasks, T_u is the set of tasks which are waiting to be assigned to service agents.

$$p_c = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_x^*, b_x, a_x >\} \quad (18)$$

$$p' = \{< T_j^*, b_j, a_j >, ..., < T_n^*, b_n, a_n >\} \quad (19)$$

$$T_u = \{t_1, t_2, ..., t_i\} \quad (20)$$

When a change happens, the integration agent considers the following cases:

Case 1: The service agent a fails after it has been executing the task t for y unit of time. In this case, the current execution plan becomes **unexecutable** and the task t need to be re-executed. So, the integration agent needs to find another executable plan to continue the execution of workflow. The integration agent reacts to the change by performing the following steps:-

- **Step 1.** Excludes all the execution plans which contain the service agent a's bids from P'(the result of pruning phase). The result is a set of execution plans P^*.
- **Step 2.** Selects execution plan p from P^*, such that:
  1. $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
  2. $p' \subset p$, and
  3. there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u \bigcup \{t\}$.

  P^{**} is the set of all the selected execution plan p.
- **Step 3.** In any $p \in P^{**}$, for all the finished tasks, replaces the bids with task execution results. For the task t, add y to the execution time. P^{**} is used to select a new execution plan as discussed in section 3.4.

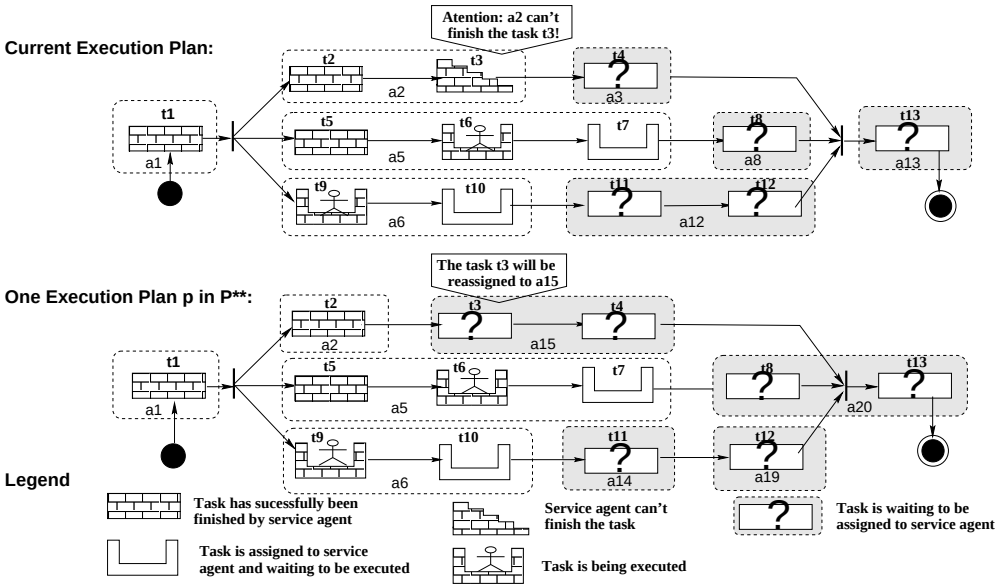

Fig. 6. Example 1: The service agent a_2 fails after it had been executing the task t_3 for 5 hours

Example 1. Figure 6. shows an example of case 1. In this example, a_2 fails so that it can not finish t_3, a_5 and a_6 are executing their tasks. t_8, t_{11}, t_{12} and t_{13} are still waiting to be assigned to service agents. First, by excluding any execution plans which contain a_2 from P', we have P^*. In P^*, selects the execution plans in which a_5 and a_6 will execute same tasks as they do in the current execution plan and one service agent executes only t_1. The p in Figure 6 is an example of such execution plan. Note that in p, t_3 will be re-assigned to a_{15}. And for the finished tasks t_1, t_2 and t_5, the bids will be replaced by the task execution results. So, the integration agent can use p to continue the execution of the workflow.

Case 2: The service agent a finish the execution of the tasks T^*, but the execution duration is different from what the integration agent estimated(i.e., t_estimate). In this case, the current execution plan is still executable. But this change may make another execution plan become a better alternative. So the integration agent reacts to the change by performing the following steps:-

- **Step 1.** Selects execution plan p from P', such that:
  1. $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
  2. $p' \subset p$, and
  3. there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u$.

  P^{**} is the set of all the selected execution plan p.
- **Step 2.** In any $p \in P^{**}$, for the finished tasks, replaces the bids with execution results. Then P^{**} is used to select a plan p_t as discussed in section 3.4. If value of $V(p_t)$ is less than value of $V(p_c)$, integration agent will change the current execution plan to p_t. Otherwise, integration agent keeps the p_c to execute the workflow.

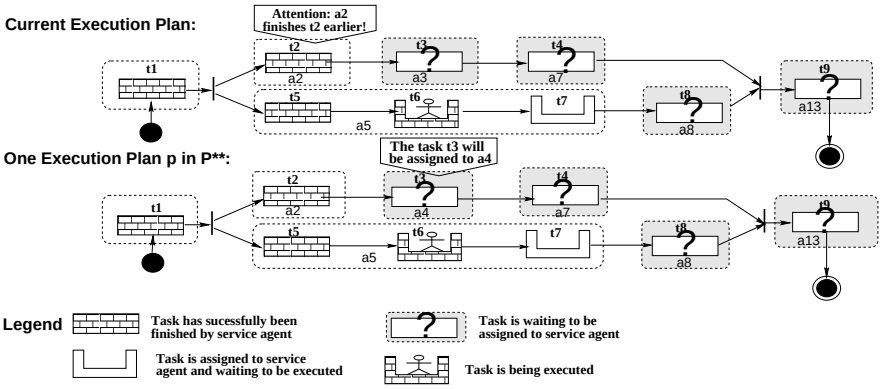

Fig. 7. Example 2: The service agent a_2 finish execution the task t_2 earlier than the integration agent estimated

Example 2. Figure 7 shows an example of case 2. In this example, when a_5 is executing t_7, a_2 has finished t_3 earlier than that the integration agent estimated. So, t_4 can have more time to be executed. The integration agent can select a_4 to execute t_4 if a_4 asks for less execution cost comparing to a_3.

Case 3: The service agent a sends a new bid b for the tasks T^*, and T^* is the subset of T_u. The bid b can be used to generate a better alternative execution plan than the current execution plan. So the integration agent reacts to the change by performing the following steps:-

- **Step 1.** Selects execution plan p in P', if:
  1. $p = \{< T_1^*, b_1, a_1 >, < T_2^*, b_2, a_2 >, ..., < T_k^*, b_k, a_k >\}$,
  2. there exists some T_i^* in p's 3-tuples, and $\bigcup T_i^* = T_u$,
  3. $p' \subset p$, and
  4. there exists some T_j^* in p's 3-tuples, and $\bigcup T_j^* = T^*$

  P^* is the set of all the selected execution plans p.
- **Step 2.** For each execution p in P^*, replace all the 3-tuples $< T_j^*, b_j, a_j >$ with the 3-tuple $< T^*, b, a >$, generates a new execution plans set P^+
- **Step 3.** In any $p \in P^+$, for the finished tasks, replaces the bids with task execution results. Then P^+ is used to select a plan p_t as discussed in section 3.4. If the value $V(p_t)$ is less than value of $V(p_c)$, integration agent will change the current execution plan to p_t.

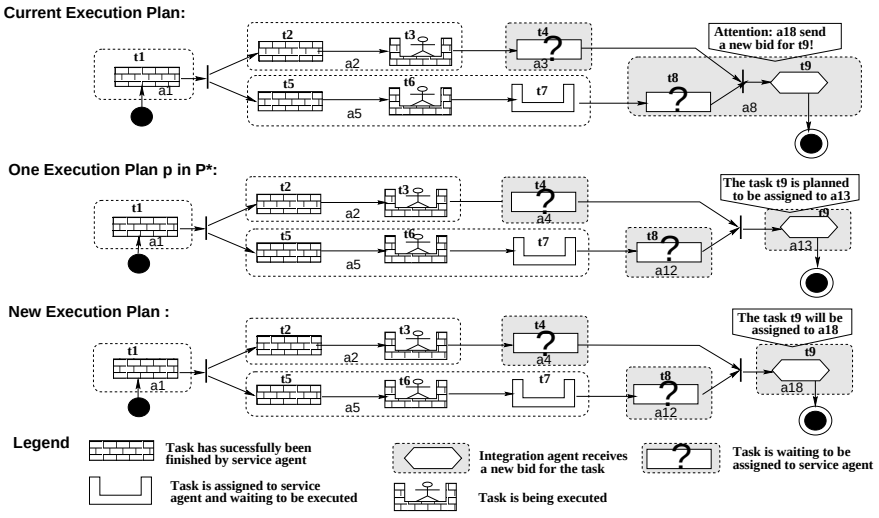

Fig. 8. Case 3: The service agent a_{18} sends a new bid b for the task t_9.

Example 3. Figure 8 shows an example of case 3. In this example, when a_2 and a_5 are currently executing tasks , a_{18} sends a new bid for t_9. The integration agent searches the execution plan p where a_2 and a_5 execute same tasks as they do in current execution plan, one service agent execute only t_1 and the other one service agent executes only t_9 . p is such an execution plan. New execution plan can be created by replacing a_{13} with a_{18}.

4.2 Workflow Execution

Based on selected execution plan, the integration agent integrates the service agents to execute the workflow by constructing an agent community. During the execution of the workflow, the integration agent acts as a facilitator for

service agents to exchange their data. Meanwhile, the integration agent acts as a monitor. It periodically pings the service agents when they are executing tasks, in order to detect whether any of them have failed or not. If the service agent has exhausted the estimated execution time and still can not finish the tasks, the integration agent will ping it more frequently, because such service agent is more prone to fail. Service agents also send messages to the integration agent when they have finished certain action during the execution of the task. The integration agent forwards such messages to the interface agent so that users can be aware of the progress of workflow execution. The integration agent also logs the execution results of each task, which will be useful for selecting execution plans for later workflow instance.

The agent community will dismiss after the workflow has been finished. A new agent community will be formed for next workflow instance.

5 Implementation

The AgFlow architecture is shown in Figure 1. The prototype is deployed using EJB(*Enterprise JavaBeans*). Agents communicate through a communication bus which is implemented using JSDT(Java Share Data Toolkit). The interface agent provides a GUI (Graphical User Interface) to access the AgFlow system. It is implemented using Java. Service agents are implemented using Java. Currently, three types of services are considered in the AgFlow: Domino-based workflows, Enterprise JavaBean applications, and Java applications that allow access to relational databases via JDBC. We are currently working on wrapping other types of services including Oracle Workflow and SAP/R3.

In a service agent, Actual Service Wrapper provides an interface to access the actual business process, Service Description is an XML document describing the service provided by the business process, Service Advertiser registers the service into discovery agent, Service Negotiator negotiates with integration agent about service execution and Service Execution Controller controls business process execution.

In the discovery agent, Service Registry, Service Maintainer and Query Engine are implemented as entity beans. We adopt the Oracle8i database server as meta-data repository. Each XML document is stored in object-relational tables. The Query Engine takes a service query message from the integration agent as input and translates it into an Oracle SQL query. We use the Oracle XML SQL Utility for Java to pass an SQL query to the underlying Oracle8i database. The results are sent back to the utility and then embedded in an XML document. The XML document includes a list of service agent's descriptions.

In the integration agent, Service Locator is implemented as entity bean, while the Workflow Execution Planner, Workflow Execution Manager and Service Negotiator is implemented as session bean. For every workflow instance, the integration agent creates a Workflow Execution Planner and a Workflow Execution Manager, and it creates a set of Service Negotiator to conduct the multiple negotiation sessions with service agents.

6 A Scenario

We consider a business trip planning workflow to illustrate the use of the AgFlow. The workflow schema includes three tasks which are flight ticket booking(t_1), hotel room booking(t_2) and car booking(t_3). The workflow has the following constraints:

- The salesman can book a flight ticket and a hotel room at the same time.
- If both bookings are successful, there is the third task: car booking.

We developed 14 service agents ($a_1, a_2, ..., a_{14}$) that wrap three types of services (Domino-based workflows, Enterprise JavaBean and JDBC applications). All the service agents are registered with the discovery agent. We will show how to define a workflow schema, create a workflow instance, and execute it using the AgFlow.

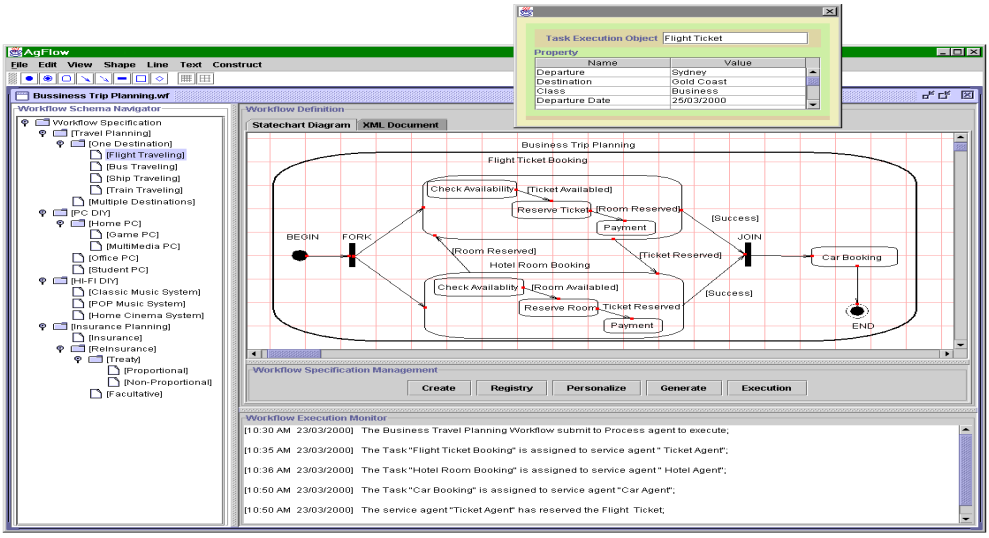

Fig. 9. Interface Agent

1. **Define the Workflow Schema.** The interface agent offers a workflow definition panel (shown in upper right panel of Figure 9) which allows process composers to draw a UML statechart diagram that defines a workflow schema. Workflow schemas are organized using domain-specific hierarchy. In left panel, every non-leaf node in the hierarchy represents a set of workflow schemas of a specific domain. Every leaf node represents a schema of particular workflow. For example, in Workflow Schema Navigator panel of Figure 9, there are four workflow domains namely, `Travel planning`, `PC DIY`, `HI-FI DIY` and `Insurance planning`. `Home PC` is sub-domain of `PC`

`DIY` and has two workflow schemas (i.e., `Game PC and Multimedia PC`). The schema of the business trip planning workflow is defined as shown in the draw panel of Figure 9. This schema is stored as a leaf node called `Business Trip` of the domain `One Destination`.

2. **Create Workflow Instance.** An end user can browse workflow schemas using the Workflow Schema Navigator panel. When she/he clicks on a leaf node, the corresponding UML statechart is displayed in Workflow Definition panel. An instance of the workflow is created by clicking on the `Create` button in the Workflow Management panel. The user can then supply values of the parameters that are needed to execute the workflow (e.g., the destination and departure date). After that, the user can click on the `Generate` button to generate the XML document that describes the workflow. After clicking on the `Execution` button, the XML document is sent to the integration agent to execute the workflow.
3. **Dynamic Business-to-Business Integration.** The integration agent parses the XML document into three tasks. For each task, it queries the discovery agent to locate the relevant service agents. C^* is created by the integration agent based on query results. $C^* = \{< A'_1, T_1 >, < A'_2, T_2 >, ..., < A'_5, T_5 >\}$, $A'_1 = \{a_1, a_2, a_3\}$, $T_1 = \{t_1\}$, $A'_2 = \{a_4, a_5, a_6\}$, $T_2 = \{t_2\}$, $A'_3 = \{a_7, a_8, \}$, $T_3 = \{t_3\}$, $A'_4 = \{a_9, a_{10}, a_{11}\}$, $T_4 = \{t_1, t_2\}$, $A'_5 = \{a_{12}, a_{13}, a_{14}\}$, $T_5 = \{t_1, t_3\}$. Since there are five 2-tuples in C^*, the integration agent creates five negotiation sessions to negotiate with service agents. The set of execution schema is: $S = \{s_1, s_2, s_3\}$ where $s_1 = \{T_1, T_2, T_3\}$, $s_2 = \{T_3, T_4\}$, $s_3 = \{T_2, T_5\}$. Based on the available bids in negotiation sessions, the integration agent will create a set of execution plans. An execution plan will be selected as discussed in 3.4. Assume that the selected execution plan is $p = \{< T_2, b_2, a_5 >, < T_5, b_5, a_{12} >\}$. Based on the execution plan p, the integration agent will integrate service agent a_5 and a_{12} to execute the business trip planning workflow.

7 Related Work

The approaches related to business processes integration exist in several fields including *shared business objects*, *cross-enterprise workflow* and *software agents*.

An early solution for B2Bi is shared business objects. Shared business objects typically relay on distributed object frameworks such as CORBA or DCOM [OH97] and other state-of-the art technologies such as message queue, publish/subscribe brokers. Different enterprises represent their high level services as business objects. In conjunction with middleware technologies, business objects are involved thereby providing the services. This solution caters for interoperability across heterogeneous environments, portability on different hardware and software platforms. It is suitable for integration of a small number of tightly coupled services.

Currently, the cross-enterprise workflow is beginning to gather momentum in B2B integration. Related projects include the CMI[GSCB99] project at MCC,

CrossFlow[Cro00] at IBM and eFlow[CIJ+00]form HP lab. In CMI, they proposed a Service Oriented Process model and a set of service management primitives such as `activity place holder`, `dynamic role assignment`, `repeated option creator`, `awareness events`, `process synchrony` etc. The goal is to be able to provide a framework for flexible, plug and play approach to cross-organization workflow composition. However, CMI has not addressed the issue of brokering and selection of services that goes beyond what is stated in the service interfaces.

CrossFlow aims at providing high-level support for workflow in dynamically formed virtual organizations. Virtual organizations are formed when some business processes of a company are outsourced to external service providers. An organization that wants a service to be performed on its behalf(the service consumer) outsources this service to an organization that can perform this service(service provider). High level support is obtained by contract model which includes the process submode. Virtual organizations are dynamically formed by contract-based match-making between service providers and consumers. eFlow provides a mechanism for defining composite services from basic services. Composite services can be preassembled or created on the fly, and can dynamically adapt to changes in the business environment.

As discussed in section 1, agent technology is one of the promising candidates for E-commerce environment. Several types of agents have been used in E-commerce applications, including source, discovery and mediator agents. The use of software agents in automating a single organization business processes have been discussed in several publications e.g. in [CS96,JFJ+96,PM99]. In [CS96], each workflow is represented by multiple personal agents, actor agents and authorization agents. These agents act as personal assistants performing actions on behalf of the workflow participants and facilitating interaction with other participants or organization specific WFMS. In [JFJ+96], the multi-agent architecture consists of a number of autonomous agencies. A single agency consists of a set of subsidiary agencies which is controlled by one responsible agent. Each agent is able to perform one or more services. These atomic agents can be combined to form complex services by adding ordering constraints and conditional control. However, neither [CS96] nor [JFJ+96] adopts agent technology to compose workflow execution engine dynamically. The logic for workflow processing is hard-coded and thus it is hard to reuse this workflow execution engine for other business processes. In [PM99], a multi-plan state machine agent model is presented. Agents are assembled dynamically from the descriptions in a blueprint language and can be modified while running. But users have to describe in detail how agents can be assembled together and what changes are allowed in priori.

Although there is little work on integration being done, agents can be extended to include integration capabilities. By doing so, the integration solution can take agent's advantage of negotiation ability and capability to adapt to dynamic changes in environments. Early effort in this direction such as MESChain[CD01] focus on using software agent to implement the supply chain integration.

We have presented in this paper the design and implementation of the AgFlow, an agent-based workflow system for dynamic business-to-business integration. In our approach, the agent community for a specific workflow instance is optimally and automatically composed based on the context of workflow execution. It can self-adapt and react to changes during execution.

References

[CD01] Ibrahim Cingil and Asuman Dogac. An architecture for supply chain integration and automation on the internet. *Journal of Distributed and Parallel Databases (to appear).*, 2001.

[CIJ+00] Fabio Casati, Ski Ilnicki, Li-Jie Jin, Vasudev Krishnamoorthy, and Ming-Chien Shan. eFlow: a Platform for Developing and Managing Composite e-Services. Technical Report HPL-2000-36, HP Laboratoris Palo Alto, 2000.

[Cro00] CrossFlow project web page, 2000. http://www.crossflow.org.

[CS96] J. Chang and C. Scott. Agent-based workflow: TRP support Environment (TSE). *Computer Networks and ISDN Systems*, 28(7-11):1501–1511, 1996.

[Geo99] D. Georgakopoulos, editor. *Information Technology for Virtual Enterprises*, Proc. of the 9th Int. Workshop on Research Issues on Data Engineering. IEEE Computer Society, March 1999.

[GSCB99] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information System, Special Issue on Information System Support for Electronic Commerce*, 24(6):429–456, 1999.

[Han92] E. Hanson. Rule condition testing and action execution in ariel. In *Proceedings of the ACM SIGMOD*, 1992.

[HS97] M. Huhns and M. Singh, editors. *Internet-Based Agents*, IEEE Internet Computing. IEEE, July 1997. Special Issue on Agents.

[JFJ+96] N. R. Jennings, P. Faratin, M. J. Johnson, T. J. Norman, P. O'Brien, and M. E. Wiegand. Agent-based business process management. *International Journal of Cooperative Information Systems*, 5(2&3):105–130, 1996.

[OH97] R. Orfali and D. Harkey. *Client/Server Programming with JAVA and CORBA*. John Wiley & Sons, Inc., 1997.

[PM99] Krzysztof Palacz and Dan C. Marinescu. An agent-based workflow management system. Technical report, Computer Sciences Department, Purdue University, 1999.

[Ski97] Steven S. Skiena. *The Algorithm Design Manual*. Springer Verlag, 1997.

[Smi89] Jeffrey D. Smith. *Design and Analysis of Algorithms*. PWS-KENT Publishing Company, 1989.

[ZNBO01] Liangzhao Zeng, Anne Hee Hiong Ngu, Boualem Benatallah, and Milton O'Dell. An agent-based approach for supporting cross-enterprise workflows. In *Proceeding of the 12th Australasian Database Conference*, 2001.