IEICE TRANS. INF. & SYST., VOL.E84–D, NO.10 OCTOBER 2001

PAPER *IEICE/IEEE Joint Special Issue on Autonomous Decentralized Systems and Systems' Assurance*

A Commitment-Based Approach for Business Process Interoperation

Jie XING[†], ***Student Member***, **Feng WAN**[†], **Sudhir Kumar RUSTOGI**[†], ***and*** **Munindar P. SINGH**[†], ***Nonmembers***

SUMMARY Successful e-commerce presupposes techniques by which autonomous trading entities can interoperate. Although progress has been made on data exchange and payment protocols, interoperation in the face of autonomy is still inadequately understood. Current techniques, designed for closed environments, support only the simplest interactions. We develop a multiagent approach for interoperation of business process in e-commerce. This approach consists of (1) a behavioral model to specify autonomous, heterogeneous agents representing different trading entities (businesses, consumers, brokers), (2) a metamodel that provides a language (based on XML) for specifying a variety of service agreements and accommodating exceptions and revisions, and (3) an execution architecture that supports persistent and dynamic (re)execution.

key words: *commitments, statecharts, business process, interoperation, e-commerce*

1. Introduction

The study of autonomous, decentralized systems gains importance with the expansion of business-to-business (B2B) electronic commerce. We would like autonomous businesses to be able to operate together logically without being subject to central control. This means we should be able to specify and obtain the correct *interoperation* of their processes.

Current software technology, based on distributed computing and databases and dealing exclusively with low-level abstractions, is inadequate for the proper treatment of interoperation. This is because the associated abstractions—remote methods and distributed transactions—address only the simpler challenges in integrating heterogeneous systems. Remote methods offer no conceptual abstractions for composing activities, whereas transactions require a tighter coupling of activities than is appropriate. Even the extended transaction models require the specification of compensatory methods or transactions, which are usually impossible to guarantee. The following major challenges are not properly addressed by existing approaches:

- *Autonomy.* In e-commerce, the interacting parties are autonomous and must retain their autonomy, limited only by their contracts.
- *Heterogeneity.* In e-commerce, the interacting software may be of any internal design and construction, and may be upgraded at different times.
- *Exceptions.* Because e-commerce presupposes autonomy and heterogeneity, interoperation frequently runs into exceptions, which (because of autonomy and decentralization of the participants) are notoriously difficult to handle.
- *Revisions.* Because e-commerce interactions can be long-lived, potentially lasting forever, they must release their results prematurely or send in corrections as a consequence of handling exceptions. This means that results ought to be able to be revised.

Workflow tools emphasize ways of structuring activities by specifying sequencing or concurrency requirements on tasks. Specifications of data resources and human actors are less important. Most importantly, however, workflow tools lack the abstractions for addressing the above challenges.

Our Approach, Conceptually. We apply agents to process interoperation in a way that naturally addresses the above challenges. We rely on two crucial properties of agents. First, the agents must interact at a high level by forming and managing *commitments* to one another. These commitments are about the information they exchange, about changes to that information, and about each other's needs. For example, an agent would not only send results to others, but may commit to notifying its consumers if it modifies those results or to satisfying its consumers with respect to a predicate. Second, the agents must be *persistent.* This is essential so the agents may form, manage, and act according to their commitments. For example, an agent may retry a task until it obtains results acceptable to it and to its peers. The agent would receive updates and send updated results to its consumers. If the agents didn't outlast their tasks, such actions would be impossible.

The agents reason about the formation and manipulation (including revocation) of social commitments. The manipulation of commitments is constrained through specified *metacommitments.* Our metamodel includes a small set of carefully engineered *metacommitment patterns* through which a given interaction can be structured. The metacommitment pat-

Manuscript received April 6, 2001.
Manuscript revised May 29, 2001.

[†]The authors are with the Department of Computer Science, North Carolina State University, Raleigh, NC 27695-7535, USA.

terns are given a formal operational semantics based on statecharts [6] and translated into rules that can be executed by different agents.

Contributions. We develop a technically well-founded metamodel for interoperation that accommodates the subtleties of e-commerce interactions. Whereas current approaches to interoperation can accommodate only the simplest client-server (request-response) kinds of interactions, our approach includes the more flexible interactions that are needed to accommodate exceptions and revisions. Our metamodel and its semantics can form the basis of for practical approaches and standards in e-commerce. Although we don't emphasize the formal aspects of our work here, the theory of commitments is well-developed [16] and has been applied in a temporal logic approach for verifying e-commerce protocol compliance [20].

Organization. Section 2 describes some important upcoming approaches to interoperation in B2B e-commerce. Section 3 describes our metamodel and presents its key concepts. Section 4 introduces the operational semantics for the main components of the metamodel, and shows how they can be mapped to a rule-based execution framework. Section 5 discusses the literature and future directions.

2. Challenge for Existing Approaches

We review some of the relevant existing or emerging approaches and standards. We consider representative languages and protocols.

2.1 Languages and Protocols

The extensible markup language (XML) has become the language of choice for encoding business information and services on the Internet [23]. The Common Business Library (CBL) promotes business interoperability by representing common business concepts (e.g., companies, services, and products) and interactions (e.g., catalog searches, purchase-order processing, and inventory updates) using a public collection of XML components [3].

RosettaNet [14] offers industry-wide electronic business interoperability standards. It develops common business process interfaces between supply-chain trading partners through common vocabularies. Coupled with an exchange protocol specifying services, transactions, and messages, these can support business dialogs in the partner interface process (PIP).

The Open Trading Protocol (OTP) [13], defines a number of different types of OTP transactions such as purchase, refund, value exchange, authentication, withdrawal, and deposit. These transactions involve trading roles, exchanges, and components. Typical trading exchanges are offer, payment, delivery, and authentication.

The Information and Content Exchange (ICE) protocol [8] defines the roles and responsibilities of syndicators and subscribers, defines the format and method of content exchange, and provides support for management and control of syndication relationships. In ICE, the syndicator produces the content that is consumed by subscribers. The ICE protocol is essentially a request/reply protocol in which the subscriber and syndicator may assume several different roles: subscriber versus syndicator, requester versus responder, and sender versus receiver.

2.2 Workflows

Open buying on the Internet (OBI) is a proposed standard framework that describes the main roles and low-level standards for use in e-commerce [12]. OBI includes buyer, seller, and payment organizations, and a requisitioner (member of the buyer). Tian et al. show how OBI-compliant workflows can be achieved [19]. Although OBI facilitates low-level interoperation, it is not organizationally flexible and does not help in handling exceptions or revisions. For example, we will presumably need a different standard to accommodate marketplaces instead of direct buying or to handle delivery delays. We suspect these limitations are the reason why OBI is marketed for low-dollar, non-mission critical procurement. However, the main challenge is in specifying the right interoperation among companies and is independent of the cost of the goods being traded.

The Workflow Management Coalition (WfMC) is developing standards to support interoperability for e-commerce [7]. These standards provide the key infrastructure through which the process information of different enterprises could be integrated. However, the state of the art covers only the simplest e-commerce scenarios. Their interoperation component is quite weak [1]. It supports the main interactions are *chaining* (where a workflow hands off control to another) and *nesting* (where a workflow makes a call to another and expects control to be returned). These abstractions better correspond to traditional distributed objects and are not suitable for autonomous interaction as necessary to handle the challenges outlined above.

2.3 Evaluation and Motivation

To evaluate the above approaches and motivate our own, we consider a simple example.

Example 1: A customer comes up with a need to travel to a certain city (with multiple hotels and airports) on a certain date. She contacts her travel agent who in turn requests an airline and a hotel clerk to make appropriate reservations. The clerks are to send confirmations to the traveler. The customer may have some additional requirements that are not initially brought out. For example, the hotel may not be close to the airport chosen by the airline clerk and, say, for a late flight, the hotel location is an important constraint of the customer. If the customer's requirements are not met, she complains to her travel agent. If the travel agent offers a high quality of service and works to satisfy his customer, he would then make a revised request to the clerks, let's say by trying for an earlier flight. □

Although small, Example 1 is not trivial. It combines the B2B and B2C aspects of e-commerce. Four autonomous parties are involved, and communication among them does not follow a simple nesting of requests and responses. Even a task that executes successfully may need to be revisited as shown in Fig. 1. Current situations work well for the simpler situations up to the first travel plan. However, because they offer no conceptual or operational support for exceptions, programmers are left to fend for themselves. Typically, they are forced to employing ad hoc techniques that lead to reduced productivity and ineffective solutions.

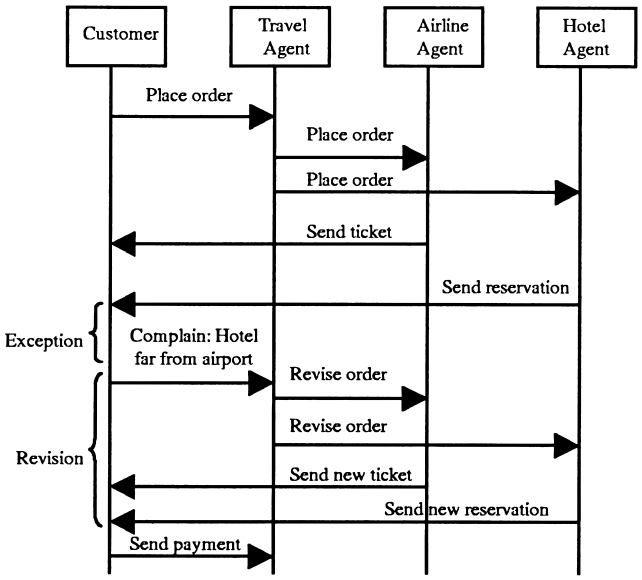


Fig. 1 An e-commerce scenario that does not follow a simple nesting of requests and responses.

Existing approaches can help as follows. CBL would facilitate the different parties agreeing about their business transactions, although some extensions might be needed to specify the various restrictions on their contracts. ICE could be used to specify how hotels and airlines respond to requests for bookings from travel agents. Possibly, our travel agent may subscribe to a listing of special promotions from an airline or hotel. OTP could handle the payment modes used by the participants. RosettaNet could be used to specify the common vocabulary and interfaces for the travel domain.

Thus, existing approaches can specify the details of the business transactions, domains, and payment, as well as simple protocols for composing the different activities. However, they either refer to the infrastructure or solely to low-level synchronization interactions. They provide little support for handling the exceptions and revisions of information that are crucial even in this simple scenario. Each such case that is improperly handled leads to tedious work by a human, not only costing time and money to the interacting organizations, but resulting in an unhappy customer. By contrast, our approach develops principled abstractions to handle just such scenarios.

3. Metamodel for Interoperation

A *metamodel* is a language for specifying models. Our metamodel is a language for specifying the desired interoperation. Traditional metamodels are based on flowcharts or activity diagrams, which are of limited help in handling complex situations. Coding the associated procedures or scripts is tedious work. While some of the complexity of the scripts arises from accessing legacy data, much of it is because of the ad hoc manner in which exceptions are handled.

We develop a simple metamodel that supports interoperating businesses represented by autonomous agents. Like traditional metamodels, our metamodel captures the structuring of activities. However, unlike traditional metamodels, our metamodel includes

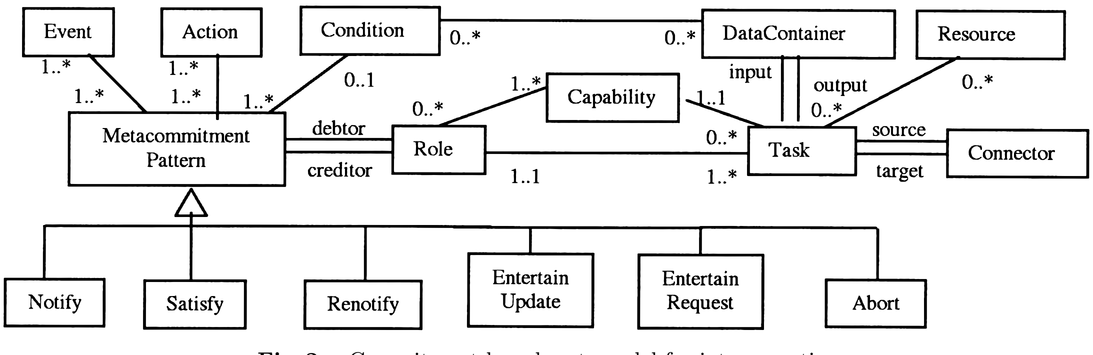


Fig. 2 Commitment-based metamodel for interoperation.

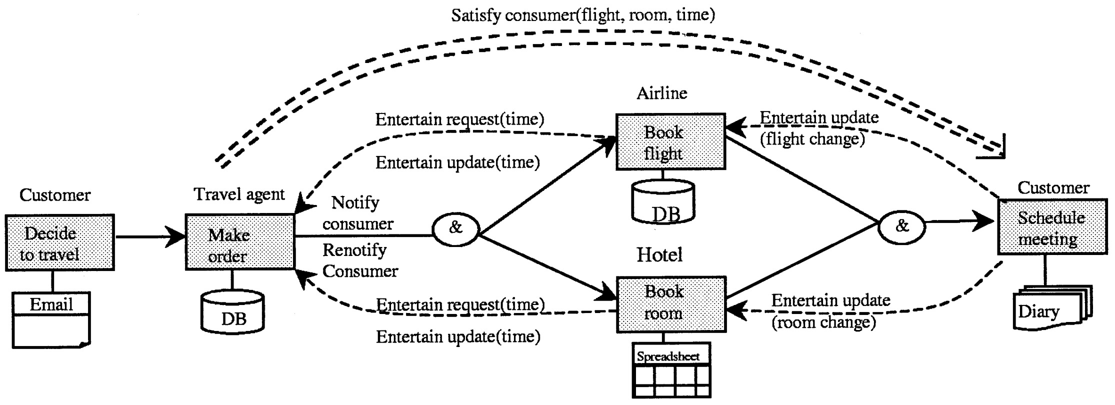


Fig. 3 Model of interoperating to plan a trip.

concepts dealing with persistence and commitments to support exceptions and revisions. Figure 2 presents our metamodel in UML [4]. We now discuss its main concepts.

- A *task* identifies a definite piece of work. Tasks may be atomic or compound. We model a task as taking inputs and producing results, both of which map to *resources* (such as databases). A task is executed only if it is triggered; upon completion it produces an event, which (through the connectors) may trigger other tasks.
- *Capabilities* implement tasks by defining the primary processing required of a task. They are handles to procedural code that would do the bulk of the computation in an enterprise.
- *Connectors* capture the control flow among the tasks. They also capture the data flow among tasks by binding the outputs and inputs of successive tasks. On the surface, these connectors are traditional and include Fork, Branch, Or-join, and And-join. However, their operational semantics is nontraditional and supports reentrance, which is essential for managing commitments properly, especially to enable revisions.
- A *role* provides the capabilities required of any tasks assigned to it. During execution, a concrete agent with matching capabilities is bound to each role. Importantly, roles are used to specify the applicable commitments. Thus, roles capture the underlying organizational structure. Agents can volunteer to assume certain roles that would require them to perform certain tasks by executing their capabilities. Thus, an agent playing a role must implement all the capabilities that the role provides and must honor the metacommitments of that role.
- A *commitment* relates a debtor role, a creditor role, and a condition, in the scope of a context group [16]. This means that the debtor is obliged to the creditor to satisfying the given condition. The context group is the virtual organization that provides the scope of the interoperation. The condition of the commitment involves the relevant predicates from the domain. In our example, the airline can commit to the given traveler that she is confirmed on a particular flight.

  Domain-level commitments, such as the above, arise at run time. They are not represented in the models and don't show up in the metamodel.

  Importantly, commitments are flexible, and can be revoked or modified. For example, even a confirmed flight booking or an entire flight can be canceled. However, the revocation or modification of commitments is constrained through *metacommitments*, which are commitments whose condition itself involves commitments. In our example, the airline can commit to the traveler that if the flight that she is on is canceled, the airline will book her on an alternative flight. In other words, if the airline has to cancel the commitment (of the booking), then the airline will create an alternative commitment. That is, the above is a metacommitment.

  Metacommitments are important because they capture the structure or qualitative aspects of service agreements. Service agreements may have additional domain-specific details.

  Our metamodel codifies the important kinds of metacommitments as *metacommitment patterns.* We have found a small set of patterns that cover real-life examples as well as other research scenarios. For expository ease and to save space, we restrict this paper to the six main patterns.

  These patterns handle revisions and capture the major classes of semantic exceptions [15] as arisen in example 1. These exceptions deal with the ongoing interoperation, not dealing with low-level programming errors such as divide-by-zero.

  In most cases, the metacommitment patterns apply in conjunction with the connectors, which carry the information about which commitments are made. However, in some cases, patterns can apply across two or more connectors.

We use the following notation. Rectangles represent tasks to each of which a role is assigned. Each task has associated resources. Solid arrows represent the connectors. Associated with each connector is a metacommitment from its source role to its target role. Single-dashed lines represent metacommitments from the target of a connector to a role. Double-dashed lines represent metacommitments that go across more than one connector.

Example 2: Figure 3 illustrates our metamodel applied on Example 1. The control and data flows are as described in Example 1. The applicable metacommitment patterns are that the roles will notify and renotify other roles, entertain requests and updates, satisfy their customers and abort from their producer. □

4. Metacommitment Semantics

Commitments help the agents behave in a coherent manner to realize the robustness and flexibility that we desire. Specifying and managing commitments through metacommitments is at the heart of our approach. Since constraining the specifications is the major purpose of metamodeling, we define a small but expressively rich set of metacommitment patterns. By adding further structure, these patterns help us specify interactions that are coherent at the level of the application domain. These patterns are based on a study of real-life interoperation as well as of examples from the literature. Our patterns have two properties crucial for inclusion in a metamodel. They are

- *minimal*, in that each pattern imposes the fewest reasonable restrictions to maximize the agents' autonomy.
- *composable*, so the patterns may be assigned in any combination by a modeler.

4.1 Basic Behavioral Model

To support minimal and composable metacommitment patterns, we require that the agents follow a basic behavioral model. Agents who follow this behavioral model may invoke any capabilities (implemented in any manner), but the agents persist and include well-defined states in which they can reexecute a capability, and enter into commitments. The proprietary details of the capability or the agent's design are not revealed.

Figure 4 shows the basic behavioral model expressed as a statechart. Statecharts are well-established in software engineering as a means to specify concurrent computations [5]. Typically, such descriptions involve complex sequences of events, actions, conditions and information flow that combine to form a system's overall

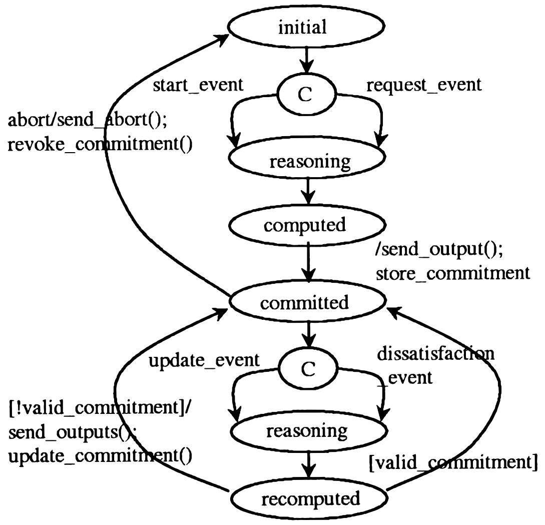


Fig. 4 Basic agent behavioral model.

behavior. A statechart is primarily composed of *states* (OR-states, AND-states, and basic states) and *transitions*. Transitions are labeled by an expression of the form $e[c]/a$. Intuitively, event e triggers the transition if condition c is true when e occurs. As a result, action a is performed. Each of e, c, and a is optional. The states in our statecharts are abstract and correspond to sets of physical states of the underlying computation.

On receiving a request or a control signal, an agent following basic behavioral model begins reasoning. Upon completion, it sends the results to some selected consumers and commits to those results. Further events may cause the agent to reexecute its reasoning. If the results change substantially to invalidate the agent's commitments, it announces the new results (canceling the old commitments and creating the new ones). The behavioral model just limits the agents' actions. However, the agent will have specific metacommitments that force it to carry out certain actions.

4.2 Metacommitment Patterns

We now consider some interesting metacommitment patterns. The ones discussed here are the simplest and most common six of the ten odd patterns that we have studied.

P1. *Notify the consumer.* This pattern comes into effect when a role finishes its execution for the first time. COMPUTING_DONE is an event that signifies the "computed" state in the statechart of Fig. 4. The rule is fired when COMPUTING_DONE is received. Results are sent to the given consumer and the associated commitment is created and stored. (Shown in Fig. 5, (p_1).)

P2. *Renotify the consumer.* This pattern is enacted when an agent finishes its execution for a second

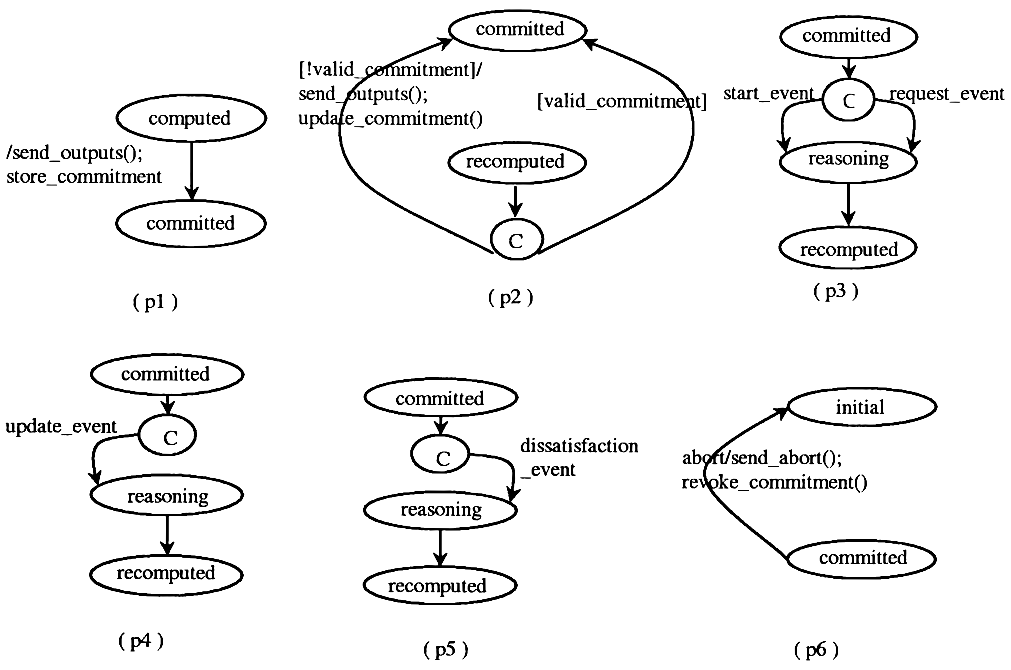


Fig. 5 Metacommitment patterns: (p1) notify consumer, (p2) renotify consumer, (p3) entertain request, (p4) entertain update, (p5) satisfy consumer, (p6) abort from producer.

or later time. If a previous commitment to a consumer doesn't hold, then it must send the update information to the consumer and store the new commitment. (Shown in Fig. 5, (p_2).)

P3. *Entertain request from another role.* This pattern consists (from Fig. 4) of the *start*, *reasoning*, and *computed* states, and the REQUEST_EVENT transition and the succeeding transition. (Shown in Fig. 5, (p_3).)

P4. *Entertain update from producer.* This pattern consists (from Fig. 4) of the *committed*, *reasoning*, and *recomputed* states, and the UPDATE_EVENT transition and the succeeding transition. (Shown in Fig. 5, (p_4).)

P5. *Satisfy the consumer.* This pattern is similar to entertain-update pattern with UPDATE_EVENT replaced by DISSATISFACTION_EVENT. The practical ramifications of this pattern are much greater, however, because it enables an agent to send a complaint upstream and to demand that a property it desires in its inputs be satisfied. (Shown in Fig. 5, (p_5).)

P6. *Abort from producer.* This pattern comes into effect when a role aborts its execution. This pattern consists of *committed*, *initial* states. When an abort event occurs, an abort message is sent to the given consumer and associated commitment is canceled. (Shown in Fig. 5, (p_6).)

Pattern composition is an important concept in object-oriented design and analysis. Here we use the concepts to compose agent behaviors. Each pattern shows one agent interaction property, which is considered as minimal granularity for representing agent behavior. Thus each pattern represents one agent property. We compose patterns: p_1, p_2, p_3, p_4, p_5, and p_6 to obtain basic agent behavior model. The basic agent behavior model satisfies each property of all of these patterns. You can find the details for pattern composition in [22].

Let's consider a special agent, which periodically supplies latest information to its consumer. The information is automatically delivered to its consumer agent without explicit request message. The method seems particularly suitable for the conveyance of sensor data. If the sensor data changes the new value is automatically forwarded to the consumer agent that processes these data. We call the agent as a monitor agent. We choose patterns: p_1, p_2, p_3, p_4 and compose them to obtain a monitor agent behavior model (see Fig. 6). Although it satisfies these properties of p_1, p_2, p_3, p_4, it doesn't satisfy properties of p_5 and p_6. Thus the monitor agent is simpler than the basic agent in behavior.

Example 3: Figure 3 illustrates our metamodel applied on Example 1. In the following, we apply these metacommitment patterns to the example.

- After the travel agent receives a request from the customer, this forces it to perform the task *Make order* by *entertain-request* pattern.
- After the travel agent finishes its task, it notifies

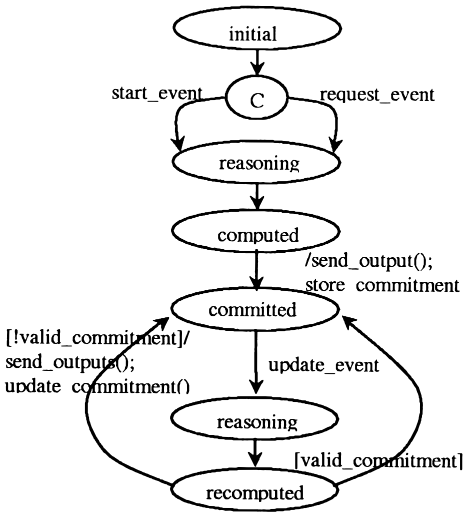


Fig. 6 Monitor agent behavioral model.

the hotel and airline agents about the customer's order information respectively, such as the date, place, and price by using *notify* pattern.
- When the hotel and airline agents receive the request from the travel agent, this forces them to perform their tasks *book flight* and *book room* by using *entertain-request* pattern.
- After the airline and hotel agents finish their tasks, they notify the flight, hotel and price information to the customer by using *notify* pattern.
- If the customer is dissatisfied with the itinerary schedule and suggests a different date or destination city, he renotifies the travel agent about the revisions by using *satisfy the consumer* pattern.
- When the travel agent receives the revised requests, it entertains this updated on account of *entertain update* commitment pattern. If the effects of the revisions are significant and lead to the violation of a prior commitment by the travel agent, the travel agent may propagate the effect of revisions to hotel and/or airline agent(s).
- When the hotel, or airline agent receives the revised information, it entertains the updated and reexecutes *Book flight* or *Book room*.
- After the hotel, or airline agent finishes its tasks, it can send the updated information to the customer by using *renotify* pattern if any of the prior commitments don't hold any more.
- Sometimes the customer's requirements cannot be satisfied by the airline agent after the airline agent tries several times. The airline agent sends the abort information to the travel agent by *abort* pattern. The travel agent tries another airline agent by *notify* pattern, or send an abort information to the customer by *abort* pattern. It depends on the customer's decision about whether it wants to abort the order, or renotify the travel agent of a new requirement. □

4.3 Connectors

For brevity, we consider a single generic connector with m inputs and n outputs. The connector fires if all inputs are received and the condition on some out-branch evaluates to `true`. This translates to a simple forward-chaining rule. When the connector requires fewer than the total number of inputs (e.g., if it is an or-join), then the rule for each of the receiving agents is a little more complex to count the number of available inputs, so it can fire when the desired number of them has arrived.

After the connector condition evaluates to true and target tasks have been executed, there might be updates or corrections sent from source tasks that can require reexecution by the target tasks. We term this behavior *reentrance.*

There are two approaches for handling reentrance. One, the connector condition is always checked to ensure that the number of executions of each task reentering the connector is identical. (This test is essential for the first execution.) Two, the connector condition is not checked for executions (after the first execution). Events generated by the source tasks are sent directly to the target, which may reexecute. The second approach is more practical, but also more challenging from a theoretical standpoint, e.g., to ensure correctness.

5. Discussion

Because of the autonomy and decentralization of the participants, specifying and managing e-commerce interactions can be challenging. Conventional techniques fall into one of two extremes, being either too rigid (over-restricting the designer) or too unstructured (not helping the designer). Our commitment-based approach takes the middle path, emphasizing the coherence desired from the activities of autonomous decentralized entities, but allowing the entities to change their mind in a controlled manner, which enables them to achieve progress in a dynamic, unpredictable world.

Sycara et al. consider the problem of matchmaking in heterogeneous agents [18]. Their approach essentially involves matching agents based on their capabilities. This complements our approach because finding agents with the right capabilities is an important subproblem for us. A possible synthesis is an enhanced approach that will also consider the metacommitments that the agents are willing to support.

There are several products and a huge body of literature on workflows. The work on rule-based systems for workflows is pertinent. A lot of it, however, deals with the lower-level aspects of workflow management. Several approaches include rules for handling exceptions [2], [17], but they do not capture the bigger patterns of long-lived interactions, as studied here.

The CREW approach provides one of the best recent approaches from the database community [10]. Kamath and Ramamritham develop a rich model for workflow that includes failure handling and enables *opportunistic* rollback, restricting the extent of the compensation activities. They consider the interactions among the different tasks. CREW can accommodate interactions among workflows, including the important case where a workflow feeds information to another, and can change its results under some circumstances. Like CREW, we consider higher-level abstractions, but unlike CREW, we also consider abstractions that relate to the organizational aspects of the workflow. Some of our patterns are designed to accommodate opportunism in rollback, e.g., by renotifying consumers of updates only when necessary.

The *language for action* metamodel involves commitments as well [21], and is applied in the *Action-Workflow* tool. This metamodel uses *loops* representing a four-step exchange between a *customer* and a *performer*: (a) a request from the customer, (b) negotiation by the two about the task, (c) actual performance of the task, and (d) evaluation of the performance by the customer. A step may potentially be nested with other loops. The loops metamodel has some limitations. It only considers two actors at a time, and does not explicitly consider the surrounding organizational structure. It cannot easily accommodate modifications or revocations of the commitments.

Several agent-based approaches exist. Klein and Dellarocas exploit a knowledge base of generic exception detection, diagnosis, and resolution expertise [11]. Specialized agents are dedicated to exception handling. This approach is complementary to ours; special roles could be included in our approach with commitments by other roles. The advanced decision environment for decision tasks (ADEPT) project considered workflow management [9]. This project emphasized negotiation among agents. However, the underlying notion of commitments doesn't allow contextual nesting, as in our approach.

Our approach synthesizes conventional software engineering techniques (namely, statecharts and process modeling) with artificial intelligence techniques (namely, agents and commitments) to develop a powerful approach for interoperation among autonomous, decentralized entities. Several interesting technical problems are opened up by our research. One, we are investigating an alternative operational semantics for our approach that goes beyond the conventional statechart semantics in terms of allowing reentrance and revision. Two, we are examining how to declaratively characterize the different metacommitment patterns and show how different behavioral models for agents can support a given set of metacommitment patterns and may show how, in effect, metacommitments can be compiled out in certain settings.

References

[1] M. Anderson and R. Allen, "Workflow interoperability: Enabling e-commerce," Whitepaper, Workflow Management Coalition, 1999. www.aiim.org/wfmc.

[2] F. Casati, P. Grefen, B. Pernici, G. Pozzi, and G. Sánchez, "WIDE workflow model and architecture," TR 96.050, Dipartimento di Elettronica e Informazione, Politecnico di Milano, 1996.

[3] CBL, Common business library, 1998. www.veosystems.com /xml/cbl/cbl.html.

[4] M. Fowler, UML Distilled: Applying the Standard Object Modeling Language, Addison-Wesley, Reading, MA, 1997.

[5] D. Harel and E. Gery, "Executable object modeling with statecharts," IEEE Comput., vol.30, no.7, pp.31–42, 1997.

[6] D. Harel and A. Naamad, "The STATEMATE semantics of statecharts," ACM Trans. Software Engineering and Methodology, vol.5, no.4, pp.293–333, 1996.

[7] J.G. Hayes, E. Peyrovian, S. Sarin, M.-T. Schmidt, K.D. Swenson, and R. Weber, "Workflow interoperability standards for the Internet," IEEE Internet Computing, vol.4, no.3, pp.37–45, May 2000.

[8] ICE, Information and content exchange, 1997. www.w3.org /TR/NOTE-ice.

[9] N.R. Jennings, P. Faratin, M.J. Johnson, T.J. Norman, P. O'Brien, and M.E. Wiegand, "Agent-based business process management," International J. Cooperative Information Systems, vol.5, no.2 & 3, pp.105–130, 1996.

[10] M. Kamath and K. Ramamritham, "Failure handling and coordinated execution of concurrent workflows," Proc. International Conference on Data Engineering (ICDE), pp.334–341, 1998.

[11] M. Klein and C. Dellarocas, "Exception handling in agent systems," Proc. 3rd International Conference on Autonomous Agents, pp.62–68, Seattle, 1999.

[12] OBI, Open buying on the Internet, 1998. www.openbuy.org.

[13] OTP, Open trading protocol, 1998. www.otp.org.

[14] RosettaNet, Home page, 1998. www.rosettanet.org.

[15] S.K. Rustogi, F. Wan, J. Xing, and M.P. Singh, "Handling semantic exceptions in the large: A multiagent approach," Technical Report, Computer Science Department, North Carolina State University, 1999.

[16] M.P. Singh, "An ontology for commitments in multiagent systems: Toward a unification of normative concepts," Artificial Intelligence and Law, vol.7, pp.97–113, 1999.

[17] M.P. Singh and M.N. Huhns, "Automating workflows for service provisioning: Integrating AI and database technologies," IEEE Expert, vol.9, no.5, pp.19–23, Oct. 1994.

[18] K. Sycara, J. Lu, M. Klusch, and S. Widoff, "Matchmaking among heterogeneous agents on the Internet," Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace, 1999.

[19] Z. Tian, J.-Y. Chung, L.Y. Liu, J. Li, and V. Guttemkkala, "Business-to-business e-commerce with open buying on the Internet," TR, IBM Institute for Advanced Commerce, 1999. www.ibm.com/iac.

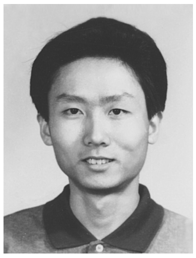
[20] M. Venkatraman and M.P. Singh, "Verifying compliance with commitment protocols: Enabling open web-based multiagent systems," Autonomous Agents and Multi-Agent Systems, vol.2, no.3, pp.217–236, Sept. 1999.

[21] T. Winograd and F. Flores, Understanding computers and cognition: A new foundation for design, Addison-Wesley, Reading, MA, 1987.

[22] J. Xing, "Commitment-based interoperation for e-commerce," Ph. D. Thesis, North Carolina State University,

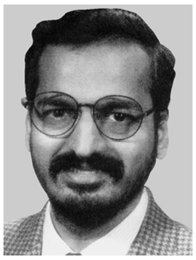
2001.
[23] XML, Extensible markup language, 1997. www.w3.org/XML/.

Jie Xing received the B.S. degree in Applied Mathematics from Shanghai Jiaotong University, China, in 1989 and the M.E. degree in Computer Aided Design from Beijing Institute of Technology, China, in 1992 and the Ph.D. degree of Operations Research Program, North Carolina State University, USA in 2001. His research interests include multiagent systems, business process interoperation and e-commerce. Currently he works at IBM Corporation.



Feng Wan received the B.S., and M.S. degrees in Computer Science from Beijing University, China, in 1994, 1997 respectively. He is presently a Ph.D. candidate of department of Computer Science, North Carolina State University, USA. His research interests include multiagent systems, workflow and e-commerce. Currently he works at Simpliciti Software Solution Inc.

Sudhir Kumar Rustogi received the Ph.D. degree in Civil Engineer from North Carolina State University, USA in 1998 and the M.S. degree in Computer Science from North Carolina State University, USA in 1999. Currently he works at Cisco Systems Inc.



Munindar P. Singh is currently an associate professor in computer science at North Carolina State University. He received the Ph.D. degree in computer science from the University of Texas at Austin in 1993. His current research interests include the theory and practice of multiagent systems with a special emphasis on communication languages, interaction protocols, and agent teams for carrying out complex, dynamic, and coordinated workflows in virtual enterprises and electronic commerce. His book, Multiagent Systems, was published by Springer-Verlag in 1994, and his coedited Readings in Agents was published by Morgan Kaufmann in 1998. He is Editor-in-Chief of IEEE Internet Computing and a member of the editorial board for Kluwer's Journal of Autonomous Agents and Multi-Agent Systems. He is the recipient of an NSF Career Award and an IBM Partnership Award.