Patterns for e-Commerce Agent Architectures: Using Agents as Delegates

Michael Weiss
Carleton University, Ottawa, Canada
weiss@scs.carleton.ca

Introduction

Agents are rapidly emerging as a new paradigm for developing software applications. They are being used in an increasing variety of applications, ranging from relatively small systems such as assistants to large, open, mission-critical systems like electronic marketplaces. One of the most promising areas of applications for agent technology is e-commerce [13, 20, 12].

In this paper we describe a group of *architectural patterns* for agent-based e-commerce systems, Agent as Delegate and its derivatives. These patterns relate to *front-end* e-commerce activities that involve interaction with the user, and delegation of tasks. They are part of a larger pattern language that will be described in a companion paper. Together, these patterns are just the beginnings of a pattern language for agent-based e-commerce system design. Consider it as the structural foundation on which more detailed pattern languages such as for the collection of user profiles, or maintaining persistence can be based.

There is no universally accepted definition of the notion of *agent*. However, the following four properties are widely accepted to characterize agents: *autonomy, social ability, reactivity* and *proactiveness* [29]. Agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactiveness). Agents typically represent different users, and there are thus several of them in a given environment. A *multi-agent system* is considered as a collection of collaborating autonomous agents, each representing an independent locus of control.

Agents also provide an appropriate *metaphor* for conceptualizing certain applications, as the behavior of agents more closely reflects that of the users whose work they are delegated to perform or support. The following characteristics of a domain are commonly quoted as *reasons* for adopting agent technology: an inherent distribution of data, control, knowledge, or resources; the system can be naturally regarded as a society of autonomous collaborating entities; and legacy components must be made to interoperate with new applications [25].

Forces

In this section, we identify forces that need to be resolved in the design of agent-based electronic commerce systems. In the presentation of our patterns we will cross-reference which of theses forces resolved by each pattern. These forces are *specific* to the e-commerce domain. We will not discuss domain-independent forces for agent system design, such as mobility and persistence. Although they still bear on the design of e-commerce systems, they are not indigenous to the design of e-commerce systems, and can be resolved according to the agent patterns documented in the literature identified elsewhere, e.g., [3] [15] [8]. We should also note that we expect this list to be continually enhanced by the community as the trade-offs involved in agent-based e-commerce system design are better understood.

AUTONOMY

The currently dominant metaphor for interacting with computers is direct manipulation. Direct manipulation requires the user to *initiate* all tasks explicitly and to monitor all events. For example, a user searches the web for an auction that offers the desired item for sale (perhaps assisted by a search engine), and subsequently monitors the state of the auction. The obvious drawback of this approach is that most of the time the user is occupied in tasks that are peripheral to its primary objectives. The user’s ability to find the best deal available at any of the many online auctions in operation is also greatly limited.

Agents can be used to implement a complementary interaction style, in which users *delegate* some of their tasks to software agents which then perform them autonomously on their behalf. This indirect manipulation style engages the user in a cooperative process in which human and software agents both initiate communication, monitor events and perform tasks. Autonomy is the capability of an agent to follow its goals without interactions or commands from the user or another agent. An autonomous agent does not require the user's approval at every step of executing its task, but is able to act on its own.

With agents performing autononmous actions, users are now facing issues of trust and control over their agents. The issue of *trust* is that by engaging an agent to perform tasks (such as selecting a vendor) the user must be able to trust the agent to do so in an informed and unbiased manner. The agent should not, for example, have entered contracts with vendors to favor them in return for a cut on their proceeds. The user would also like to specify the *degree of autonomy* of the agent. For example, the user may not want to delegate decisions to the agent that have legal or financial consequences, although a buyer agent is capable of not only finding the cheapest vendor, but also placing a purchase order itself.

NEED TO INTERACT

Agents typically only have a *partial representation* of their environment, and are thus *limited* in their ability – in terms of their expertise, access to resources, location, etc. – to interact with it Therefore, they *rely on other agents* to achieve goals that are outside their scope or reach. They also need to coordinate their activities with those of other agents to ensure that their goals can be met, avoiding that they interfere with each other.

The behavior of an individual agent is thus often not comprehensible outside its social structure – its relationships with other agents. For example, the behavior of a buyer agent in an auction cannot be fully explained outside the context of the auction itself, and of the conventions that govern it (for example, in which order – ascending or descending – bids must be made, and how many rounds of bidding there are).

An important issue in designing systems of interacting agent is dealing with *openness*. The Internet and e-commerce applications over the Internet are both examples of open systems. Open systems pose unique challenges in that their components are not known in advance; they can change unexpectedly; and they are composed of heterogeneous agents implemented by different developers, at different times, with different tools and methodologies.

One can also not assume that agents are cooperative. Some agents may be benevolent and agree on some protocol of interaction, but others will be *self-interested* and follow their own best interests. For example, in an electronic marketplace, buyer and seller agents are pursuing their own best interests (making profit), and need to be constrained by conventions.

INFORMATION OVERLOAD

People and organizations wish to find relevant information and offerings to make good deals and generate profit. However, the large set of vendors in conjunction with the many different interfaces makes it difficult for a human to overview the market. One solution has been to provide *portals* or common entry points to the web. These portals periodically collect information from a multitude of information sources and condense them to a format that users find easier to process, typically taking the form of a hierarchical index. The disadvantage of this solution is that the categories of the index will be the same for every user. Individual preferences are not taken into account when compiling the information.

MULTIPLE INTERFACES

One of the difficulties in finding information (e.g., when comparing the offerings of different vendors) is the large number of different interfaces used to present the information. Not only are store fronts organized differently, vendors don't follow the same conventions when describing their products and terms of sale. For instance, some vendors include the shipping costs in the posted price, others will advertise one price, but add a handling charge to each order. A solution is to agree on *shared vocabularies*, but these must also be widely adopted.

With the introduction of the eXtensible Markup Language (XML) for associating meta-content with data, this is slowly becoming a reality. For example, a price in a catalog can be marked up with its currency, and whether it already includes the shipping cost. However, the difficulty with any standard format is that it takes a considerable amount of time to find agreement among the interested parties. One also needs to allay the fear of vendors in losing business to competitors, once their product information becomes this easily accessible.

Ensuring Quality

Shopping online lacks the immediate mechanisms for establishing trustworthiness. How can you trust a vendor, with whom you had no previous encounter, whether the order you placed will be fulfilled satisfactorily? For exampls, any seller in an online auction could claim that the widget offered for sale is in superior condition, when the customer cannot physically verify that claim. One solution is to solicit *feedback* about the performance of a vendor (customer) from customers (vendors) after order fulfillment.

For example, the online auction site eBay keeps records of how a vendor was rated by other customers. Potential new customers will take the ratings from previous customers into account before considering buying from a vendor. However, eBay's solution falls short in two ways. Old low ratings are not discarded or discounted when more recent ratings are higher. Also, if a vendor gets a low overall rating, it is easy for her to assume a new identity and start afresh with a new rating. A mechanism for ensuring quality must avoid this.

Adaptability

Users differ in their status, level of expertise, needs, and preferences. The issue of adaptability is that of tailoring information to the features of the user, for example, by selecting the products most suitable for the user from a catalog, or adapting the presentation style during the interaction with the user. Any approach to tailoring information involves creating and maintaining a user model. When creating a user model, two cases need to be distinguished [2]: for first time visitors no information about them is available, and the user characteristics must be recognized during the interaction; on subsequent visits a detailed user model about a visitor is already available, and can be used to tailor the information.

Several design considerations for user modeling are detailed in [2]. Users need to register permanently with the system to have their data stored, otherwise user models will only be maintained during a single interaction. In the context of online shopping, a system must also deal with direct and indirect users. A customer of a web store may browse for products for himself, as well as for other people (indirect users), who have different needs and preferences. Finally, the user model must be able to reflect changes in interest over time.

One approach to collecting user information is to ask the user to provide the information explicitly, for example, by filling out a form. This allows you to create a profile of the user that is potentially very accurate, and to provide personalized service to the user from the start. However, there are at least two problems with this solution. First, by requiring the user to provide this information upfront, the threshold for the user to do so is very high. Only very advanced users will want to tune their own profiles. Second, when the user’s interests change, this will not be reflected in the profile, unless the user keeps updating her profile. Again, in practice, user’s don’t update their profiles after installation.

PRIVACY CONCERNS

Personalization requires that personal information, such as the customer’s vendor preferences, are made available to the agent providing personalized service. One way of personalizing interactions between customers and vendors is for the vendor to collect information about the user from the user’s behavior (e.g. their clickstream). The user may not be aware of the information collected, nor does she always have control over what information is gathered about her. Although effective from the vendor’s perspective, this is not a desirable situation from the perspective of the customer.

Users are typically not willing to allow just anyone to examine their preferences and usage patterns, in particular without their knowledge or consent. They want to remain in control, and decide on an interaction-by-interaction basis which information is conveyed to the vendor. A solution that addresses the force of privacy concerns must put the user in charge of which information is collected and who it is made available to. An additional compexity results from the desire of some buyers, often for obvious reasons, to remain anonymous in a transaction. On the other hand, if a buyer remains anonymous, a seller cannot provide personalized service. Thus, generally, users are willing to share personal information with vendors, if the expected gains outweigh the possible threats for privacy.

SEARCH COSTS

It can be expensive for vendors and the customers to find each other. In a static marketplace, each customer can store a contact list of vendors for each product, and then quickly locate an appropriate vendor when a particular product is needed. However, an electronic marketplace is dynamic. Vendors and customers can join and leave the marketplace, and change their offerings and requirements qualitatively and quantitatively at any point in time. It therefore becomes impossible for a market participant to maintain an up-do-date list of contact. Another problem is that of selecting an optimal (e.g. in terms of price) vendor for a given requirement. If each customer maintains its own list of contacts, they run the risk of not being aware of a better deal available from a vendor not on their list.

One possible solution to these problems is to use a mediator which can match potential trading partners in the market. With the introduction of mediators, customers and vendors no longer maintain their own lists of contacts, or need to contact a large number of alternative trading partners in order to find the optimal one. One trade-offs of this solution is, however, that individual preferences or history of interaction with a particular trading partner cannot be accounted for by a mediator. Thus it is reasonable to maintain individual lists of trading partners that one has dealt with in the past, keeping track of the quality provided, and use this personalized ranking of partners to filter the list of contacts provided by a mediator.

IDENTITY

For various reasons, customers and vendors need to be represented by unique identities. The most important reasons are authentication, repudiation, and tracking. One way of assigning a unique identity to trading partners is to use one of the many unique labels which are readily available on the Internet, e.g. an email address, or a Yahoo account name. A problem with this approach is that it is also very easy to obtain a new identity, thus making authentication, repudiation, or tracking schemes that rely on such identities impractical. Similarly, a user could obtain multiple identities and pretend to represent different parties, where instead there is only one. A solution that remedies this situation must make it advantageous for individuals to keep their identities over those users who change them often.

Patterns

Rather than attempting to describe all patterns of the pattern language for agent-based e-commerce systems within a single paper and in insufficient detail, we decided to present a longer description of the Agent as Delegate patterns, and patterns that are direct consequences of this pattern. The structure of the pattern language, indicating how the patterns relate to each other, is shown in Figure 1.

Each dependency arrow in the diagram points in the direction from a pattern to a pattern in its resulting context, that is, a pattern which designers may want to consult next. As the main purpose of these patterns is to describe the architectural trade-offs faced by a designer, note that a detailed description of techniques for user profiling or ontology design is outside the scope of this paper. Finally, another intended use of these patterns is to provide guidance and training for developers who are not already expert at agent technology.

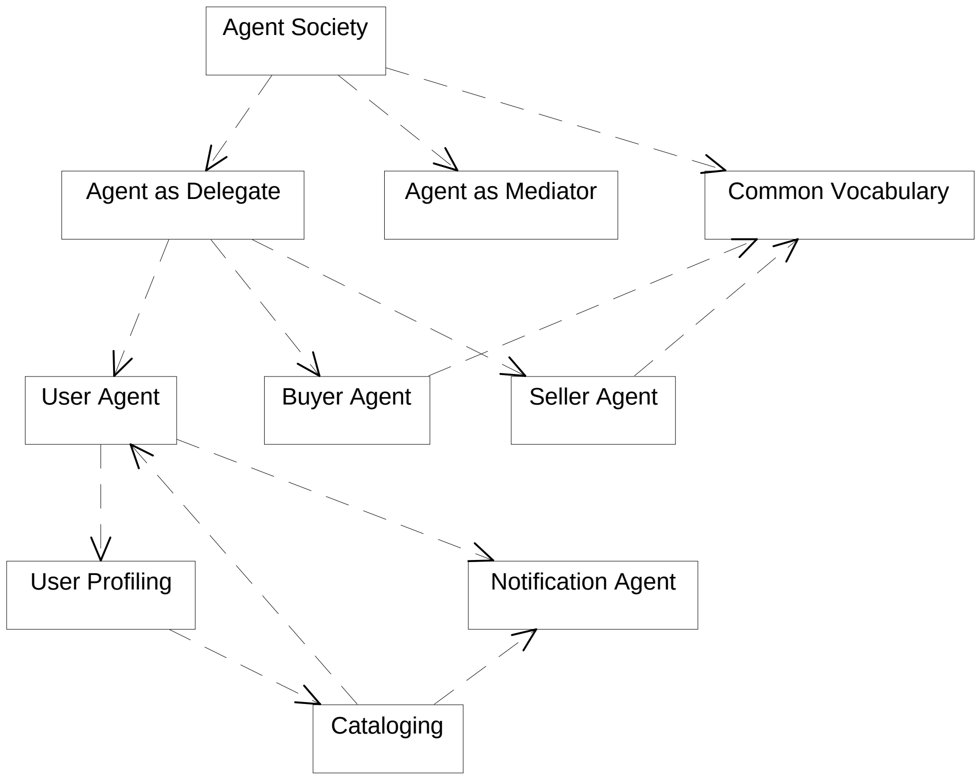

Figure 1: Map to the pattern language

AGENT SOCIETY

Context

Your application domain satisfies at least one of the following criteria: your domain data, control, knowledge, or resources are decentralized; your application can be naturally thought of as a system of autonomous cooperating entities; or you have legacy components that must be made to interoperate with new applications.

Problem

How do you model systems of autonomous cooperating entities in software?

Forces

- Autonomy
- Need to interact

Solution

Model your application as a society of agents. Agents are autonomous computational entities (autonomy), which interact with their environment (reactivity) and other agents (social ability) in order to achieve their own goals (proactiveness). Often, agents will be able to adapt to their environment, and have some degree of intelligence, although these are not considered mandatory characteristics. These computational entities act on behalf of users or groups of users [17]. Thus agents can be classified as *delegates*, representing a single user, and acting on her behalf, or *mediators*, acting on behalf of a group of users, facilitating between them.

The key differentiating characteristic between agents and objects is their *autonomy*. Autonomy is here used in an extended sense. It not only comprises the notion that agents operate in their own thread of control, but also implies that agents are long-lived (they execute unattended for long periods), they take initiative (they do not simply act in response to their environment), they react to stimuli from the environment as guided by their goals (the *receiving* agent decides whether and how to respond to a stimulus), and interact with other agents to leverage their abilities in support of their own as well as collective goals. *Active objects*, on the other hand, are autonomous only in the first of these senses.

A society of agents can be viewed from two dual perspectives: either a society of agents emerges as a result of the interaction of agents; or the society imposes constraints and policies on its component agents. Both perspectives, which we can refer to as micro and macro view of the society, respectively, mutually reinforce each other, as shown in Figure 2. For example, emergent behaviors such as specialization of agents leads to the notion of roles that agents play. Roles, in turn, impose restrictions on the possible behaviors of agents [10].

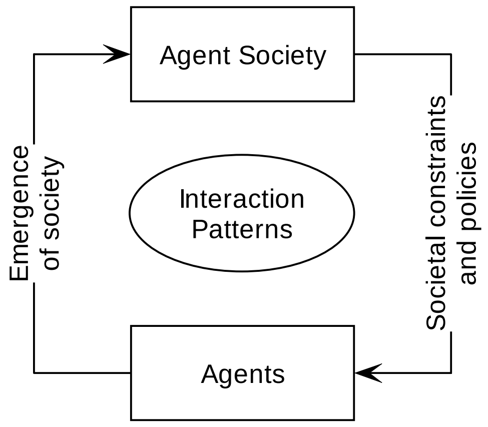

Figure 2: Micro-Macro View of an Agent Society

This suggests two approaches to systematically designing agent societies. In the first approach, we identify top-level goals for the system, and decompose them recursively, until we can assign them to individual agents. In the second approach, we construct an agent society incrementally from a catalog of interaction patterns [15]. These interaction patterns are described in terms of roles that agents can play and their interactions, and may also specify any societal constraints or policies that need to be satisfied.

Roles are *abstract* loci of control [15, 10, 22]. Protocols (or patterns of interaction) describe the way the roles interact. Policies define constraints imposed by the society on these roles [31, 27]. As an example of a policy, consider an agent-mediated auction, which specifies conventions specific to its auction type (for example, regarding the order of bids) that participating agents must comply with in order for the auction to function correctly.

In agent-based e-commerce applications, we identified four top-level roles, from which more specialized roles can be derived through subtyping: User, Task, Service, and Resource. Figure 3 depicts these roles and their subtypes used in this paper in a role diagram, using the notation introduced in [15]. *Role diagrams* are more abstract than class diagrams. Each role in the diagram defines a position and a set of responsibilities. A role has collaborators – other roles it interacts with. Arrows between roles indicate dynamic interactions between roles; their direction represents the direction in which messages are sent. The triangle indicates a subtyping relationship between roles; subtypes inherit the responsibilities of their parent roles.

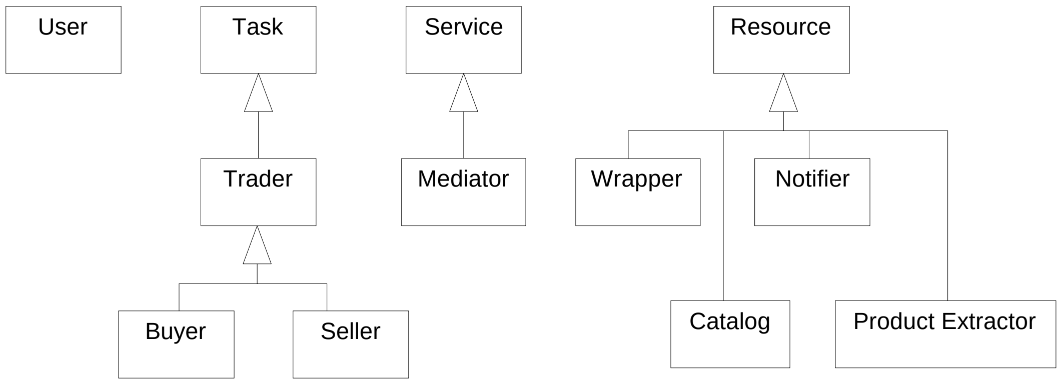

Figure 3: Top-level roles and their subtypes in this pattern language

The *User* role encapsulates the behavior of managing a user's task agents, providing a presentation interface to the user, and collecting profile information about the user, among other responsibilities. *Task* roles represent users in a specific task. This is typically a long-lived, rather than a one-shot transaction. In this context, we only consider trader agents, which can either represent the user as a buyer or a seller. The common functionality of both roles (for example, they represent the negotiation strategies of their users) is represented in the trader role. Another example of a Task role would be a Search role that encapsulates the behavior of making queries to the web on the user's behalf and filtering the results.

A *Service* role typically provides a service to a group of users. It mediates the interaction between two or more agents through this service. One example is a Directory role that provides a reference to an agent given its name (white pages agent), or references to agents that can provide a certain product or service (yellow pages agent). More advanced services facilitate more complex interactions between agents, for example, enforcing an auction protocol. For examples, see the Agent as Mediator part of the pattern language [28].

The *Resource* role abstracts information sources. These can be legacy data sources wrapped by “glue” code that converts generic requests from other agents to the API of the data source. These can be agents that process more complex queries by first breaking them down into subqueries and then collating the results. Notifier agents also belong into this category; they can be instructed to generate an alert when a certain condition in a data source is being met (for example, a product meeting given criteria is added to a product catalog).

Resulting Context

- For members of an agent society to understand each other, they need to agree on common exchange formats, as described in COMMON VOCABULARY.
- For agents that act on behalf of a single user, which is what e-commerce front-end system are concerned with, consult AGENT AS DELEGATE.
- For agents that facilitate between a group of users and their agents, refer to AGENT AS MEDIATOR in the the companion pattern language [28].

AGENT AS DELEGATE

Context

You are designing your system as a society of autonomous agents – AGENT SOCIETY. You want to delegate a user’s time-consuming, peripheral tasks to software assistants.

Problem

How do you instruct the assistant? How much discretion should you give to the assistant (e.g., authority to complete a trade)? How do assistants interact with the world?

Forces

- Information overload
- Search costs

Solution

Use agents to act *on behalf* of the user, as buyer or seller agents in an electronic marketplace for the purposes of identifying a product that meets the user's requirements; locating a vendor for the product, or customer willing to buy it, as the case may be; or negotiating a price (e.g. bidding in an auction). User agents learn about the needs of the user and build a user profile. This user profile allows vendors to customize their offerings to specific user tastes, while the user agents control what part of a profile vendors can access.

The structure of this pattern in shown in Figure 4. The role diagram shows the roles agents need to fill in the pattern. When the pattern is applied, the same agent may fill several roles. Note that we introduced the superrole of a trader agent, which contains the beliefs and behavior common to both buyer and seller agents. For example, both buyer and seller agent have a belief for the desired price, and make offers and counter-offers to other agents.

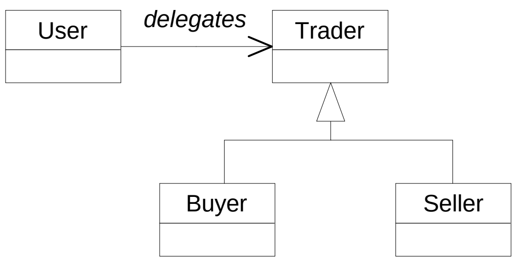

Figure 4: Role diagram for Agent as Delegate

Resulting Context

- For the interaction with the user to gather their requirements and feedback on the quality of the service received by a vendor, consult USER AGENT.
- Also consult USER AGENT for measures to control access to user profiles.
- For the design of trader agents, consult BUYER AGENT and SELLER AGENT.

COMMON VOCABULARY

Context

When agents in an AGENT SOCIETY interact, they need to agree on common exchange formats. One scenario is that you are using agents to represent customers and vendors in individual transactions – BUYER AGENT and SELLER AGENT. Buyer and seller agents need to understand each other in order to exchange messages with one other. Another scenario is that you need to derive a user profile from user interactions – USER PROFILING – and you require your information sources such as product catalogs to be well structured.

Problem

How do you enable agents (for example, buyer and seller agents) to interact? How can you make it easier for a comparison-shopping agent to extract data from a catalog?

Forces

- Multiple interfaces

Solution

In the example, for buyer and seller agents to understand each other, they need to agree on a common message format that is grounded in a common ontology. The ontology defines product-related concepts that each party must use during interaction, their attributes and valid value ranges. On receiving a message, the seller agent translates the request to a vendor-specific format and fetches the contents of the product database.

It is generally impractical to define a general-purpose ontology for agent interaction. These are unlikely to include the intricacies of all possible domains. Instead, the common ontology will be application-specific. Given such a shared ontology, the communicating agents need to map their internal representations to the shared ontology. Much progress has been made on XML-based ontologies for electronic commerce, for example xCBL, cXML, and RosettaNet, in recent years. Given that buyer and seller agents adopt one of these standards as their internal representation of domain concepts, the problem remains of building inter-ontology translators.

Fortunately, these are relatively straightforward to build using XSLT, a language for transforming XML documents into other XML documents [26, 6].

The structure of this pattern is shown in Figure 5.

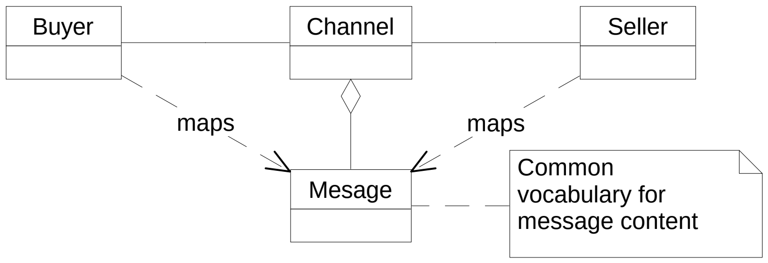

Figure 5: Role diagram for Common Vocabulary (based on [7])

Resulting Context

This pattern does not itself lead to other patterns in this pattern language. However, design patterns for ontologies have been documented elsewhere [24, 21].

USER AGENT

Context

You are delegating tasks to an agent – AGENT AS DELEGATE – that autonomously performs the task on behalf of the user. Now you want to create a locus of control through which the users can interact with the agents created on their behalf.

Problem

How do you manage the user’s profile and control access to it, and the various (concurrent) transactions in which the user participates?

Forces

- Privacy concerns
- Ensuring quality

Solution

User agents form the *interface* between the user and the system. They receive the user's queries and feedback, and present information *tailored* to users. A user agent delegates the user's queries or orders (offers to buy/sell an item) to a trader agent, and manages the various buyer and seller agents on behalf of its user. The user agent manages the user's profile and controls who can access it. The Open Profiling Standard [19] employs user agents in this way. The profile information for different vendors is stored in a personal profile repository. Sections of the profile can be restricted to a subset of the vendors only.

The structure of this pattern is shown in Figure 6.

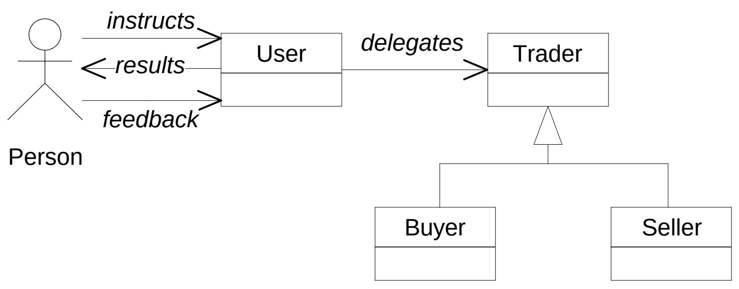

Figure 6: Role diagram for User Agent

Resulting Context

- A user can subscribe to the progress of a query or an order by instructing a NOTIFICATION AGENT.
- For more on modeling the user in such a way that required explicit user input is kept at a minimum, consult USER PROFILING.

Buyer Agent

Context

You are delegating tasks to an agent – AGENT AS DELEGATE – and provide a locus of control where users can interact with their agents – USER AGENT. You now need an agent to represent a user as a customer in a transaction with vendors in an electronic marketplace.

Problem

How do you communicate your goals and buying strategy? How do you keep in control?

Forces

- Autonomy

Solution

A buyer agent can *tailor* the selection and presentation of products to the needs of its user. It *locates* seller agents that offer the requested product or service and negotiates with them about price and other terms of sale (e.g. shipping). Kasbah and Tete-a-Tete [13] are systems that use this approach. In these systems, each buyer agent is given such information as the desired price, the maximum price to pay, and the time by which a purchase needs to be completed. In Tete-a-Tete, the agents are negotiating about other criteria than just price such as shipping terms, or extended warranties. The virtual marketplace architecture in [11] uses mobile buyer agents to represent the user in a particular transaction. A further consideration is whether a buyer agent needs final approval to complete a transaction.

The buyer role is defined as a subrole of the trader role in Figure 6 above.

Resulting Context

- For buyer and seller agents to understand each other, they need to agree on a common exchange format, as described in COMMON VOCABULARY.
- Buyer and seller agents find each other through AGENTS AS MEDIATORS [28].

Seller Agent

Context

You are delegating tasks to an agent – AGENT AS DELEGATE – and provide a locus of control where users can interact with their agents – USER AGENT. You now need an agent to represent a vendor in a transaction with customers in an electronic marketplace.

Problem

How do you communicate your selling strategy? How do you personalize product offerings?

Forces

- Autonomy

Solution

A seller agent *offers* products and services to buyer agents (directly or through a market). It answers queries for information about its owner's products or services, responds to RFPs, and enters into negotiation with buyer agents. A seller agent encapsulates its *execution state* and the vendor's selling *strategy* (e.g. function for setting the bidding price). As a design choice, a seller agent may need the vendor's approval to complete a trade. Systems using this pattern are Kasbah [13] and the virtual marketplace described in [11].

The seller role is defined as a subrole of the trader role in Figure 6 above.

Resulting Context

- For buyer and seller agents to understand each other, they need to agree on a common exchange format, as described in a COMMON VOCABULARY.
- Buyer and seller agents find each other through AGENTS AS MEDIATORS [28].
- If you wish to create a custom catalog for different buyers by tailoring the contents of a generic catalog to the buyer's profile, consult CATALOGING.

User Profiling

Context

You provide a locus of control – USER AGENT – where users can interact with their agents. You receive instructions and feedback from the user. You want to derive a profile that describes the user from these interactions. Once you have deduced a user profile you can provide it to vendors who want to personalize the customer's experience, e.g., by creating a custom catalog based on the user's preferences and buying pattern.

Problem

How to capture user profiles without requiring too much explicit input from the user. This can become involved, if you want to track users' profiles as their interest changes.

Forces

- Adaptability

Solution

During the interaction with a user, the system builds a user profile by static user modeling, classifying the user into predefined user classes (stereotypes), or dynamic user modeling (monitoring user activity and changing the profile dynamically). For example, in [2] user profiles contain a collection of interests in addition to generic information about the user. Interests contain descriptions of the topics the user is interested in. Interests may also contain weights that define the degree of interest the user has in a topic. The agent may also tailor its interaction style, providing more or less detail depending on the user's receptivity.

It is desirable to limit the amount of explicit input required from the user, for example, in the form of rules. Although manual setting of the user profile results in a very precise profile, this is difficult to perform correctly except for an advanced user [5]. On the other hand, it is advantageous to know as much as possible about the user to start with in order to keep the learning phase short. The user agent can request the user to provide demographic information such as sex, age, and profession, which allow us to group the user into a class. The interests of the user are then initialized to those typical for this class of users.

From the point of view of limiting user input, initializing the user profile by observing the user’s behavior over time is most desirable [5]. The user’s personal preferences can then be deduced using cluster analysis. This has the added advantage that it is easy to adapt the user profile to changes in their interests. Once the user profiles have been initialized, the user agent can update the profile by soliciting feedback on query results from the user, or to rate a vendor on her quality of service after the order has been fulfilled. A typical example for learning a user profile from user feedback is the Letizia user agent [16]. The Sporas system for reputation management [30] ranks vendors on their past performance.

Resulting Context

- For using user profiles to create custom catalogs for buyers by tailoring the content of a generic catalog to the buyer’s profile, consult CATALOGING.
- To construct complex user profiles (e.g. based on user navigation patterns), the information sources such as product catalogs must be well structured – COMMON VOCABULARY – and comparable on the basis of their content.
- If the information sources are heterogeneous, collaborative filtering can still help you create useful simpler user profiles; see RECOMMENDATION BROKER [28].

CATALOGING

Context

You are deriving a user profile from user interactions – USER PROFILING. You are using an agent to represent a vendor in a transaction in a marketplace – SELLER AGENT. You wish to tailor the contents of your catalog to individual buyers based on their user profiles.

Problem

How do you customize a catalog?

Forces

- Information overload

- Multiple interfaces

Solution

A product extractor dynamically creates a customized view on the catalog. It selects the products that best match the customer's preferences by comparing the product features with those contained in the profile. A user profile contains predictive information about the user's preferences toward product features as defined by the product categories. The view may also be customized regarding generic traits of the user (e.g., his level of expertise, job, or age).

Tailoring the product catalog to the user resolves the force of information overload. A large electronics parts catalog, for example, may contain 50,000 parts, of which only a small subset are relevant to the needs of an individual customer. By extracting a customized view of the catalog we show only those parts matching features specified in the user profile. An example of using this pattern is the personalized web store architecture presented in [2].

Existing product catalogs are integrated with the agent-based subsystem through catalog agents. Each catalog agent is a wrapper that "agentifies" an existing catalog, converting a catalog-specific schema to a common schema – WRAPPER AGENT [28] – used by the product extractor. Further customizations are possible. In addition to filtering the contents of the catalog against the buyer's profile, discounts can be offered to individual customers to entice existing customers to remain loyal to the business, as well as to attract new customers.

The structure of the pattern is shown in Figure 7.

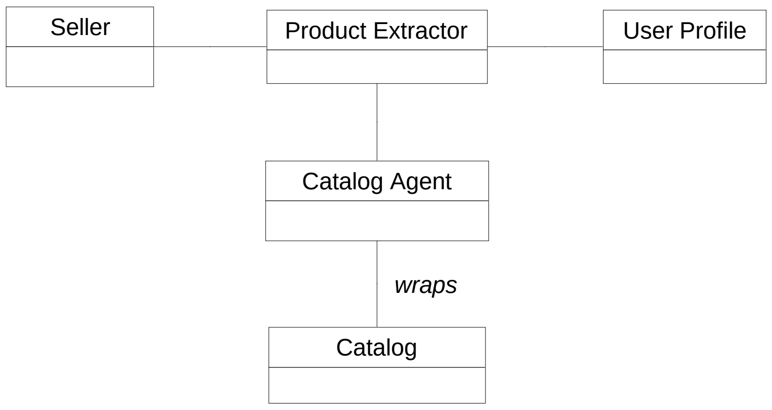

Figure 7: Role diagram for Cataloging

Resulting Context

- To control what information is disclosed to the seller, the user profile should be maintained by the user agent, and supplied to the seller agent on request. For addressing privacy concerns when accessing user profiles, consult USER AGENT.
- Consult NOTIFICATION AGENT, if you want to notify the customer about changes to the catalog (e.g., the addition of a new widget that matches the user’s profile).
- Each catalog agent is a wrapper that agentifies an existing catalog, converting a catalog-specific schema to a common schema – WRAPPER AGENT [28].

NOTIFICATION AGENT

Context

You provide a locus of control – USER AGENT – where users can interact with their agents, and want to subscribe to the progress of a transaction. You want to notify the customer about changes to the catalog – CATALOGING. In both cases, you need to monitor a data source (e.g., a product catalog) for changes, or the progress of a transaction.

Problem

How to keep users informed about relevant changes?

Forces

- Information overload

Solution

The solution involves a notification agent that resides close to the information source. It monitors the information source for changes of interest by registering with the information source, or continuously polling it. Notification agents can be considered a special type of mobile agent that only move to a single remote location. A notification agent is created with a condition, typically comprising an event and a boolean expression on the event data.

Whenever it detects an event of the specified type at the remote location, it evaluates this condition. If the condition is satisfied, the agent notifies the user agent.

The benefits of using this pattern are similar to those of using a mobile agent. It reduces the number of messages between the user agent and the information source. The example of a meta-auction engine such as [4] makes this clear: it creates an agent that monitors the progress of an auction on the user's behalf, and notifies the user about events of relevance, e.g., when the user has been outbid, and should increase her maximum bid. Without a meta-auction engine, the user agent would have to poll the state of the auction periodically.

The structure of this pattern is shown in Figure 8.

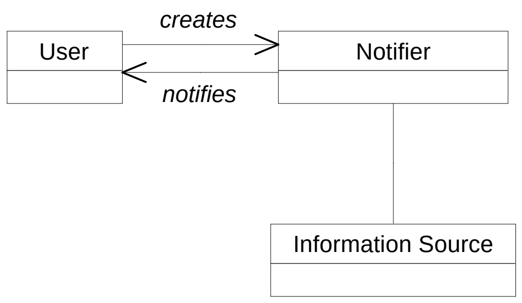

Figure 8: Role diagram for Notification Agent

Resulting Context

- You want to use the same notification agent with different vendors. Therefore, you decide to employ WRAPPER AGENTS [28] that translate from the format of vendor-specific information sources to that understood by the notification agent.

Related Patterns

There is by now a growing literature on the use of patterns to capture common design practices for agent systems, for example, [3] [14] [8]. Aridor and Lange [3] describe a set of domain-independent patterns for the design of mobile agent systems. They classify mobile agent patterns into traveling, task, and interaction patterns. Kendall et al [14] capture common building blocks for the internal architecture of agents. She introduces a layered model for agents that encompasses mobility, translation, collaboration, actions, reasoning, beliefs, and sensors. Deugo and Weiss [8] identify a set of patterns for agent coordination, which are, again, domain-independent. They classify agent patterns according to architectural, communication, traveling, and coordination patterns. They also describe an initial set of global forces that push and pull solutions surrounding the task of coordination.

Complementary to these domain-independent pattern languages, different fields have made efforts to capture pattern languages for specific domains, in particular, manufacturing and telecommunications. In the domain of agent-based e-commerce systems, we are only aware of one previous effort [11]. Our presentation adds to [11] by identifying an explicit set of domain-specific forces for e-commerce, documenting additional patterns, and establishing the positions of these patterns in a pattern language of agent-based e-commerce systems.

Examples

The following examples illustrate some uses of the patterns described in this paper. The first example, an online auctio, demonstrates how agents can be used in the negotiation phase [13] of an e-commerce transaction, in which buyers and sellers negotiate the terms of a transaction (such as price, and warranty). The second and third examples show agents in operation during the information phase [13], in which buyers navigate product catalogs, and use mediators to locate sellers that offer a certain product, or establish their actual needs.

ONLINE AUCTION

In this example, an online auction, users post items for sale, while other users place bids on those items. A user may be represented by multiple trader (buyer and seller) agents in concurrent transactions. In our example, shown in Figure 9, Bob is represented in the auction by a seller agent, while both Alice and Mary act as buyers via their buyer agents.

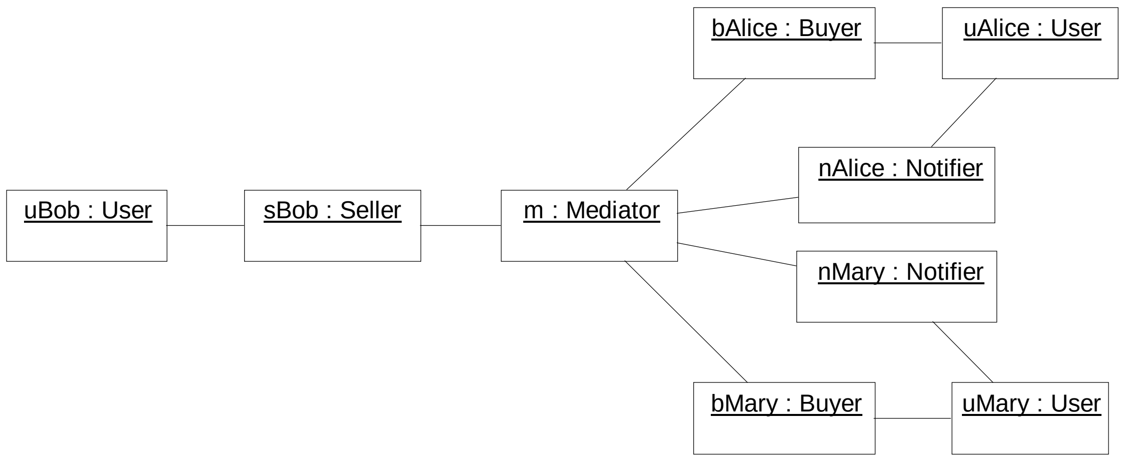

Figure 9: Collaboration diagram for an electronic auction

The auction mechanism is implemented by a mediator agent. This agent provides a meeting place for buyers and sellers. It maintains a catalog of items for sale, and a list of auctions with administrative information for each one (such as the highest bid, seller’s reserve price, and remaining duration of the auction). The mediator receives requests from sellers to create an auction, bids from buyers, and, if the type of auction allows it (e.g. in an English auction), informs each buyer agent about a new bid, as well as the outcome.

The buyer agents implement the bidding strategies of their users (characterized by parameters such as the starting price, maximum price, and rate of increase). They also inform them about important events regarding the current auction (e.g. winning or losing the auction). While buyer agents only notify a user about the current auction, notifier agents can be used to monitor other auctions on a user’s behalf. The user creates a notifier agent by initializing it with a specification of the product or service she is looking for. On receiving a notification about an auction, a user can decide to create a new buyer agent to join it.

LOCATING SELLERS BY PRODUCT OR SERVICE

This example switches the positions of buyer and seller agents in the online auction example. Bob wants to buy a certain product or service, and via his user agent creates a buyer agent and initializes it with a specification of the product or service. This buyer agent queries a service agent – in this case, a search agent or a directory agent – for the locations of sellers for the product or service in question. At this architectural level, we could use static or mobile agents to represent buyer agents. Greengrass et al [11] describe the mobile agent scenario in which a buyer agent moves to the known site of a directory agent, queries this directory agent, and obtains an itinerary of the locations of seller agents to visit.

CUSTOMIZING NAVIGATION

Another example is using agents to assist users in navigating through a product catalog. From a front-end perspective, a user agent can be used to collect profile information about a user, and control how this information is made available to seller agents. The profile information for different vendors is stored in a personal profile repository in the user agent. Sections of the profile can be restricted to a subset of the vendors only.

Using the profile information the seller agent can classify the user into predefined stereotypes and thus infer a profile of product preferences. With these preferences, a product extractor agent can tailor the contents of the product catalog (such as in the personalized webstore in [2]). In addition to filtering the catalog contents against the buyer’s profile, the product extractor can propose discounts to retain existing or attract new customers.

Although we have focused on using user profiles in the front-end, this is not to say that they cannot equally be used in the back-end. However, the basic trade-off in this case is that users will have less control over profiles collected and stored by vendors. Customers may decide to do business with vendors who do not collect profiles in this way, or make the process of collecting profiles transparent and the profiles themselves accessible.

Concluding Remarks

This paper describes a group of architectural patterns for agent-based e-commerce, Agent as Delegate and its derivatives, and gives several examples illustrating how to use them. While these patterns relate to front-end e-commerce activities that involve interaction with the user, and delegation of tasks, a companion paper will describe patterns for back-end e-commerce activities that do not involve direct interaction with the user, but rather depict mechanisms for mediating between agents representing users. Together, these patterns are just the beginnings of a pattern language for agent-based e-commerce system design, based on our current understanding of the technology. As e-commerce matures and our understanding of how we can use agents in e-commerce applications improves, this language will grow as well.

Acknowledgements

Thanks to Dirk Riehle for many fruitful discussions during the shepherding process.

References

1. Alexander, C., The Timeless Way of Building, Oxford University Press, 1979
2. Ardissono, L., Barbero, C. et al, An Agent Architecture for Personalized Web Stores, Conference on Autonomous Agents (Agents-99), ACM, 182-189, 1999
3. Aridor, Y., Lange, D., Agent Design Patterns: Elements of Agent Application Design, Conference on Autonomous Agents (Agent-98), 108-115, ACM, 1998
4. AuctionWatch.com, Auction Manager, http://www.auctionwatch.com/, last accessed April 2001
5. Brenner, W., Zarnekow, R., and Wittig, H., Intelligent Software Agents: Foundations and Applications, Springer, 1998
6. Carlson, D., Modeling XML Applications with UML : Practical e-Business Applications, Addison-Wesley, 2001
7. Collis, J., and Lee, L., Building Electronic Marketplaces with the ZEUS Agent Tool-Kit, in [3], 1-24, 1999
8. Deugo, D., Weiss, M., and Kendall, L., Reusable Patterns for Agent Coordination, in: Omicini, A. et al (eds.), Coordination of Internet Agents, Springer, 2001
9. Ferber, J., and Gutknecht, O., A Meta-Model for the Analysis and Design of Organizations in Multi-Agent Systems, International Conference on Multi Agent Systems (ICMAS-98), ACM, 1998
10. Ferber, J., Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence, Addison-Wesley, 13-16, 1999
11. Greengrass, E., Sud, J., and Moore, D., Agents in the Virtual Marketplace, Component Strategies, 42-52, SIGS, June 1999
12. Griss, M., My Agent Will Call Your Agent ... But Will It Respond?, HP Labs Technical Report, HPL-1999-159, 1999, http://www.hpl.hp.com/techreports/1999/HPL-1999-159.html
13. Guttman, R., Maes, P., Agent-Mediated Integrative Negotiation for Retail Electronic Commerce, in [3], 70-90, 1999
14. Kendall, E., Murali Krishna, P., Pathak, C. et al, Patterns of Intelligent and Mobile Agents, Conference on Autonomous Agents (Agents-98), ACM, 1998
15. Kendall, E., Role Models: Patterns of Agent System Analysis and Design, Agent Systems and Applications/Mobile Agents (ASA/MA-99), ACM, 1999

16. Lieberman, H., Integrating User Interface Agents with Conventional Applications, Conference on Intelligent User Interfaces (IUI-01), ACM, 1998
17. Maes, P., Agents that Reduce Work and Information Overload, Communications of the ACM, 31-41, July 1994
18. Noriega, P., and Sierra, C. (eds.), Agent-Mediated Electronic Commerce, Lecture Notes in Artificial Intelligence, LNAI 1571, Springer, 1999
19. Open Profiling Standard, version 1.0, submitted to the W3C as note 1997/6, 1997
20. Papazoglou, M., Agent-Oriented Technology in Support of E-Business, Communications of the ACM, 71-77, April 2001
21. Reich, J., Ontological Design Patterns: Modelling the Metadata of Molecular Biological Ontologies, Information and Knowledge, Database and Expert System Applications (DEXA-00), 2000
22. Riehle, D., and Gross, T., Role Model Based Framework Design and Integration, Conference on Object-Oriented Programs, Systems, Languages, and Applications (OOPSLA-98), 117-133, ACM, 1998
23. Steinmetz, E., Collins, J., Jamieson, S., et al, Bid Evaluation and Selection in the MAGNET Automated Contracting System, in [3], 105-125, 1999
24. Stuckenschmidt, H., A Pattern-Driven Approach to Ontology Language Customization, http://www.informatik.uni-bremen.de/~heiner/Ontology-Patterns.pdf
25. Sycara, K., Multiagent Systems, AI Magazine, 79-92, Summer 1998
26. W3C, XSL Transformations, http://www.w3.org/TR/xslt
27. Weiss, M., Gray, T., and Diaz, A., Experiences with a Service Environment for Distributed Multimedia Applications, Feature Interactions in Telecommunications and Distributed Systems, IOS, 1997
28. Weiss, M., Patterns for the Design of Agent-Based Electronic Commerce Back-end Systems, companion paper to this paper, 2001 (in progress)
29. Wooldridge, M., and Jennings, N., Intelligent Agents: Theory and Practice, The Knowledge Engineering Review, 10(2):115-152, 1995
30. Zacharia, G., Moukas, A., and Maes, P., Collaborative Reputation Mechanisms in Electronic Marketplaces, Hawaii International Conference On System Science (HICSS-99), IEEE, 1999
31. Zambonelli, F., Jennings, N., et al, Agent-Oriented Software Engineering for Internet Applications, Omicini, A. et al (eds.), Coordination of Internet Agents, Springer, 2001