Pattern-Driven Design of Agent Systems: Approach and Case Study

Michael Weiss

School of Computer Science, Carleton University, Ottawa, Canada
weiss@scs.carleton.ca

Abstract. Current approaches to agent system design are generally goal-driven. An agent system is designed by iteratively decomposing system goals until they can be assigned to individual agents. However, this may lead developers to rediscover solutions to common design problems without benefiting from how they were resolved in the past. This results in duplicated effort, inconsistent design, brittle systems, and poor traceability. A more effective approach is to build an agent system incrementally from well-documented agent patterns. An agent pattern documents a proven assignment of roles to agents, and their interaction. It also documents the system qualities achieved by the application of this pattern. Individual patterns can, furthermore, be linked to each other in the form of pattern languages, which guide the designer through the design process. In this paper we describe a pattern-driven agent design process that complements goal-driven design approaches. What makes our approach different from most other pattern-based approaches is the use of softgoals for representing the system qualities affected by a pattern. We demonstrate the approach by applying it to a problem in the domain of agent-based electronic commerce.

1 Introduction

Current approaches to agent system design such as Gaia [15], and MASSIVE [11], are generally goal-driven. An agent system is designed by iteratively decomposing system goals until they can be assigned to individual agents. However, this may lead developers to rediscover solutions to common design problems without benefiting from how they were resolved in the past. This results in duplicated effort, inconsistent design, brittle systems, and poor traceability.

A more effective and less ad hoc approach is to build an agent system incrementally from well-documented agent patterns. Each agent pattern documents a proven assignment of roles to agents, and their interaction. It also documents the system qualities achieved by the application of this pattern. A pattern can therefore be considered a rule that transforms the system in such a way as to achieve certain desired non-functional requirements. Furthermore, individual patterns can be linked to each other in the form of pattern languages, which guide the designer through the design process.

J. Eder and M. Missikoff (Eds.): CAiSE 2003, LNCS 2681, pp. 711–723, 2003.

In this paper we describe a pattern-driven agent design process that complements goal-driven design approaches. This process consists of identifying the core design trade-offs for the application domain, documenting the roles agents can play, documenting patterns and their dependencies, identifying the overall system goals, and iteratively selecting and applying patterns. What makes our approach different from most other pattern-based approaches is the use of softgoals [5] for representing the system qualities affected by a pattern. We demonstrate the approach by applying it to a problem in the domain of agent-based electronic commerce.

2 Agent Patterns

Patterns are reusable solutions to recurring design problems, and provide a vocabulary for communicating these solutions to others. The documentation of a pattern goes beyond documenting a problem and its solution. It also describes the *forces* or design constraints that give rise to the proposed solution [1]. These are the undocumented and generally misunderstood features of a design. Forces can be thought of as pushing or pulling the problem towards different solutions. A good pattern balances the forces.

Patterns are not used in isolation. Although individual patterns are useful at solving specific design problems, we can benefit further from positioning them among one another to form a *pattern language*. Each pattern occupies a position in a network of related patterns, in which each pattern contributes to the completion of patterns "preceding" it in the network, and is completed by patterns "succeeding" it.

A pattern language guides developers through the process of generating a system. Beck and Johnson [4] describe this generative quality of patterns: "Describing an architecture with patterns is like the process of cell division and specialization that drives growth in biological organisms. The design starts as a fuzzy cloud representing the system to be realized. As patterns are applied to the cloud, parts of it come into focus. When no more patterns are applicable, the design is finished."

There is by now a growing literature on the use of patterns to capture common design practices for agent systems [3, 8, 7, 10]. 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 [8] capture common building blocks for the internal architecture of agents in patterns.

Deugo and Weiss [7] identify a set of patterns for agent coordination, which are, again, domain-independent. They classify agent patterns into architectural, communication, traveling, and coordination patterns. They also describe an initial set of global forces that push and pull solutions for coordination. Kendall [9] reports on work on a domain-specific catalog of patterns developed at BT. Weiss [14] describes a pattern language for agent-based e-commerce. In related work, Kolp and Giorgini [10] document organizational styles for multi-agent systems using the Tropos framework.

The separate notion of an *agent pattern* can be justified by differences between the way agents and objects communicate, their level of autonomy, and social ability [6]. Agent patterns are documented in a similar manner as other software patterns, except

for the structure of an agent pattern where we will make use of role models [13, 9] instead of collaboration diagrams. The distinction between role models and collaboration diagrams is the level of abstraction: a collaboration diagram shows the interaction of instances, whereas a role model shows the interaction of roles to be filled.

3 Pattern-Driven Agent System Design

Instead of proceeding from high-level goals and arriving at an implementation through iterative refinement, in a pattern-driven approach we start from proven solutions, and *compose* our system by systematically instantiating patterns. The goal-driven and pattern-driven approaches to design are complementary as shown in Fig. 1. Our experience building agent systems suggests that a design approach that proceeds both in a top-down and a bottom-up direction in parallel will lead to the best results

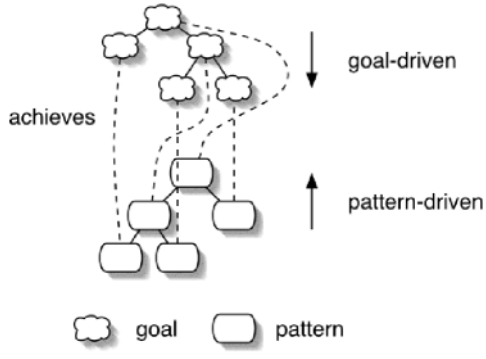

Fig. 1. Complementarity of the pattern-driven and goal-driven approaches

The implementation of the pattern-directed approach involves the steps shown in Fig. 2. The process provides two entry points, one for the pattern author, the second for the pattern user. The first three steps provide an approach for harvesting recurring design solutions and making them available as patterns. The last two steps start with a given set of patterns or pattern languages, and guide the designer through the selection of patterns appropriate to their specific application requirements.

Certain steps of this process can be supported by tools, eg the documentation of patterns, and the selection of patterns given a set of requirements. However, only partial implementations of such tools exist, for example, modeling tools that support the definition of patterns. However, the kind of tool we have in mind goes much further in supporting the designer in selecting the appropriate pattern for their needs.

Identify Domain Forces. For a given domain identify the core design trade-offs (called forces in patterns) that push and pull the design into different directions. In

addition, there are forces motivating the use of agents (such as autonomy, need to interact, multiple interfaces, and adaptability) to be considered for all domains.

Document Roles. Document the roles (and their subtypes) that will be used in the patterns. Individual patterns document how these roles interact in a given design context. The task of the designer is, by selecting patterns, to assign roles to agents. Documenting the roles separately from the patterns, as does identifying the domain forces, provides a point of reference for the following description of the patterns.

Document Patterns and their Dependencies. Document the patterns and their dependencies in the form of a pattern language. Each pattern should document the forces it helps resolve, and how instantiating the pattern will change the system. This includes the resulting role model and the forces that still need to be resolved. Semi-formal methods can be used to document the forces in a pattern. For example, in [2] we have investigated the use of softgoals to document forces and their trade-offs. Here we use the concept of softgoal as defined in [5]: a softgoal represents a non-functional requirement. The prefix "soft" indicates that they are often subjective in nature.

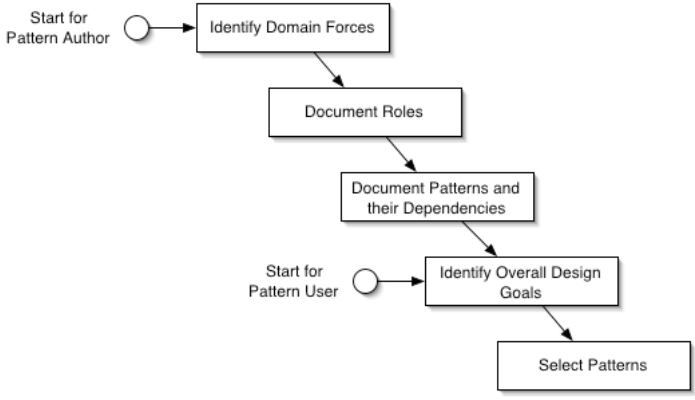

Fig. 2. Steps of the pattern-driven agent design process

Identify the Overall Design Goals. Identify the overall design goals, both functional and non-functional. Generally, the identification of patterns based on merely functional goals is rather straightforward. However, although multiple patterns may satisfy the *same* functional goals, their implications on the design in terms of non-functional goals must be carefully considered. The main thrust of our semi-formal pattern representation is geared towards matching on non-functional goals.

Select Patterns. In a first pass, select patterns based on how well they match the functional goals of the system, and then refine the selection by considering non-functional design goals. Compare the patterns and rank them on basis of their com-

patibility with these goals. In our tool we plan to use the algorithm described in McPhail and Deugo [12]. Repeat this step until all forces have been resolved.

4 Case Study

As a way of demonstrating the pattern-driven design approach, let us now look at a problem from the domain of agent-based electronic commerce. Consider that we want to design an *agent-based online auction* system. In an online auction system, users post items for sale, while other users place bids on those items. Users are faced with a multitude of auctions in which they can participate. Users may also need help in selecting a suitable auction, given that the number and types of items for sale often change frequently. Users also want to be informed about the status of an auction.

4.1 Identify Domain Forces

For the agent-based e-commerce domain, our analysis (see [14] for more details) identified the following forces that a design needs to trade-off. Fig. 3 shows these forces and their interactions. The label "agrees" indicates that the forces mutually support each other. The label "disagrees" means that they are in conflict, and need to be balanced.

Autonomy. An autonomous agent does not require the user's approval at every step of executing its task, but can act on its own. With agents performing autonomous actions, users are now facing the issues of trust and control over their agents.

Need to interact. Agents 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 interference with each other.

Information overload. Users wish to find relevant information and offerings to make good deals and generate profit. The large set of traders in conjunction with their many different interfaces makes it difficult for a user to overview the market.

Multiple interfaces. One of the difficulties in finding information (eg when comparing offerings) is the number of interfaces used to present the information. One solution is to use common vocabularies, but these must also be widely adopted.

Ensuring quality. Electronic commerce lacks the immediate mechanisms for establishing trustworthiness. How can buyers trust sellers, with whom they had no previous encounter, that their order will be fulfilled satisfactorily? Even if rating mechanisms (based on the identity of the trader) are used how can we ensure that a trader cannot easily assume a new identity and start afresh with a new rating?

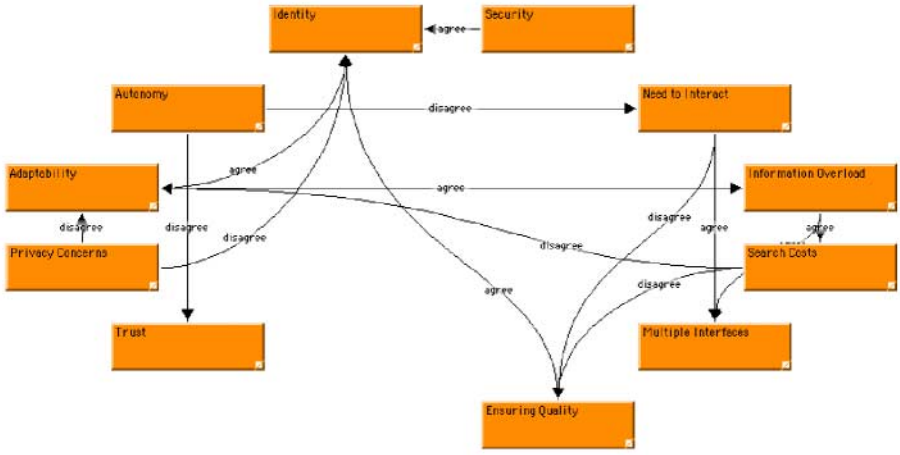

Fig. 3. Domain forces for the e-commerce domain

Adaptability. As users differ in their status, level of expertise, needs, and preferences, we need to tailor information to the features of the user, for example, by selecting the products most suitable for them 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, and thus raises privacy concerns.

Privacy concerns. Personalization requires that personal information, such as a buyer's preferences for sellers are made available to the agent providing personalized service. At the same time, users want to remain in control, and decide on an interaction-by-interaction basis which information is conveyed to another party.

Search costs. It can be expensive for traders to find each other. Electronic marketplaces are dynamic. Buyers and sellers can join and leave the marketplace, and change their needs and offerings qualitatively and quantitatively at any point in time. Thus it is difficult for market participants to keep an up-to-date list of contacts.

Identity. For various reasons, users need to be represented by unique identities. The most important are authentication (to ensure quality), repudiation, and tracking (to adapt to the user). However, tracking a user's behavior raises privacy concerns.

Trust. Trust in the sense of trusting another party with sensitive information includes storing the sensitive information (similar to privacy concerns), and trusting that proper use be made of the information (eg credit card information, age, or body measurements) are used only within the context of the current transaction.

Security. Security in the sense that the user's identity, and any sensitive information associated with it is protected. This is different from privacy concerns.

4.2 Document Roles

In our analysis, we identified four top-level roles for agent-mediate e-commerce systems as User, Task, Service, and Resource. Fig. 4 shows these roles with their subroles. The *User* role encapsulates the behavior of providing an access point for the user, managing a user's task agents, 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 represent the user either as a buyer or a seller.

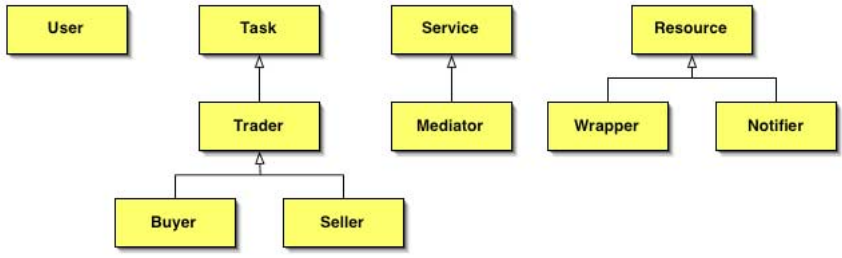

Fig. 4. Role hierarchy for e-commerce agents

Service roles typically provide a service to a group of users. They mediate the interaction between two or more agents through services. 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 (represented summarily by the Mediator role) facilitate more complex interactions between agents, for example, enforcing an auction protocol.

The *Resource* role abstracts from 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 (this glue is usually referred to as a wrapper). These can be agents that process more complex queries by first breaking them down into subqueries and then collating the results. Notifier agents, which monitor a data source for changes, and may filter for certain conditions, also belong into this category.

4.3 Document Patterns and Their Dependencies

The structure of the pattern language is shown in Fig. 5. The arrows indicate refinement links between the patterns. Each arrow in the diagram points in the direction from a "larger" pattern to a "smaller" pattern. The starting point for the language is the Agent Society pattern, which motivates the use of agents for building the application. At the next level of refinement, the diagram leads the designer to consider the patterns Agent as Delegate, Agent as Mediator, and Common Vocabulary.

Agent as Delegate and the patterns it links to deal with the design of agents that act on behalf of a single user. Agent as Mediator guides the designer through the design of agents that facilitate between a group of agents and their users. Common Vocabulary provides guidelines for defining exchange formats between agents.

The rest of Fig. 5 shows refinements of the Agent as Delegate pattern. For example, the User Agent pattern prescribes to use a single locus of interaction with the user, and represent the concurrent transactions a user participates in as buyer and seller agents. User interaction also includes profiling the user (User Profiling), and subscribing to information (eg the status of an auction) relevant to the user (Notification).

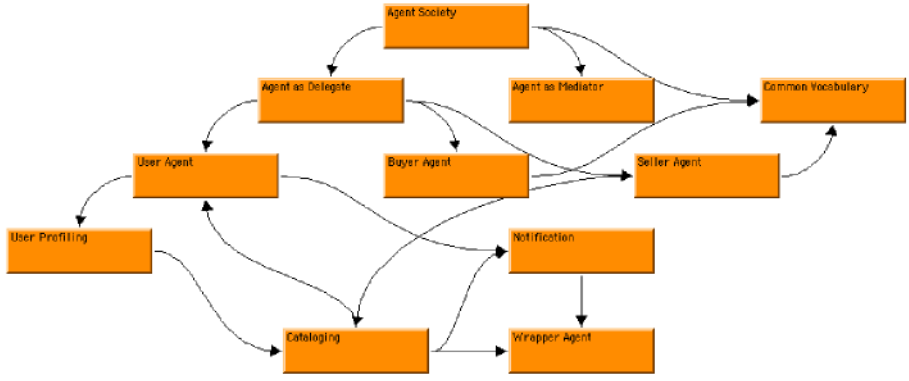

Fig. 5. Patterns for e-commerce agents and their dependencies

Each pattern is represented by its context, the problem, a discussion of the forces, and its solution. The context is represented by the dependencies between the patterns. The problem is a succinct statement on what problem the pattern addresses. The solution takes the form of a role diagram. These roles will be filled by agents.

For example, consider the User Agent pattern. It is applied after the Agent as Delegate pattern, and, in turn, refined by User Profiling, and Notification. The problem it addresses is how users instruct agents to act on their behalf (as buyers and sellers), and how they keep in control over what the agent does (eg does it have authority to complete a trade?). The role diagram for the User Agent pattern is shown in Fig. 6.

We use softgoals [5] to represent the forces a pattern helps achieve, or prevents from achieving (that is, both positive and negative contributions). A link between a pattern and a force in the softgoal graph indicates that the pattern directly contributes to its achievement. Fig. 7 shows the contributions of the User Agent pattern.

Each pattern is the result of a trade-off/balancing of forces. Representing the contributions of a pattern as a softgoal graph makes the contributions of the pattern toward achieving the domain forces explicit. It highlights the trade-offs made by a pattern, for example, if it achieves certain forces but hinders the achievement of other forces. It also makes visible the forces that remain unresolved after applying a pattern.

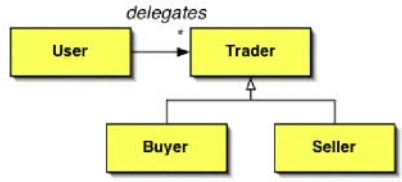

Fig. 6. Role diagram for the User Agent pattern

When we want to evaluate the effect of applying several patterns, we can combine the softgoal graphs for the individual patterns, and obtain, as a result, a softgoal graph in which the patterns are operationalizations. We can also see the result of applying alternative solutions to the same problem represented by different patterns. The choice of the pattern to use then depends on the prioritization of the forces by the designer.

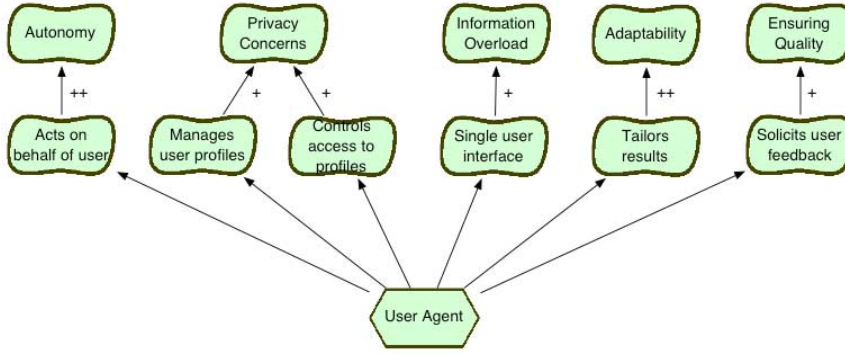

Fig. 7. A softgoal graph shows which forces the User Agent pattern helps achieve

Our notation for softgoal graphs is based on [5]. Softgoals are indicated as clouds, and patterns (that is, operationalization) are shown as hexagons. The labels on the arrows indicate the strength of the contribution, which can range from -- (breaks) and – (hurts) to + (helps) and ++ (makes the achievement of a softgoal).

In Fig. 7, the User Agent pattern helps address privacy concerns in two ways: one is by managing profiles on behalf of the user (the profile is collected at the user's end), the other is by controlling who can access those profiles, which portions of them, and to what use (in the sense of the P3P protocol for privacy protection).

4.4 Identify the Overall Design Goals

Fig. 8 lists the forces implied by the requirements of the online auction system stated earlier. Users are faced with a large number of auctions in which they can participate. Information overload is thus a driving force for the design of our system. In addition,

there are high search costs involved in selecting a suitable auction for an item, given that the number and types of items for sale change on a very frequent basis.

The design of such a system involves dealing with (potentially multiple concurrent) longstanding transactions. Users wish to delegate the task of bidding and monitoring the progress of the auctions they take part in to agents. This is reflected in the force of autonomy. Users may also want to be informed about newly created auctions for items of interest. Again they would confront information overload, if they were to monitor auction sites for newly created auctions manually.

Another concern is the varying quality of the items offered and the reliability of the sellers. An auction system should provide services to ensure the quality of items and reliability of participants. Finally, as a consequence of delegating bidding and monitoring tasks to agents, the user is revealing sensitive information to the agents (such as their price preferences) that must not be revealed to other agents or misappropriated in any way. This leads us to consider the user's privacy concerns.

Information Overload	Users participate in multiple concurrent auctions	User Agent Notification
Search costs	Cost of selecting a suitable auction for an item (high variability)	Mediator
Autonomy	Users wish to delegate bidding and monitoring progress	User Agent
Ensuring quality	Varying quality of items for sale and reliability of sellers	User Agent Mediator
Privacy concerns	Sensitive information is revealed to agents (eg maximum bid)	User Agent Mediator

Fig. 8. Overall design goals for the online auction

Fig. 8 also indicates (in the right column) the pattern(s) that help address the forces identified in the left column. This is the result of the next step, selecting patterns.

4.5 Select Patterns

The last step is concerned with selecting patterns. Having identified the overall design goals, and (optionally) a prioritization among them, we now select the patterns that help achieve them. Given the softgoal representation of each pattern we can identify the patterns that best help achieve the forces implied by the requirements. Here we perform this selection manually with the help of a reverse index that lists the patterns achieving a particular force. This can be derived from the individual softgoal graphs.

For example, an inspection of the User Agent pattern shows that it contributes to four of the overall design goals: information overload (+), autonomy (++), ensuring quality (+), and privacy concerns (++). However, User Agent does not provide all the

desired functionality, such as monitoring an auction, or helping find a suitable auction. Only its contribution to autonomy is strong enough that no other patterns are required to fully satisfy the non-functional design goals. For example, some of the information overload is caused by the need to monitor auctions for their current status. The Notification pattern addresses (+) this design concern. The Mediator pattern fully addresses the search cost (++) force, while raising some privacy concerns (–).

More complex schemes, for example, using a weighted distance metric (where each force is weighted by its priority to the designer), or a decision-theoretic model such as the one explored by McPhail and Deugo [12], are required if the search for matching patterns involves a larger number of patterns than were considered in this case study. Such methods are amenable to tool support, and for the future we envision that such selection algorithms can be embedded into standard object modeling tools.

Fig. 9 shows the combined softgoal graph with the contributions of each pattern. It allows us to validate that the overall design goals have been achieved. But it also points us to a potential issue wrt privacy concerns that we need to follow up on.

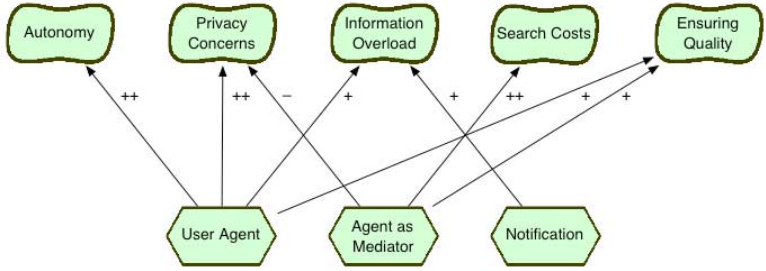

Fig. 9. Contributions of each selected pattern shown in a combined softgoal graph

Figure 10 shows how the role diagrams for the three patterns are instantiated into a concrete agent system. In the solution generated by applying the patterns of our pattern language, user agents manage trader agents that represent their users in multiple concurrent transactions (as buyers and seller). 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 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 that auction. A mediator agent implements the auction mechanism. 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 (such as the current bid, the seller's reservation price, and the remaining duration of the auction), and oversees the execution of the auction (creating auctions, receiving bids, informing buyers/sellers).

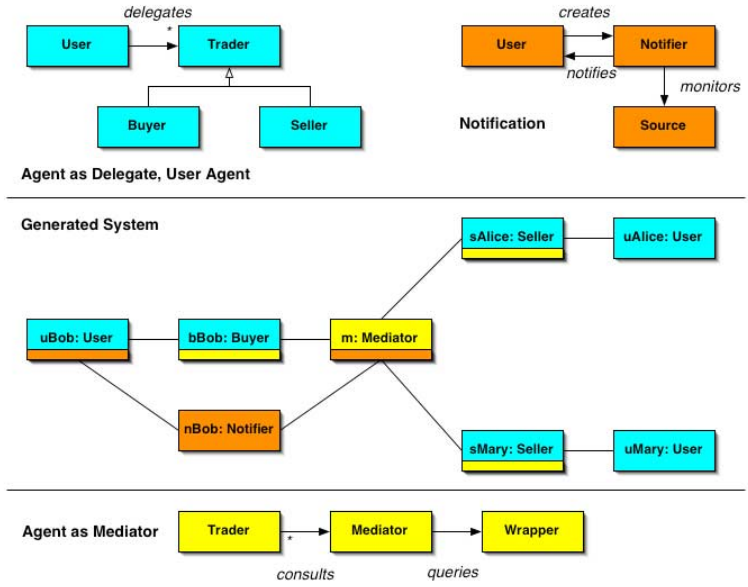

Fig. 10. Result of instantiating the selected patterns

5 Conclusion

In this paper we described a pattern-driven approach to agent system design. This approach is complementary to the goal-driven approach that most published methodologies use. The combined design approach proceeds, in parallel, in a top-down (goal-driven) and a bottom-up (pattern-driven) direction.

We illustrated the approach with a case study from the domain of agent-based electronic commerce. For space limitations we could only present the example of designing a system for online auctions. However, we have applied the approach to several other scenarios such as a yellow-pages lookup service, and customized navigation. Given our prior experience with applying a softgoal-based approach to modeling patterns in other domains such as distributed programming (see [2]), we are confident that these results can be generalized into the pattern-driven design process as described.

More work is required to document the forces that govern agent system design, and to understand their interactions. A related open problem is the selection of a pattern that is compatible with the stated functional and non-functional goals of an application. We are working on tool support to automate this task.

References

1. Alexander, C., A Pattern Language, Oxford University Press, 1977
2. Araujo, I., and Weiss, M., Using the NFR Framework for Representing Patterns, Conference on Pattern Languages of Programming (PLoP-02), 2002
3. Aridor, Y., Lange, D., Agent Design Patterns: Elements of Agent Application Design, Second Intl. Conference on Autonomous Agents, IEEE, 1998
4. Beck, K., and Johnson, R., Patterns Generate Architectures, European Conference on Object Oriented Programming (ECOOP-94), 139–149, 1994
5. Chung L., Representing and Using Non-Functional Requirements: A Process-Oriented Approach, Department of Computer Science University of Toronto, 1993
6. Deugo, D., Oppacher, F., et al, Patterns as a Means for Intelligent Software Engineering, Intl. Conference on Artificial Intelligence (IC-AI 99), 605–611, 1999
7. Deugo, D., Weiss, M., and Kendall, L., Reusable Patterns for Agent Coordination, in: Omicini, A., et al (eds.), Coordination of Internet Agents, Springer, 2001
8. Kendall, E., Murali Krishna, P., Pathak, C. et al, Patterns of Intelligent and Mobile Agents, Second Intl. Conference on Autonomous Agents, IEEE, 1998
9. Kendall, E., Role Models: Patterns of Agent System Analysis and Design, Agent Systems and Applications/Mobile Agents (ASA/MA-99), ACM, 1999
10. Kolp, M., Giorgini, P., Mylopoulos, J., A Goal-Based Organizational Perspective on Multi-Agent Architectures, Eighth Intl. Workshop on Agent Theories, Architectures, and Languages (ATAL-2001), 2001
11. Lind, J., Iterative Software Engineering for Multi-Agent Systems: The MASSIVE Method, LNCS 1994, Springer, 2001
12. McPhail, J.C., and Deugo, D., Deciding on a Pattern, 14th Intl. Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE-01) LNCS 2070, Springer, 2001
13. 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
14. Weiss, M., Patterns for e-Commerce Agent Architectures: Using Agents as Delegates, Conference on Pattern Languages of Programming (PLoP-01), 2001
15. Wooldridge, M., Jennings, N., and Kinny, D., The Gaia Methodology for Agent-oriented Analysis and Design, Journal of Autonomous Agents and Multi-Agent Systems, 2002