ENGINEERING AGENT-BASED INFORMATION SYSTEMS
A Case Study of Automatic Contract Net Systems

Vincent Couturier, Marc-Philippe Huget and David Telisson
LISTIC – Polytech'Savoie, Université de Savoie, B.P. 80439, F - 74944 Annecy-le-Vieux Cedex, France

Keywords: Enterprise Information Systems, Agents, Software Patterns, Reuse.

Abstract: In every business the tender has become an indispensable part to foster the negotiation of new trade agreements. The selection and the attribution are nowadays a long process conducted manually. It is necessary to define criteria for selecting the best offer, evaluate each proposal and negotiate a business contract. In this paper, we present an approach based on agents for the development of an automatic award of contracts (here called Automatic Contract Net Systems). The selection and negotiation are then automatically performed through communication between agents. We focus in this paper on the tendering and selection of the best offer. To facilitate the development of complex systems such as multi-agent systems, we adopt software patterns that will guide the designer in the analysis, design and implementation on an agent-based execution platform.

1 INTRODUCTION

Historically, companies have felt the need to communicate and share information in order to create economic partnerships. The concept of extended enterprise has spread widely in the industrial world where it is common now to launch a tender via a website in order to award a new contract to a subcontractor. However, beyond the tender and deposit records, the whole process (receipt of nominations, terms of partnership and implementation) is rarely automated.

Various research works have focused on management of tenders by developing information systems. These approaches allow to deploy systems upstream (dissemination and consultation of proposals) and downstream (data exchange) of the awarding contracts. However, the selection of tenders and awarding contracts still remain a binding process as it requires decision making and strong autonomy of the actors involved.

In this paper we propose an approach and tools (software patterns) to design an automatic contract net system (ACNS) based on intelligent agents. The choice to develop our solution with intelligent agents comes from the autonomy requirement given above. Indeed, the tasks of an award procedure require that different market actors (suppliers, customers, partners, etc.) take decisions correlated to the sociological and environmental parameters defined by characteristics to their field of activities. These autonomous decisions should promote cooperation and coordination tasks to end in an economic partnership. It is necessary to have a system able to search for the best offer according to given criteria and to negotiate with the contractor if his/her proposal is not fully satisfactory. We choose an intelligent-based agent solution since each intelligent agent presents intelligent decision making, and autonomy regarding its actions. We adopt the Contract Net Protocol (Smith, 1980) (FIPA, 2002) for the whole process of proposing tasks, issuing proposals and enacting contract. As part of a multi-agent system, agents are active entities, autonomous, intelligent and goals directed. They will work together to achieve their personal goals (Wooldridge, 2002).

Agent theory is now well-developed and covers many domains from intelligent decision making, negotiation, trust to planning to name a few. Existing development methodologies (Gaia (Wooldridge *et al.*, 2000), MaSE (DeLoach *et al.*, 2001), etc.) for agent-based systems are not suitable, in general, for inexperienced developers. In particular, they may face various problems in the analysis or design stage because of their inexperience. This leads to inconsistent analysis or design and poor reliability of the resulting agent-based systems.

Couturier V., Huget M. and Telisson D. (2010).
ENGINEERING AGENT-BASED INFORMATION SYSTEMS - A Case Study of Automatic Contract Net Systems.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 242-248
Copyright © SciTePress

In this paper, we propose to use the concept of software patterns for the development process of agent-based solutions, like existing approaches such as Tropos (Giorgini *et al.*, 2005) or PASSI (Cossentino *et al.*, 2004).

Patterns we propose in this paper cover the whole development process from requirements analysis to code generation. The designer of an ACNS can thus follow our guidelines to develop his/her solution. Thus, it will lead novice developers to produce complex applications and to rapidly adapt to any implementation environment.

The following paragraph explains the concept of pattern. Then, we present the categories of patterns dedicated to engineering ACNS and the reuse process.

2 THE CONCEPT OF PATTERN

Alexander introduced the concept of pattern in 1977 for the design and construction of homes and offices (Alexander *et al.*, 1977). Patterns were adapted to software engineering and mainly to object-oriented programming and are called *software patterns*.

In Alexander's proposition, a pattern describes a problem which occurs over and over again in an environment as well as a solution that can be used differently several times. A *software pattern* follows the same principle and offers a solution to developers when building software in a specific context. Thus, patterns can be seen as abstractions used by design or code experts that provide solutions in different phases of software engineering.

Patterns can be divided into four categories: analysis patterns (Fowler, 1997), architectural patterns (Buschmann *et al.*, 1996), design patterns (Gamma *et al.*, 1995), and idioms (Coplien, 1992), the latter are also known as implementation patterns.

3 CATEGORIES OF PATTERNS DEDICATED TO AUTOMATIC CONTRACT NET SYSTEMS AND THEIR REUSE

3.1 Pattern Categories

The first patterns applied for engineering of Automatic Contract Net Systems are *Agent Analysis Patterns*. They define agent structure and design multiagent systems at a high level of abstraction. They can be applied to design either reactive or cognitive agents. As mentioned before, we propose to develop agent-based Automatic Contract Net Systems. Thus, the designer will be able to reuse these patterns to design agents for his/her ACNS at a high level of abstraction.

Agent Analysis Patterns are generic and have to be applied with *ACNS domain patterns*. These ones capitalize the domain knowledge specific to Automatic Contract Net Systems and favour its reuse.

Patterns dedicated to architectural representation and design of ACNS are *ACNS architectural patterns* and *design patterns*.

The former have to be applied at the beginning of the design process and help defining the ACNS structural organization. They represent the different architectural styles for ACNS which are means of capturing families of architectures and can be seen as a set of design rules that guide and constrain the development of ACNS architecture (levels, internal elements, collaborations between elements…). Architectural styles depend on which architecture we choose for the ACNS: Market-based one, Subcontract-based one or Peer-to-Peer-based one.

Design patterns describe technical elements required to develop agent-based Automatic Contract Net Systems. Analysis and conceptual models obtained by applying ACNS domain patterns and Agent Analysis Patterns are refined with behaviour, collaboration and software entities. Thus, the ACNS design model is obtained by adapting software elements specified in the design patterns solutions. Among them, we can distinguish generic *Agent Design Patterns* (dedicated, for instance, to define interaction protocols between agents such as Contract Net Protocol), and patterns specific to ACNS design, called *ACNS design patterns*.

Finally, we have specified two kinds of support patterns: *Model Transformation Patterns* and *Reuse Support Patterns*.

Model Transformation Patterns help ACNS developers to build applications from ACNS and Agent design patterns and can be applied at the end of the design phase. They specify transformation rules to map design models to models specific to agent development frameworks such as JADE (Bellifemine *et al.*, 2007), Madkit (Gutknecht & Ferber, 2000) or Mercury (Huget, 2008).

Reuse Support Patterns are process patterns which help developers navigating into a collection of patterns and reusing them. They describe, by using activity diagrams, a sequence of patterns to apply to resolve a problem. ACNS Reuse Support Patterns help developers to design and build such systems by guiding them among our collection of patterns.

The different patterns described here regarding the development cycle of an ACNS are shown on Figure 1.

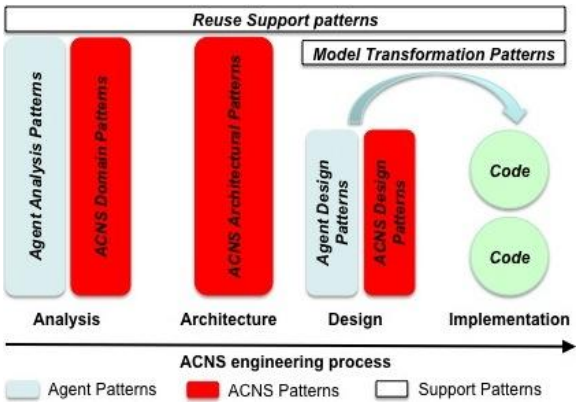


Figure 1: The use of the different proposed patterns in the development cycle of an ACNS.

Note: The different patterns presented here are shorter version ones.

3.2 Pattern Reuse

The reuse of patterns dedicated to develop ACNS consists in applying them during analysis, design and implementation phases. First, developers analyze context and problem and select relevant patterns. This activity can be favoured by using *Reuse Support Patterns* (see Table 1) which represent sequences of patterns that can be applied to develop Automatic Contract Net Systems. Thus, developers adapt pattern solution elements (instantiation) to represent the system they want to develop. Finally, the third activity aims at using Model Transformation Patterns to generate skeleton application from pattern instances.

We present, in next section, Agent Patterns and ACNS Patterns we designed to develop Automatic Contract Net Systems.

4 PATTERNS FOR ENGINEERING AUTOMATIC CONTRACT NET BASED SYSTEMS

4.1 Patterns for the Analysis Phase

Agent Analysis Patterns are generic ones used for building agents at a high level of abstraction:

- Pattern "Agent Structure": this pattern defines what an agent is in terms of its structure. An agent is a set of roles it will play during its execution.
- Pattern "Adding cognitive interaction to Role": this pattern inserts the notion of *protocol* and *message* within the agent. The agent is then able to communicate via high-level messages with other agents.
- Pattern "Adding behavior to cognitive interaction": this pattern defines the behavior associated to sending and receiving messages.

Two domain patterns were designed for the ACNS application. An Automatic Contract Net System considers two kinds of interacting entities: (1) one or more *Consumer* and (2) one or more *Producer*. The "Consumer" pattern defines the *Consumer* concept. The "Producer" pattern is equivalent to this one except it considers the concept of *Producer* instead of the concept of *Consumer*.

4.2 Patterns for the Design Phase

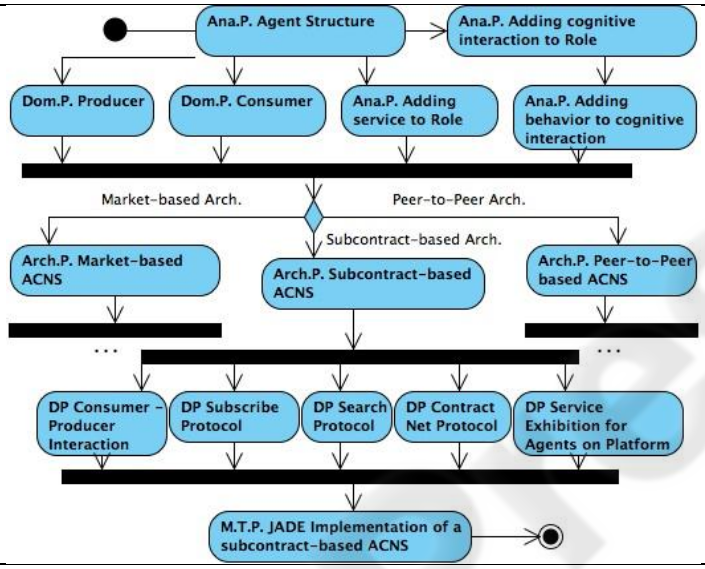
In this section, we present ACNS Architectural Patterns for the architectural design of a Contract Net system then patterns for the detailed design of an ACNS. These patterns have to be applied after analysis patterns and domain patterns described above.

4.2.1 ACNS Architectural Patterns

We develop three architectural patterns related to the different architectures an ACNS could have:

- Pattern "Market-based ACNS": a marketplace is defined with this pattern. A marketplace is composed of several consumers and several providers. Consumers try to find the best proposal for a service. Two approaches are possible to retrieve this best proposal: (1) A descending price auction or (2) A call for proposals.
- Pattern "Subcontract-based ACNS": An ACNS with subcontracts is a restricted version of the previous pattern "Market-based ACNS". In this particular case, there is only one consumer and several providers. The best proposal is found after a call for proposals.

Table 1: Reuse Support Pattern "Engineering an Automatic Contract Net System".

Interface
Name
Engineering Automatic Contract Net Systems
Classification
Reuse Support Pattern
Rationale
This pattern presents ACNS and agent patterns that can be applied to develop Automatic Contract Net Systems. *Note: We only describe here patterns for the development of a subcontract-based ACNS for JADE platform.*
Solution
Process


- Pattern "Peer-to-Peer-based ACNS": previous patterns impose to use a central server so as to store the address of the different consumers and providers. This approach does not resist to the scalability problem and the bottleneck is located on searching the central server to retrieve the different consumers and providers.

In this pattern here, there is no more central server and the different consumers and providers know each other via social networks. This kind of architecture copes with the scalability problem.

4.2.2 Agent Design Patterns and ACNS Design Patterns

These patterns describe the different concepts needed for designing an ACNS (ACNS Design patterns) and for designing a multiagent system used for ACNS (Agent Design Patterns). Only one pattern of the former is presented in abstract form (Cf. Table 2).

4.3 Patterns for the Phase of Implementation: Model Transformation Patterns

We define several Model Transformation patterns for developing ACNS for different architectures (subcontract-based architectures, market-based ones and peer-to-peer-based ones) and for different execution platforms (JADE, Madkit, and Mercury). Due to space restriction, we only present here in Table 3, a short version---without method transformations---of a Model Transformation pattern for JADE implementation of a subcontract-based ACNS.

Table 2: ACNS Design Pattern "Consumer-Producer Interaction".

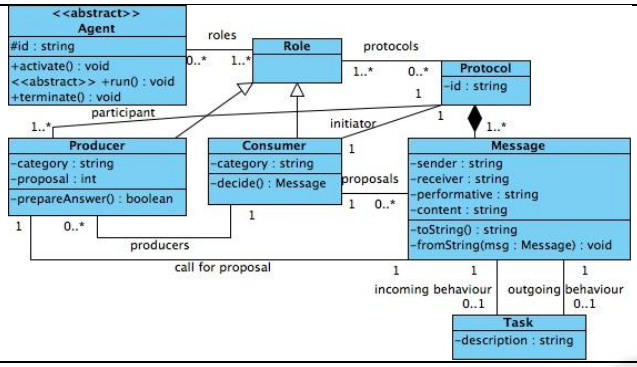
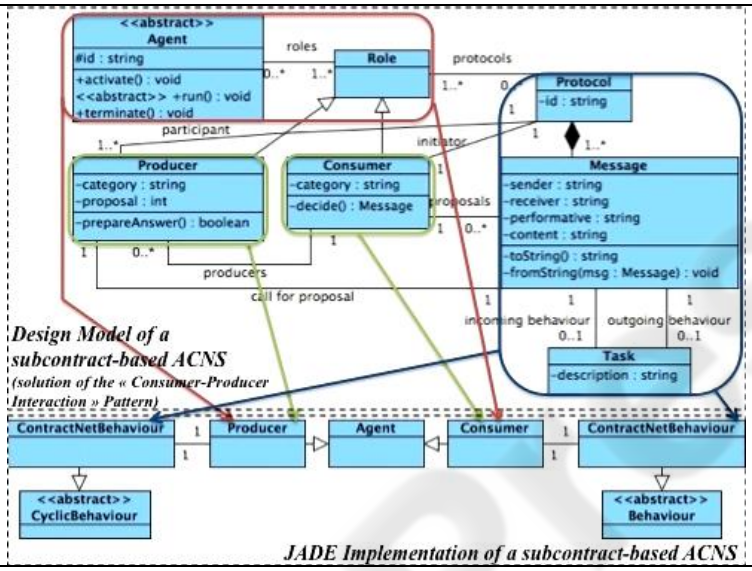
Interface
Name
Consumer-Producer Interaction
Classification
Design Pattern ^ACNS Pattern
Rationale
This pattern describes the interaction between *Consumer* agents and *Producer* agents and behaviours associated with this interaction.
Solution
Model

Participants
The *Agent* concept corresponds to the notion of agent defined in the agent theory. An agent is an autonomous, active entity which asynchronously interacts with other agents and cooperates with others so as to solve a global problem. The *Role* concept defines a catalogue of behaviours the agent will play within the multiagent system. The *Consumer* concept refers to the notion of customers in commerce. Customers acquire a service from providers. The *Consumer* concept inherits from the *Role* concept since it represents a specific role in a Contract Net system. A field *category* is associated to this concept to refer to the kind of service the *Consumer* is looking for. The *Producer* concept refers to the notion of producers in commerce. Producers propose services to customers. The *Producer* concept inherits from the *Role* concept since it represents a specific role in a Contract Net system. The *Producer* concept has two attributes: *category* and *proposal*. *Category* defines the kind of service the *Producer* provides. *Proposal* refers to the proposal the *Producer* issues when receiving a call for proposals from the *Consumer*. Here, the proposal is an amount for performing the service. The *Consumer* concept is related to the *Producer* concept through an association entitled *producers*. The consumers association describes a *Consumer* asks to 0 or more *Producer* for a service given criteria and a *Producer* provides service to a *Consumer*. It is possible a *Consumer* does not find a *Producer* if there is no *Producer* for the given criteria. The *Consumer* concept is related to the *Message* concept through an association entitled *proposals*. A *Consumer* sends 0 or more *Message* to *Producer* and a *Message* is related to only one *Consumer*. Exchanged messages are those found in the Contract Net Protocol pattern. The different proposals coming from the *Producer* are those in this association. The *Producer* concept is related to the *Message* concept through an association entitled *call for proposal*. A *Producer* receives one *Message* from *Consumer* and a *Message* is related to only one *Producer*. This message corresponds to the call for proposals the *Consumer* does in the context of the Contract Net protocol (Contract Net Protocol pattern). The *Protocol* concept defines a protocol. This *Protocol* is composed of an attribute *id* that uniquely identifies the protocol. A protocol is a legal suite of sequences of messages. The *Message* concept refers to the notion of messages in protocols. A message has several attributes: *sender* and *receiver* which respectively refer to the sender of the message and the receiver of the message. The attributes *performative* and *content* correspond to the content of the message for the latter and the illocutionary force--the verb--for the former. A performative is the verb used to describe the action and is applied on the content. The *Task* concept refers to the notion of behaviours. Here, behaviours correspond to the behaviours when sending and receiving messages. Behaviours when sending messages are denoted by the *outgoing behaviour* association between the *Task* concept and the *Message* concept. Behaviours when receiving messages are denoted by the *incoming behaviour* association between the *Task* concept and the *Message* concept. It is optional to describe behaviours when designing protocols and messages. The *Task* has one attribute *description* that gives the associated behaviour. There is no predefined format for this attribute. Designers may decide to use formal methods or programming languages, to represent only one action or a workflow depending on behaviour complexity.

Table 3: Model Transformation Pattern "JADE Implementation of a Subcontract-based ACNS".

Interface
Name
JADE Implementation of a subcontract-based ACNS
Classification
Model Transformation Pattern ^ ACNS Pattern
Rationale
An Automatic Contract Net System considers two kinds of interacting entities: (1) one or more *Consumer* and (2) one or more *Producer*. This pattern defines the *Consumer* concept.
Solution
Model

Participants
This pattern ensures the transformation from a conceptual model of an ACNS to a set of classes for the JADE platform. Agents on the JADE platform are defined as a specialization of the Agent class provided by the JADE platform. The *Consumer* and *Producer* concepts are derived as Consumer and Producer classes inheriting from the Agent class. The Agent class from the JADE platform provides the different methods required for the Agent lifecycle (creation, invocation, execution and deletion). These methods correspond to the ones proposed in the *Agent* concept. The set of attributes and methods from the *Role* concept is added to the Consumer and Producer classes. Behaviours on incoming messages and outgoing messages are defined via the *Message* and the *Task* concepts in the upper part of the figure. Behaviours in the JADE platform are defined as a specialization of the Behaviour class. We distinguish the Consumer behaviours from the Producer behaviours during the transformation. Actually, we consider the Producer behaviours as cyclic; the Producer learns from previous calls for proposals to prepare better proposals. It is not possible with the Behavior class to define cyclic behaviour hence explaining why the behaviour for Producer inherits from CyclicBehavior.

5 CONCLUSIONS

This paper describes our work about specifying and reusing patterns so as to engineer agent-based Automatic Contract Net Systems (ACNS). The different patterns presented here represent the building blocks which, after adaptation, can be used to develop analysis and design models for a new ACNS, define the architecture and ease implementation. The patterns cover the phases of analysis, design and implementation for engineering an ACNS.

The approach developed and the different patterns are experimentally validated on a specific ACNS for booking flight tickets. The client is here the travel agency and the providers are the different flight companies. Reusing the patterns help eliciting the business entities (analysis model), architecting the system (the architecture is a subcontract-based one since there is a unique consumer and several producers), defining the design model and generating skeleton code for the JADE platform.

We have developed a toolkit so as to ease engineering Information System applications and specifically ACNS. This toolkit is based on our software patterns. It takes as input a Reuse Support Pattern, guides the developer--by asking questions--through the different patterns to be used, and finally generates code skeleton. The process is then not fully automated due to developer's interventions. Thus, s/he can complete and refine the generated code and run his/her agents on a JADE platform.

Future work aims at reusing the different patterns presented here and augmenting the set of patterns so as to develop other Enterprise Information Systems (schedule management for instance).

REFERENCES

Alexander, C., Shikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S., 1977. *A pattern language: towns, buildings, construction*, New York, Oxford University Press.

Bellifemine, F. L., Caire, G., & Greenwood, D., 2007. *Developing Multi-Agent Systems with JADE*, New York, Wiley.

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P. & Stal, M., 1996. *Pattern-Oriented Software Architecture: A System of Patterns*, New York, John Wiley & Sons.

Coplien, J.O., 1992. *Advanced C++: programming styles and idioms*, Addison-Wesley.

Cossentino, M., Luca, S., & Antonio, C., 2004. Patterns Reuse in the PASSI Methodology, *In A. Omicini, P. Petta, J. Pitt (eds.), LNCS (LNAI): vol. 3071, ESAW 2003*, pp. 294-310, Springer.

DeLoach, S.A., Wood, M.F., & Sparkman, C.H., 2001. Multiagent Systems Engineering. *International Journal of Software Engineering and Knowledge Engineering*, 11(3).

Foundation for Intelligent Physical Agents, 2002. FIPA Contract Net Interaction Protocol Specification, from http://www.fipa.org/specs/fipa00029/

Fowler, M., 1997. *Analysis Patterns*, Addison-Wesley.

Gamma, E., Johnson, R., Helm, R., & Vlissides, J., 1995. *Design patterns, elements of reusable object-oriented software*, Addison-Wesley.

Giorgini, P., Kolp, M., Mylopoulos, J., & Castro, J., 2005. Tropos: A Requirements-Driven Methodology for Agent-Oriented Software, *Agent-Oriented Methodologies*, pp. 20-45.

Gutknecht, O., & Ferber, J., 2000. The MADKIT agent platform architecture, *In T. Wagner (Ed.), LNCS : vol. 1887, International Workshop on Infrastructure for Multi-Agent Systems*, Springer-Verlag, pp. 48-55.

Huget, M. Ph., june 2008. Mercury: une plate-forme pour l'exécution de systèmes multi-agents, Paper presented at the *8ème Conférence Internationale sur les NOuvelles TEchnologies de la REpartition (NOTERE 2008)*, Lyon, France (in French).

Smith, R.G., 1980. The Contract Net Protocol: High level Communication and Control in a Distributed Problem Solver, *IEEE transactions on Computers*, 29(12), pp. 1104-1113.

Wooldridge, M., Jennings, N. R., & Kinny, D., 2000. The Gaia Methodology For Agent-Oriented Analysis And Design, *Journal of Autonomous Agents and Multi-Agent Systems*, vol. 3, pp. 285-312.

Wooldridge, M.J., 2002. *An Introduction to MultiAgent Systems*, New York, Wiley.