E-COMMERCE TRANSACTION MODELING USING MODERATELY OPEN MULTI-AGENT SYSTEMS

A. Garcés, R. Quirós, M. Chover, J. Huerta
Department of Computer Systems, Jaume I University, Castellón, Spain

E. Camahort
Department of Computer Systems, Politechnic University of Valencia, Valencia, Spain

Keywords: Software Engineering, Agent Paradigm, e-Commerce, Transaction Systems.

Abstract: In this paper we describe how to completely develop a Multi-Agent System using the HABA Development Framework. We propose a variant of the GAIA methodology to reduce the gap between the abstract modeling of Multi-Agent Systems and their practical implementation. To achieve this goal, we reduce the scope of our methodology to a specific class of systems that we call Moderately Open Multi-Agent Systems. As an example, we use the implementation of a set of transactions for an electronic commerce system.

1 INTRODUCTION

Computer systems have evolved from single-processor centralized computing to multi-processor distributed computing over wide-area networks. In these systems, agents have become the choice of paradigm for programming. *Agent Oriented Software Engineering* has been proposed to understand, model and develop a class of cooperative, distributed systems called *Multi-Agent Systems* (MAS).

Design techniques for current MAS have many features. They allow the specification of task scheduling policies and information exchange protocols. Also, they support code migration and the definition of intentions and wishes for the system's components. However, implementation techniques have not made it so far, yet. They are based on extensions of high-level languages that maintain the artifacts and limitations of those languages.

GAIA (Wooldridge et al, 2000) was the first complete methodology for the analysis and design of MAS. GAIA offers a conceptual framework large enough to model agent-based systems focusing on the system analysis and design stages. However, its high abstraction level limits its application to real-world problems because GAIA is not related to any particular language or programming application.

To overcome these limitations we propose a GAIA-based methodology for the development of MAS (Garcés et al., 2007). Our methodology tries to reduce the gap between abstract modeling and practical implementation of a MAS in GAIA. The methodology, called *Homogeneous Agents Based Architecture* (HABA.DM) is suitable for a reduced set of MASs with the following features:

- We impose a static organizational structure. Agent classes and relationships do not change during execution time. Their skills and services are static.
- Agents are homogeneous; all are supported by the same language and execution platform.
- There is a centralized mechanism that creates, activates, shuts down and removes the agents of the system. It manages all the global resources and the interaction protocols of the application.

We call this type of MAS, *Moderately Open Multi-Agent Systems* (*MOMAS*). They provide a new model for agent definition and implementation. The features just described may seem unsuited for large-scale agent communities with high pro-activity. However, there is a large class of systems with reactive components that can benefit from our model; for example, monitoring and control systems, telecommuting services, and electronic commerce systems. Our model allows both high- and low-level behavior specifications. It comes with a programming language and a project manager for fast prototyping of MAS (Garcés et al., 2006).

Garcés A., Quirós R., Chover M., Huerta J. and Camahort E. (2007).
E-COMMERCE TRANSACTION MODELING USING MODERATELY OPEN MULTI-AGENT SYSTEMS.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - SAIC*, pages 167-172
DOI: 10.5220/0002397701670172

In this paper we describe how to completely develop a MOMAS using the HABA Development Framework. As an example, we use the implementation of a set of transactions for an electronic commerce system. We start giving some background information about multi-agent systems, the architecture of a MOMAS, and e-commerce systems. Then, we describe how to model and implement a Transaction System following the analysis, design and implementation stages of HABA Development Framework. We conclude our work with a discussion.

2 BACKGROUND

2.1 Agent-Oriented Systems

In the literature one may find many different methodologies for the development of agent-oriented systems like *BDI* (Kinny et al, 1996) or *MAS-CommonKADS* (Iglesias et al, 1998). GAIA (Wooldridge et al, 2000) was the first complete methodology for the analysis and design of MAS. The system was conceived like a computational organization of agents. Each agent has certain roles in the organization. The agents cooperate to achieve the common objectives of the application.

There have been theoretical studies and practical applications of GAIA, both emphasizing its strengths and weakness (Zambonelli et al, 2003). Cernuzzi (Cernuzzi et al, 2004) outlines two fundamental limitations of GAIA: first, it is not suitable for complex open environments and, second, it uses a notation not based on any widely-accepted Software Engineering standard. In (Cernuzzi et al, 2004) three different variants of GAIA are presented. They overcome some conceptual problems and extend the development of MAS to complex open environments. Still, none of them proposes how to implement MAS on real platforms.

2.2 Moderately Open Multi-Agent Systems (MOMAS)

As an alternative to regular MAS, we describe how to model and architect a MOMAS to organize agents in communities. Our methodology allows modeling and prototyping MOMAS with the architecture illustrated in Figure 1. Applications have a static structure since agent classes and their relationships do not change during execution. Services provided by the agents are also static. Agents are clustered into communities called *packages*. A *management module* within the MAS handles the life cycle of packages and agents. It also manages their communication, and the social state's public information. This module is a special agent that interfaces between the *social state*, the agents and the *user* in charge of running the system.

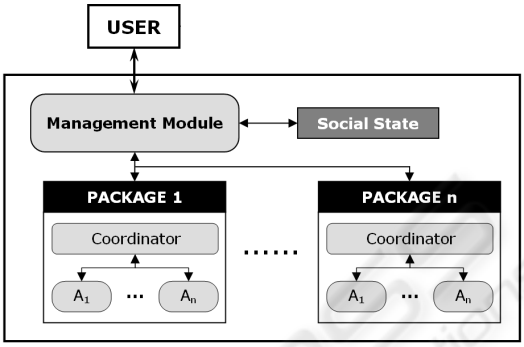

Figure 1: Architecture of a Moderately Open Multi-Agent System.

We propose a new development framework suitable for MOMAS architecture. It includes a *Development Methodology* (HABA.DM), a *Programming Language* (HABA.PL) and a *Project Manager* (HABA.PM) for fast prototyping of MOMAS. Describing the entire development framework is beyond the scope of this paper. The reader can find a complete description in the following references: (Omitted, 2006), (Omitted, 2007).

2.3 Electronic Commerce Systems

Since the beginning of Internet, electronic commerce has been one of its fastest growing applications. Most proposals are informal and based on Web development and programming. They consider functionality more relevant than formal specification.

There are three types of e-Commerce applications: formal based applications (Ehikiova, 2001), knowledge based applications (Tran, 2006), and applications based on traditional Software Engineering (Pastor et al, 2001). The last two types are closer to traditional programming practices. They include development tools that produce efficient implementations. Formal approaches, however, build models with higher reliability.

Some works have focused on MAS applications to e-Commerce (Karacapilidis and Moraitis, 2001). However, they do not provide an implementation approach like other methodologies such as Object-Oriented Programming. Our development framework supports rapid application prototyping keeping the formalism of MAS. We apply this framework to an application based on e-Commerce transactions.

We show how MOMAS simplifies processing and communications. This guarantees complete and efficient implementations particularly for small and medium enterprises.

3 AN E-COMMERCE TRANSACTION SYSTEM

We show how our development framework can be used to model and implement e-Commerce transaction systems. As an example we present a simple Ordering System with a limited number of transactions.

3.1 System Description

Our example Ordering System supports *Business to Consumers, B2C* e-Commerce. It is an improved version of the e-Commerce system programmed using OO technology by Ehikiova (Ehikiova, 2001). They use a formal specification to define the information resources. In our work, however, such resources are used at higher level of abstraction. High-level entities are autonomous agents that run in a MOMAS architecture.

We present an example company that markets products to an arbitrary number of customers. The company needs processes for product display and purchase. These processes include placing and completing orders, as well as payment management. Our example application is best suited for small virtual shops. Still, it suffices to show the features of our MOMAS development framework.

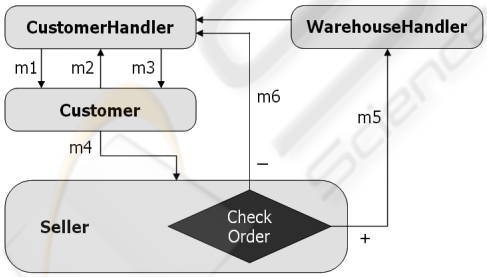

Figure 2: Flowchart of a purchase process.

We distinguish four main roles in our system: the *Customer*, the *CustomerHandler*, the *Seller* and the *WarehouseHandler*. The flowchart of a purchase is shown in Figure 2. A Customer starts the process by requesting a product catalog from the company (message m1). A CustomerHandler delivers the catalog (message m2). Then, the customer places an order by selecting a set of products. That set is sent to the CustomerHandler to complete the order (message m3). The CustomerHandler converts the order into a virtual *ShoppingCart*, which is in turn processed by a Seller (message m4). The Seller manages the order using the ShoppingCart: it validates the customer's payment and tells the WarehouseHandler to deliver the products if the payment is successful (message m5). Otherwise, the seller rejects the order and informs the Customer about the problem (message m6). The WarehouseHandler waits for shipping orders, collects the appropriate order paperwork and delivers the products to the customer.

The following sections describe the analysis, design and implementation stages of the ordering system. Sections 3.2 and 3.3 are devoted to the analysis and design stages using HABA.DM. Section 3.3 describes the implementation of the design in HABA.PL, and the execution of the system on the HABA.PM project manager.

We describe the process of role-guided development using the refinement process of the CustomerHandler role. This refinement proceeds through the different development stages. The other roles could also be similarly refined.

3.2 The Analysis Stage

The analysis stage of HABA.DM supports the specification of the abstract organization of the MOMAS. It produces a set of models that represent the *organizational structure*, the *environment*, a *preliminary description of the roles*, and the *interaction protocols* of the MOMAS.

3.2.1 The Environmental Model

A MOMAS organization is not just made of a simple collection of roles, but also of an environmental model. Our methodology produces abstractions of the environment that specify global information resources. We use two communication methods: role interactions and shared information resources. The *environmental model* contains the computational resources of the MAS. It identifies the objects of the environment and their constraints. These constraints are expressed in *Z notation*. (Spivey, 1992).

Figure 3 shows the Ordering System environment. A *Client* is an individual or organization that buys *Product*s from a company. Each client has an identifier id_Client used to place orders. The virtual ShoppingCart stores the list of products that the client has chosen for purchase. Constraints establish that the ids of the object's elements are unique.

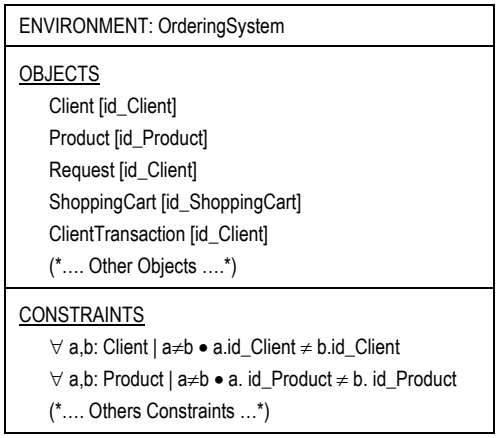

Figure 3: Environmental model: Ordering System.

3.2.2 The Preliminary Role Model

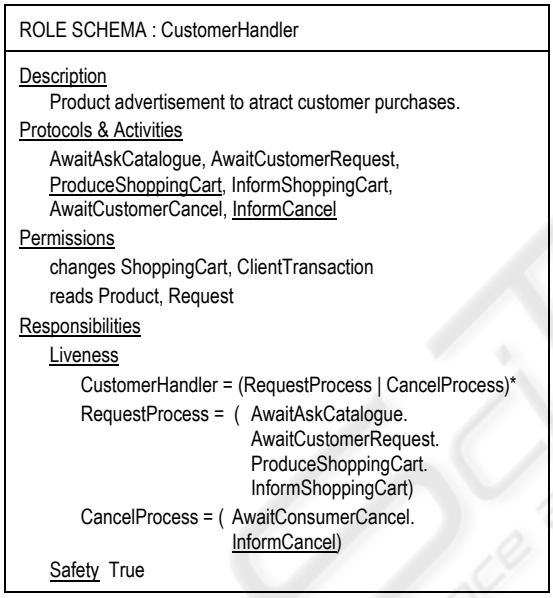

Figure 4: CustomerHandler role preliminary model.

Roles define the behavior of entities within an organization. They also provide a high degree of reusability. During the analysis stage, roles are initially modeled using the notation shown in Figure 4. This initial role model is similar to GAIA's original role model (Wooldridge et al, 2000). Our roles are developed using a top-down approach that refines the roles step by step. Each step adds more detail to the role's specification until the highest level is reached, which corresponds to the role's implementation.

3.2.3 The Interaction Model

Due to the scope of our methodology, we limit the interaction between components to asynchronous controlled ask-reply communications. This is the kind of interaction supported by the programming language HABA.PL.

Figure 5 shows the AwaitCustomerRequest protocol, which belongs to the CustomerHandler role. This protocol is initiated by the Customer role and answered by the CustomerHandler role. Using this protocol, the Customer role informs the Seller role of the order just placed.

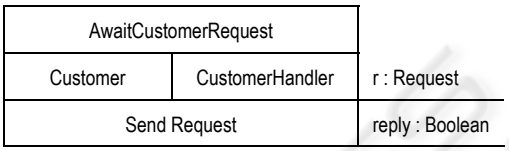

Figure 5: Interaction model: AwaitCustomerRequest protocol.

3.3 The Design Stage

The design stage of HABA.DM defines all the entities that stem from the models specified in the analysis stage. The design models in HABA.DM are closer to the lower-levels of the architecture of the MOMAS.

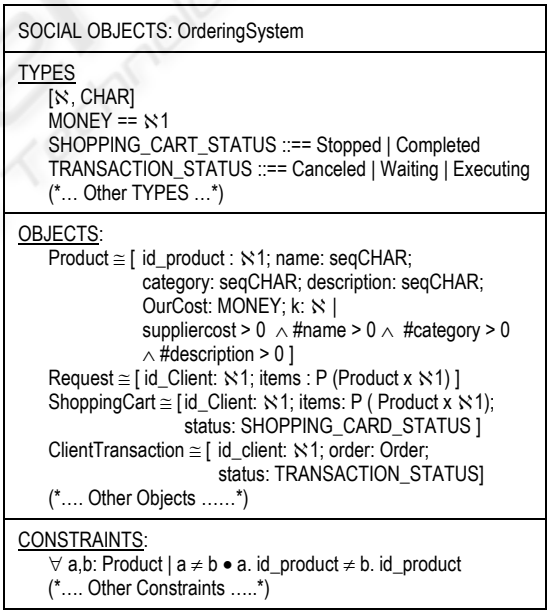

Figure 6: Social object model of our Ordering System.

In HABA.DM, the design stage produces the *social object model*, the *role model*, the *agent model* and the *structural model*. In order to describe these four models we use the formal Z notation.

A *social object model* describes the information resources that belong to the social state of the MOMAS. These resources are types, variables and con-

straints managed by the MOMAS *Management Module* (see Figure 1). Figure 6 shows the notation used to define a social object model.

The social object model of our Ordering System is shown in Figure 6. This model is derived from the environmental model produced by the analysis stage. It describes specification details that are later used during the implementation stage. Figure 10 only shows types, objects and constraints used by the CustomerHandler role.

The *role model* describes each role of the MOMAS in terms of its *services*, *activities*, *permissions* and *responsibilities*. Services and activities define the functionality of the role. Responsibilities can be *liveness* or *safe*. Liveness properties describe the role life-cycle. Safety properties are invariants that must be accomplished during role execution. The model derives directly from the preliminary role model and the interaction model defined in the analysis stage.

Figure 7 shows the role model specification for the CustomerHandler role. This role derives from the initial role model by refining the interaction model. This refinement process translates protocols into services or messages.

ROLE : CustomerHandler

Description
Product advertisement to atract customer purchases.
Services
AwaitAskCatalogue [Customer] : → P(Product)
AwaitCustomerRequest [Customer] : P(Product) → Boolean
AwaitCustomerCancel [Customer] : ℵ1 → Boolean
Activities
ProduceShoppingCart : Request → ShoppingCart
InformCancel : ℵ1 → ClientTransaction
Permissions
changes ShoppingCart, ClientTransaction
reads Product, Request
Responsibilities
Liveness
CustomerHandler = (RequestProcess | CancelProcess)*
RequestProcess = (AwaitAskCatalogue( ).
AwaitCustomerRequest(in r: Request).
ProduceShoppingCart(in r: Request; out s: ShoppingCart).
Seller :: InformShoppingCart(in s: ShoppingCart))
CancelProcess = (AwaitCustomerCancel. InformCancel)
Safety True

Figure 7: CustomerHandler Role model.

As an example of a protocol translated into a service, consider the AwaitCustomerRequest protocol of Figure 5. It is started by the Customer role and served by the CustomerHandler role. In the role model, the AwaitCustomerRequest protocol becomes a service of the CustomerHandler role during the refinement process.

The *agent model* specifies the agent classes used in the MOMAS. Instances of these classes are the agents that the system creates at runtime. This model is the same as GAIA's. Figure 8 shows an example where two agent classes, CustomerAgent and CustomerHandlerAgent, are defined with the roles Customer and CustomerHandler, respectively. The * symbol, next to the left arrow, means that we can create 0 or more instances of the CustomerAgent class. The string 1..10 next to the right arrow denotes that we can create between 1 and 10 instances of the CustomerHandlerAgent class.

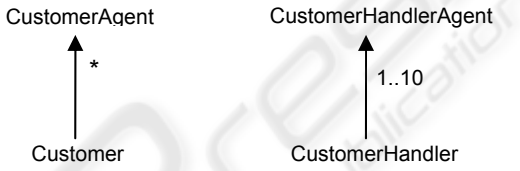

Figure 8: Agent model.

3.4 The Implementation Stage

To illustrate the implementation of a MOMAS we describe how to program the CustomerHandler role using the HABA.PL language of our development framework. We focus on the general structure of the system omitting unnecessary low-level details. We assume that all the types and social objects are well defined in the Social State of the MOMAS. We also assume the following declarations:

```
int n, m, l;
Product P[2000];
ShoopingCart S[1000];
ClientTransaction T[1000];
```

Given these declarations the CustomerHandler role is specified in the HABA.PL programming language as shown in Figure 9.

```
Role CustomerHandler
{
  boolean canceled = false;
  /* Add other attributes here */

  knowledge
  {
    preferred(ID,100):-ID>0,ID<=1000).
    /* Add other knowledge data here */
  }

  InformCancel(int Id_Client):canceled
  {
    int i;
    i=FindShoppingCart(Id_Client);
    j=FindClientTransaction(IdClient)
```

```
    S[i].status=Stopped;
    T[j].status=Cancel;
  }
  /* Add other activities here */
  Service AwaitAskCatalogue
  (Product &Catalogue[2000];int &nmax):
  [Customer]
  {
    nmax = 0 ; int i = 0;
    while (i < n)
    {
      if (P[i].k > 0)
      {
        Catalogue[nmax] = P[i];
        nmax= nmax+1;
      }
      i = i+1;
    }
  }

  Service AwaitCustomerCancel
  (int id_Client) : [Customer]
  {
    canceled = true;
    InformCancel(Id_Client);
  }

  /* Add other services here */
}
```

Figure 9: Implementation of the CustomerHandler role.

Roles are kept in a role library which is part of the social state. Having the CustomerHandler role in the library we can build an agent class *CustomerHandlerAgent* with the role's functions and other functions. Given these definitions we can build an application using the HABA.PM project manager. This project manager supports the construction of a library of reusable roles. Those roles and the agent classes can be programmed within HABA.PM using the HABA.PL programming language. The reader is referred to (Garcés et al., 2006) for a detailed description of both HABA.PL and HABA.PM.

4 DISCUSSION

In this work we present the HABA.DM methodology for the development of MAS. Our methodology is a variant of the GAIA methodology. The goal of HABA.DM is to reduce the gap between the abstract modeling of MAS and their practical implementation. To achieve this goal, we reduce the scope of our methodology to a specific class of MAS that we call Moderately Open Multi-Agent Systems (MOMAS).

We demonstrate how our methodology can be used to model an ordering system for e-commerce. The system is targeted at Business to Consumers, B2C, companies. Our implementation takes advantage of the parallel and distributed capabilities offered by modern information and communication technologies.

Our development methodology is applied incrementally. Roles defined for the system are refined step by step during the three different modelling stages: analysis, design and implementation. This improves on GAIA's approach that does not include the implementation stage. For the implementation of our MOMAS systems we use the HABA.PL programming language and the HABA.PM Project Manager.

REFERENCES

Cernuzzi, L., Juan, T., Sterling, L., and Zambonelli, F. (2004), The Gaia Methodology: Basic Concepts and Extensions. In Methodologies and Software Engineering for Agent Systems. Kluwer.

Ehikiova, S.A. (2001) A Formal perspective to modelling electronic commerce transactions, Colombian Journal of Computation, Vol. 2, No. 2, pp. 21-40.

Garcés, A., Quirós, R., Chover, M. and Camahort, E. (2006) Implementing Moderately Open Multi-Agent Systems. IADIS International Conference WWW / Internet 2007.

Garcés, A., Quirós, R., Chover, M., Huerta, J. and Camahort, E. (2007) A Development Methodology for Moderately Open Multi-Agent Systems. IASTED Conference on Software Engineering SE 2007.

Iglesias, C.; Garito, M.; González, J. and Velaso, J. (1998) Analysis and Design of multi-agent systems using MAS-CommonKADS. Intelligent Agents IV, LNAI vol. 1365, pp. 313-326. Springer Verlag.

Karacapilidis, N. and Moraitis, P. (2001). Inteligent Agents for an Artificial Market System. In AGENTS´01. Fifth International Conference on Autonomous Agents. ACM Press.

Kinny, D., Georgeff, and Rao, A. (1996) A methodology and modelling technique for systems of BDI agents. LNAI vol. 1038, pp.56-71. Springer-Verlag.

Pastor, O., Abrahäo, S., and Fons, J. ((2001). Building e-commerce applications from Object- Oriented Conceptual models. ACM SIGecom Exchanges, vol. 2, issue 2, pp. 28-36.

Spivey, J. M. (1992), The Z notation: A reference manual. Prentice Hall, Second Edition.

Tran, T. (2006). Designing Recommender Systems for e-commerce: an integration approach. 8th International conference on electronic commerce. ACM Press.

Wooldridge M., Jennings, N, and Kinny, D. (2000) The Gaia Methodology for Agent-Oriented Analysis and Design. Autonomous Agents and Multi-Agent Systems vol. 3 no. 3, September, pp. 285-312

Zambonelli, F.;, N. R. Jennings, and Wooldridge, M. (2003) Developing Multi-agent Systems: The Gaia Methodology. In ACM Transactions on Software Engineering Methodology, vol. 12, No.3, pp. 317-370.