AGENT-ORIENTED DESIGN OF E-COMMERCE SYSTEM ARCHITECTURE

S. Faulkner, M. Kolp, A Coyette, T. Tung Do
Information Systems Research Unit, University of Louvain,
1 Place des Doyens, 1348 Louvain-la-Neuve, Belgium
Email: {faulkner, kolp, do, coyette}@isys.ucl.ac.be

Keywords: Agent Systems, Architectural Description Language, Organizational Styles, BDI Agent Model, System Architecture, E-commerce Application.

Abstract: Agent architectures are gaining popularity for building open, distributed, and evolving software required by e-commerce applications. Unfortunately, despite considerable work in software architecture during the last decade, few research efforts have aimed at truly defining patterns and languages for agent architectural design. This paper proposes a modern approach based on organizational structures and architectural description languages to define and specify agent architectures notably in the case of e-commerce system design.

1 INTRODUCTION

The meteoric rise of Internet and World-Wide-Web technologies has created overnight new application areas for enterprise software, including e-commerce applications. These areas demand software that is robust, can operate within a wide range of environments, and can evolve over time to cope with changing requirements. Moreover, such software has to be highly customisable to meet the needs of a wide range of users and sufficiently secure to protect personal data and other assets on behalf of its stakeholders.

Not surprisingly, researchers are looking for new software designs that cope with such requirements. One promising source of ideas for designing such e-commerce software is the area of agent architectures. They appear to be more flexible, modular and robust than traditional including object-oriented ones. They tend to be open and dynamic in the sense they exist in a changing organizational and operational environment where new components can be added, modified or removed at any time.

To cope with the ever-increasing complexity of the design of software architecture, architectural design has received through the last decade increasing attention as an important field of software engineering.

Practitioners have come to realize that getting an architecture right is a critical success factor for system life-cycle and have recognized the value of making explicit architectural descriptions and choices in the development of new software.

To this end, a number of architectural description languages (ADL) [Cle96] and architectural styles [GAO94] have been proposed for representing and analyzing architectural designs. An *architectural description language* provides a concrete syntax for specifying architectural abstractions in a descriptive notation while an *architectural style* constitutes an intellectually manageable abstraction of system structure that describes how system components interact and work together.

Unfortunately, despite this considerable work (see e.g., [SG96)], few research efforts have aimed at truly defining styles and description languages for agent architectural design. To fill this gap, we have defined, in the SKwyRL[1] project, architectural styles for agnt systems based on an organizational perspective [DFK01] and have proposed in [FK01] SKwyRL-ADL, an agent architectural description language. This paper continues and integrates this research: it focuses on an agent perspective for designing and specifying e-commerce software architecture based on organizational styles and SKwyRL-ADL. The structure-in-5 organizational style will be instantiated to design the architecture of the system and the specifications will be expressed in a formal way with SKwyRL-ADL.

The rest of the paper is organized as follows. Section 2 introduces some perspectives of SKwyRL

[1] Socio-Intentional ArChitecture for Knowledge Systems and Requirements Elicitation (http://www.isys.ucl.ac.be/skwyrl/)

insisting on the BDI model, our ADL and organizational styles. Section 3 describes our agent oriented approach on e-commerce system development, including the design of the global architecture with organizational styles, its formal specification with SKwyRL-ADL and the corresponding implementation on an agent-oriented platform. Finally, Section 4 concludes the research.

2 ADL AND STYLES IN SKWYRL

We have detailed in the SKwyRL project an agent ADL called SKwyRL-ADL [FK03] that proposes a set of abstractions that are fundamental to the description and specification of agent architectures based on the BDI (Belief-Desire-Intention) agent model. To help to reader to understand our ADL specification in the rest of the paper, we briefly present the main elements of SKwyRL-ADL including the BDI agent model. SKwyRL-ADL is composed of two sub-models which operate at two different levels of abstraction: *internal* and *global*. The internal model captures the states of an agent and its potential behavior. The global model describes the interaction among agents that compose the agent architecture. We will also introduce organizational styles through the description of one of them, the structure-in-5, that will be used later on in the paper.

2.1 The BDI Agent Model

An *agent* defines a system entity, situated in some environment that is capable of flexible autonomous action in order to meet its design objective [WJ96].

An agent can be useful as a stand-alone entity that delegates particular tasks on behalf of a user. However, in the overwhelming majority of cases, agents exist in an environment that contains other agents. Such environment is a agent system that can be defined as an *organization* composed of autonomous and proactive agents that interact with each other to achieve common or private goals [KGM01].

In order to reason about themselves and act in an autonomous way, agents are usually built on rationale models and reasoning strategies that have roots in various disciplines including artificial intelligence, cognitive science, psychology or philosophy. An exhaustive evaluation of these models would be out of the scope of this paper or even this research work. A simple yet powerful and mature model coming from cognitive science and philosophy that has received a great deal of attention, notably in artificial intelligence, is the Belief-Desire-Intention (BDI) model [Bra87]. This approach has been intensively used to study the design rationale of agents and is proposed as a keystone model in numerous agent-oriented development environments such as JACK [Jac03]. The main concepts of the BDI agent model are in addition to the notion of agent itself we have just explained:

- *Beliefs* that represent the informational state of a BDI agent, that is, what it knows about itself and the world;
- *Desires (or goals)* that are its motivational state, that is, what the agent is trying to achieve;
- *Intentions* that represent the deliberative state of the agent, that is, which plans the agent has chosen for possible execution.

2.2 Internal Model

Figure 1 illustrates the main entities and relationships of the internal model of SKwyRL-ADL. The agent needs knowledge about the environment in order to reach decisions. Knowledge is contained in agents in the form of one of many *knowledge bases*. A Knowledge base consists of a set of *beliefs* that the agent has about the environment and a set of *goals* that it pursues. A belief is a finite set of objects, things with individual *identities* and *properties,* that represents a view of the current environment states of an agent. However, beliefs about the current state of the environment are not always enough to decide what to do. In other words, as well as a current state description, the agent needs some sort of goal information, which describes an environment state that are (not) desirable.

The intentional behavior of an agent is represented by their *capabilities* to react to *events*. An event is generated either by an *action* that modifies beliefs or adds new goals, or by service*s* provided from another agent. Note that these services are represented in the global model because they involve interaction among agents that compose the agent system.

An event may invoke (trigger) one or more *plans*; the agent commits to execute one of them, that is, it becomes intention. A plan defines the sequence of action to be chosen by the agent to accomplish a task or achieve a goal. An action can query or change the beliefs, generate new events or submit new goals.

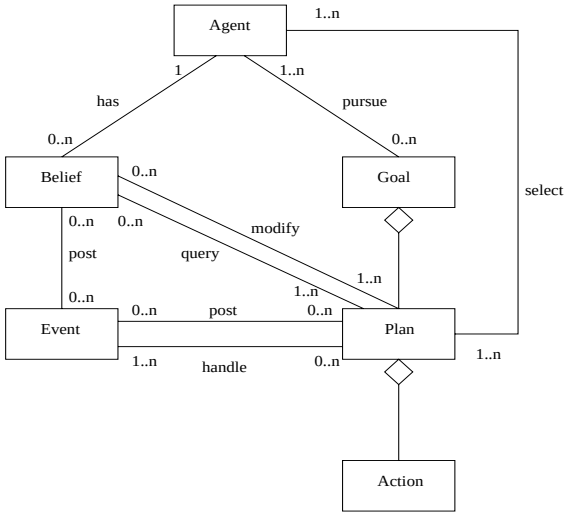

Figure 1: Conceptual Representation of the Internal Model

2.3 Global Model

Figure 2 conceptualizes the global model which describes the interaction among agents that compose the agent system.

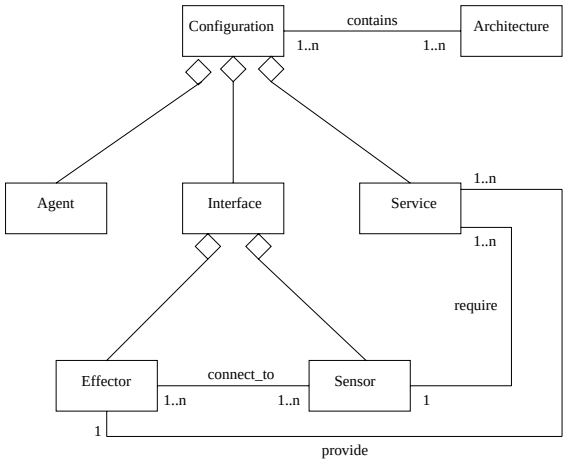

Figure 2: Conceptual Representation of the Global Model

Configurations are the central concept of architectural design, consisting of an interconnected set of *agents*. The topology of a configuration is defined by a set of bindings between *provided* and *required* services.

An agent interacts with its environment through an *interface* composed of *sensors* and *effectors*. An effector provides to the environment a set of servives. Then, a sensor require a set of services from the environment. A service is an action involving an interaction among agents.

The whole agent system is specified with an *architecture* which contains a set of configurations. An architecture represents agents by one or more detailed, lower-level configuration descriptions.

2.4 Agent Architectural Styles

A key aspect to conduct architectural design in SKwyRL is the specification and use of *organizational styles* (see e.g., [KGM01, DFK03]) These are socially-based design alternative inspired from models and concepts from organizational theories that analyze the structure and design of real-world human organizations. These are the structure-in-5, the joint venture, the chain-of-values, the matrix, the takeover, …

For instance, the agent architecture we propose in Figure 3 has been designed following the structure-in-5 organizational style detailed in [DFK03]. In a few words, the structure-in-5 style is a meta-structure that defines an organizational architecture that as an aggregate of five sub-structures, as described by Minztberg [Min92]. It consists of five typical strategic and logistic components found in many organizations. At the base level one finds the *Operational Core* where the basic tasks and operations -- the input, processing, output and direct support procedures associated with running the system -- are carried out. At the top lies the *Apex* composed of strategic executive components. Below it, sit the control/standardization, management components and logistics: *Coordination*, *Middle Agency* and *Support*, respectively. The *Coordination* component carries out the tasks of standardizing the behavior of other components, in addition to applying analytical procedures to help the system adapt to its environment. Components joining the *Apex* to the *Operational Core* make up the *Middle Agency*. The *Support* component assists the *Operational Core* for non-operational services that are outside the basic flow of operational tasks and procedures.

3 AGENT ARCHITECURE FOR E-COMMERCE SYSTEM

E-Media[2] is a typical business-to-consumer application we have developed using the architectural concepts explained in Section 2. The application offers an e-commerce architecture supporting the creation of information sources that facilitate the on-line transaction of products, services, and payments resulting in an effective and efficient interaction among sellers, buyers and intermediaries.

This section describes how we have applied the structure-in-5 style to design the architecture of

[2] http://www.isys.ucl.ac.be/skwyrl/emedia

E-media and used SKwyRL-ADL to formally specify each architectural aspect (belief, goal, plan, action, interface, configuration, service …) of the application.

Based on this architectural specification, we have implemented the application using JACK, a JAVA agent-oriented development environment.

3.1 E-Media Architecture

E-Media includes the following features:

- An on-line web interface allows customers to examine the items in the *E-Media* catalogue, and place orders;
- Customers can search the on-line store by either browsing the catalogue or querying the item database. An online search engine allows customers to search title, author/artist and description fields trough keywords or full-text search;
- If an item is not available in the catalogue, the customer has the option to order it
- Internet communications are supported
- On-line financial transactions including credit card and anonymity are protected;
- All web information (e.g., product and customer turnover, sales average, …) of strategic importance is recorded for monthly or on-demand statistical analysis.
- Based of this statistical and strategic information, the system permanently manages and adapts the stock, pricing and promotions policy. For example, for each product, the system can decide to increase or decrease stocks or profit margins. It can also adapt the customer on-line interface with new product promotions.

Figure 3 models the architecture of E-Media using the *i** model [Yu95] following the structure-in-5 organizational style we have introduced in Section 2 . *i** is a graph, where each node represents an *actor* (or system component) and each link between two actors indicates that one actor depends on the other for some goal to be attained. A dependency describes an "agreement" (called *dependum*) between two actors: the *depender* and the *dependee*. The *depender* is the depending actor, and the *dependee,* the actor who is depended upon. The type of the dependency describes the nature of the agreement. *Goal* dependencies represent delegation of responsibility for fulfilling a goal; *softgoal* dependencies are similar to goal dependencies, but their fulfilment cannot be defined precisely; *task* dependencies are used in situations where the dependee is required.

As show in Figure 3, actors are represented as circles; dependums – goals, softgoals, tasks and resources – are respectively represented as ovals, clouds, hexagons and rectangles; dependencies have the form *depender* → *dependum* → *dependee*.

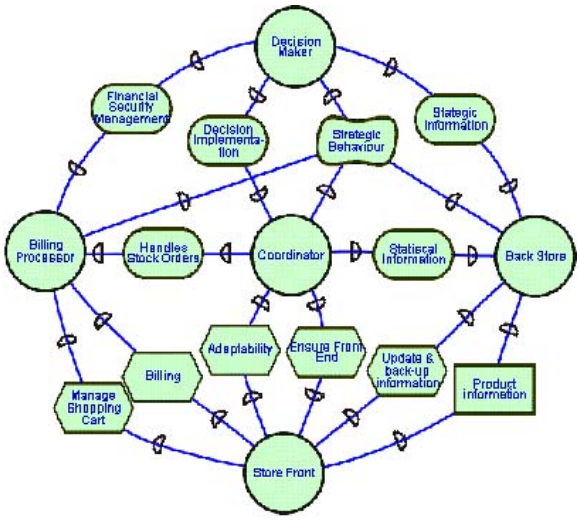

Figure3: The E-Media Architecture in Structure-in-5

The *Store Front* plays the role of the structure-in-5's *Operational Core*. It interacts with customers and provides them with a usable front-end web application for consulting, searching and shopping media items.

The Back Store constitutes the structure-in-5's *Support* component. It manages the product database and communicates to the *Store Front* relevant product information. It stores and backs up all web information about customers, products and sales to be able to produce statistical information (e.g., analyses, average charts and turnover reports). Such kind of information is computed either for a predefined product (when the *Coordinator* asks it) or on a monthly basis for every product. Based on this monthly statistical information, it provides also the *Decision Maker* with strategic information (e.g., sales increase or decrease, performance charts, best sales, sales prevision, …).

The *Billing Processor* plays the role of the structure-in-5's Technostructure in handling customer orders and bills. To this end, it provides the customer with on-line shopping cart capabilities. It also ensures the secure management of financial transactions for the *Decision Maker*. Finally, it handles, under the responsibility of the *Coordinator* component, stock orders to avoid shortages or congestions.

As the structure-in-5's Middle Agency, the *Coordinator* assumes the central position of the architecture. It is responsible to implements strategic decisions for the Decision Maker (Strategic Apex). It supervises and coordinates the activities of the Billing Processor (initiating the stock and pricing policy), the Front Store (adapting the front end interface with new promotions and recommendations) and the Back Store

(parametrizing statistical computing) ensuring that the system fulfills its mission in an effective way.

Finally, the Decision Maker assumes the *Strategic Apex* role of the structure-in-5. It defines the *Strategic Behavior* (e.g., sales and turnover, product visibility, hits, …) of the system ensuring that objectives and responsibilities delegated to the *Billing Processor*, *Coordinator* and *Back Store* are consistent with respect to their capabilities.

3.2 E-Media Formal Specification

The architecture described in Figure 3 gives an organizational representation of the system-to-be including relevant actors and their respective goals, tasks and resource inter-dependencies. This model can serve as a basis to understand and discuss the assignment of system functionalities but it is not adequate to provide a precise specification of the system details. As introduced in Section 2, SKwyRL-ADL provides a finite set of formal agent-oriented constructors that allow to detail in a formal and consistent way the software architecture as well as its agent components and their behaviors.

Figure 4 shows a high-level formal description of the *Back-Store* agent. Three aspects of this agent component are of concern here: the *interface* representing the interactions in which the agent will participate, the *knowledge base* defining the agent knowledge capacity and the *capabilities* defining agent behaviors.

```
Agent:{Back-Store
    Interface
      Effector[provide(strategic_info)]
      Effector[provide(statistical_info)]
      Effector[provide(product_info)]
      Effector[provide(back-up)]
      Sensor[require(stategic_behavior)]
    KnowledgeBase:
        Product_KB          Statistical_KB
        BS_System_KB        BS_Customer_KB
    Capabilities:
        Statistical_CP      Strategic_CP
        Data_Management_CP
          }
```

Figure 4 : Agent Structure Description of the Back-Store

SkwyRL-ADL allows to work at different levels of architectural abstractions (i.e., different views of the system architecture) to encapsulate different components of the system in independent hierarchical descriptions. For instance, in Figure 4 the *Back Store* agent has three knowledge bases (KB) and three capabilities (CP), but the description level chosen here does not specify the details of the beliefs composing the KB or the plans and events composing each capability.

The rest of the section focuses on the *Back Store* agent to give an example of a refinement specification with our ADL for each of the three aspects of the agent: interface, KB and capabilities.

Interface. The agent *interface* consists of a number of effectors and sensors for the agent. Each of them represents an action in which the agent will participate. Each effector provides a service that is available to other agents, and each sensor requires a service provided by another agent. The correspondence between a required and a provided service defines an interaction. For example, the Back Store provides the *statistical_info* service that the Coordinator requires.

Such interface definition points two aspects of an agent. Firstly, it indicates the expectations the agent has about the agents with which it interacts. Secondly, it reveals that the interaction relationships are a central issue of the architectural description. Such relationships are not only part of the specification of the agent behavior but reflect the potential patterns of communication that characterize the ways the system reason about itself.

The specification of a service is given in Figure 5. Each provided or required service is detailed by specifying the *sender* agent that initiates the service, the set of *receiver* agents that interact with the sender, the *reply-with* and *content* statements that define the information about which the service expresses an interaction and optionally a set of *parameters* to define the information required to execute the service. Like the *parameters* the *reply-with* information is represented with a belief or a set of terms (e.g., function, constant or variable).

```
Service: {Ask(statistical_info)
  sender:  Coordinator
  parameters:  (tw: TimeWindows), (id: Id_product)
  reply_with:  to: Turnover ∨ sl: Sales
  receiver:  Back-Store
Effect:   Add(Statistical_KB,
                Achieve(statistic("today","on_product")
        }
```

Figure 5: A Service Specification

Knowledge Bases. A *knowledge base* (KB) is specified with a name, a body and a type. The name identifies the KB whenever an agent wants to query or modify them (add or remove a belief). The body represents a set of beliefs in the manner of a relational database schema. It describes the beliefs the agent may have in terms of fields. When the agent acquires a new belief, values for each of its fields are specified and the belief is added to the appropriate KB as a new tuple. The *KB type* describes the kind of formal knowledge used by the agent. A *Closed world* assumes that the agent is operating in a world where every tuple it can express is included in a KB at all times as being true or false.

Inversely, in an *open world* KB, any tuple not included as true or false is assumed to be unknown.

Back-Store has four KBs: the *Product_KB*, the *Customer_KB*, the *Statistical_KB* and the *System_KB*. Figure 6 specifies the Product_KB:

```
KnowledgeBase: {Product_KB
  Kb_body:
    product(Id_Prod,Title,Class,Description,Price)
    purchase(Id_Card,Id_Prod,Date,Quantity,Pay_Means)
    audio(Id_Prod,Artist(+),Compositor(+))
    book(Id_Prod,ISBN,ISNN,Author(+),Publisher)
    dvd(Id_Prod,Actor(+),Realizator(+))
    software(Id_Prod,Version,Editor(+))
    stock(Id_Prod,Availability,Date_levering)
    newproduct(Id_Prod,InsertDate)
  kb-type: closed_world
                }
```

Figure 6: A Knowledge Base Specification

For instance, the predicate *purchase* with four arguments represents the customer and product ids, the quantity and the means of payment. The '+' symbol means that the attribute is multi-valued.

Capabilities formalize the behavioral elements of an agent. It is composed of plans and events that together define the agent's abilities. It can also be composed of sub-capabilities that can be combined to provide complex behavior.

Figure 7 shows the *Data_Management* and the *Statistical* capabilities of the *Back-Store* agent. The body contains the plans the capability can execute and the events it can post to be handled by other plans or can send to other agents. For example, the *Data_Management* capability is composed of four plans: *Send_Product_Data* is used to send product information to the Store-Front, *Backup_Database* backs up all information about customers, products and sales in order to provide statistical information, *Interface_Custom_Id* allows user registration and *Check_Identification* checks the customer login/password when an order is made.

```
Capability: {Data_Management_CP
  CP_body:
    Plan Send_Product_Data
    Plan Backup_Database
    Plan Check_Identification
    Plan I nterface_Custom_Id
    endEvent update_interface
    PostEvent backup
              }
Capability: {Statistic_CP
  CP_body:
    Plan Prov_Turnover_On_Demand
    Plan Prov_Turnover
    Plan Sales_Average
              }
```

Figure 7: Capabilities Specification

A plan defines the sequence of actions and/or services (i.e., actions that involve interaction with other agents) the agent selects to accomplish a task or achieve a goal. A plan consists of:

- an *invocation condition* detailing the circumstances, in terms of beliefs or goals, that cause the plan to be triggered;
- an *optional context* that defines the preconditions of the plan, i.e., what must be believed by the agent for a plan to be selected for execution;
- the *plan body*, that specifies either the sequence of formulae that the agent needs to perform, a formula being either an action or a service to be executed;
- an *end state* that defines the post-conditions under which the plan succeeds;
- and optionally a set of services or actions that specify what happens when a *plan fails* or *succeeds*.

Figure 8 specifies the *Prov_TurnOver_On _demand* plan that gives the turnover on every product.

```
Plan:{Prov_TurnOver_On_demand
Invocation: Achieve(stat_Comput("today","on_produc »))
Context:
    ¬newproduct(id,"today-15d")
    ∧ ¬is_done("prov_turnover_on_demand",Today)
    ∧ ¬day(now ="25")
Body:
  action: compute_turnover(tw, id)   as  to: Turnover
  //with  id: Id_product From
                Coordinator.Ask(statistical_info).parameter/
  //with   tw:TimeWindows From
                Coordinator.Ask(statistical_info).parameter/
  effect:  Add(Statistical_Kb, product_turnover(tw, id,to))
  service: {Tell (statistical_info)
        sender: Back-Store     content: to: Turnover
        receiver: Coordinator
  effect:  Add(System_KB,
              is_done("prov turnover_on_demand",Today))
          ∧  Add(Statistical_Kb,
                    stat_comput("today","on_product", id))
  Endstate:
          Add(Statistical_Kb,
                    stat_comput("today", "on_product", id))
Suceed:
    action: compute_sales(tw, id)   as  sl: Turnover
    //with tw:TimeWindows From
                Coordinator.Ask(statistical_info).content/
    effect: Add(Statistical_Kb,  product_sales(tw, id, to))
    service: {Tell(statistical_info)
      sender: Back-Store       content: sl: Sales
      receiver: Coordinator
Fail:
    action:  search_set(prov_turnover(),id,tw1))
                                        as  set_of to: Turnover
    // with tw1 ∈ [today-12mth;today] /
    action: extrapol_turnover(set_of to)   as  to1: Turnover
    service: {Tell(statistical_info)
      sender: Back-Store       content: to1: Turnover
      receiver: Coordinator
    effect:  Add(Statistical_Kb,
                    stat_comput("today","on_product",id)
            }
```

Figure 8: A Plan Specification

The invocation condition is defined with the goal *Achieve* meaning that the today's computation of product statistics must be true in the current or in

some future state. The agent then determines the applicability of the plan by analysing its context.

The context states that the product must be on sale for at leat 15 days and that the current day is not the 25th. of the month. The plan could not have been executed today.

The body is composed of a sequence of an action and a service. The Back-Store first computes the product turnover from data (*id* and *tw*) transmitted during the execution of the *Ask(statistical_info)* service (see Figure 3). The result (*to*) is then sent to the Coordinator. The *service effect* results in satisfying the goal event.

In case of failure, the Back-Store searches in its KBs the turnovers that have been previously computed during the last twelve months in order to extrapolate a current value. The extrapolation is sent to the Coordinator. With respect to the case of a successful plan execution, the service effect satisfies the goal event but it does not prevent the re-execution of the plan during the same day.

Configuration To describe the complete topology of the system architecture, the agents of an architectural description are combined into a SkwyRL *configuration*.

```
Configuration E_Media
    Agent DecisionMaker   Agent Coordinator
    Agent BackStore       Agent StoreFront
    ...
    Service Ask(strategic_info)
    Service Tell(strategic_info)
    Service Import(product_info)
    Service Export(product_info)
    Service Ask(statistical_info)
    Service Tell(statistical_info)
    Service Achieve(back-up)
    Service Do(back-up)
    ...
Instances
    DM: DecisionMaker          CO: Coordinator
    BS: BackStore              ST: StoreFront
    Askstrat: Ask(strategic_info)
    Tellstrat: Tell(strategic_info)
    Importprod: Import(product_info)
    Exportprod: Export(product_info)
    Askstat: Ask(statistical_ifo)
    Tellstat: Tell(statistical_info)
    Achievebkup: Achieve(back-up)
    Dobkup: Do(back-up)
    ...
Collaborations
    DM. Askstrat  --- Tellstrat.BS;
    ST.Exportprod --- Importprod.BS;
    ST. Achievebkup --- Dobkup.BS;
    CO.Askstat --- Tellstat. BS;
    ...
End E_Media.
```

Figure 9: The E-Media configuration

Instances of each agent or service that appear in the configuration must be identified with an explicit and unique name.

The configuration also describes the collaborations (i.e., which agent participates in which interaction) through a one-to-many mapping between provided and required service instances.

Part of the E-Media configuration with instance declarations and collaborations is given in Figure 9.

To allow dynamic reconfiguration and architecture evolvability at run-time, configurations separate the description of composite structures from the description of the elements that form those compositions. This permits to reason about the composition as a whole and to reconfigure it without having to examine each component of the system.

3.3 E-Media Implementation

Based on the structure-in-5 architecture described in Section 3.1 and the formal SKwyRL-ADL architectural specification overviewed in Section 3.2, the E-Media application has been implemented with JACK [Jac03], a BDI agent-oriented development environment for JAVA. We briefly describe the E-Media implementation to illustrate the role of the agents and their interactions.

Figure 10: E-media Main Interface

When an on-line customer gets connected to E-media, an instance of the Front-Store is created and displays the interface depicted in Figure 10. It allows the new coming user to register (1). The Back-Store handles the information provided by the user and checks its validity (2). If the access is granted, the user can purchase products on E-Media by adding catalogue items to the shopping cart (4) managed by the Billing-Processor. At any time the user can use the navigation-bar (3) to switch from one section of the website to another. Promotions (5) best sales (6) are part of the strategic behaviour objective. The promotions policy is initiated by the Decision-Maker based on the strategic information provided by the Back-Store. The Coordinator chooses the best promotions and adapts

consequently the Store Front layout. The Coordinator acts similarly for the best sales: the Back-Store computes the five best sellers and the Coordinator updates accordingly the Store-Front.

Figure 11: Interface of E-Media DVD Section

Figure 11 describes the Store-Front interface for the DVD section, i.e., when the "DVD" button of the navigation-bar is activated. To search E-Media DVD catalogue, the user must fill at least one field of the search engine (1). The Store-Front sends the query parameters to the Back Store which provides the results back to the Store-Front (2). At any moment during the session, the user can click on a product (best seller, query result, shopping cart…), a request is then sent to Back Store to provide more information on this product (3). When the user activates the billing process, the Billing-Processor displays the items in the shopping cart and computes the total and sub-total for each product. It then checks the validity of the user's credit card number. Once the payment is accepted, the Billing-Processor informs the Store-Front. A confirmation message is displayed and the shopping cart is reset.

4 CONCLUSION

Nowadays, software engineering for new enterprise application domains such as eBusiness is forced to build up open systems able to cope with distributed, heterogeneous, and dynamic information issues. Most of these software systems exist in a changing organizational and operational environment where new components can be added, modified or removed at any time. For these reasons and more, Agent architectures are gaining popularity in that they do allow dynamic and evolving structures which can change at run-time.

Architectural design has received considerable attention for the past decade which has resulted in a collection of well-understood architectural styles and formal architectural description languages. Unfortunately, these works have focused object-oriented rather than agent-oriented systems. This paper has described an approach based on organizational styles and an agent architectural description language we have defined to design agent architectures in the context of e-commerce system engineering. The paper has proposed a validation of the framework: it has been applied to develop E-Media, an e-commerce platform implemented on the JACK agent development environment.

REFERENCES

[Bra87] M . E. Bratman. Intention, Plans and Practical Reason. *Harvard University Press*, 1987.

[Cle96] P. C. Clements. A Survey of Architecture Description Languages. In *Proc. of the Eighth International Workshop on Software Specification and Design*, Paderborn, Germany, March 1996.

[DFK02] T. T. Do, S. Faulkner and M. Kolp. Organizational Multi-Agent Architectures for Information Systems. in *Proc. of the 5th Int. Conf. on Enterprise Information Systems* (ICEIS 2003), Angers, France, April 2003.

[FK03] S. Faulkner and M. Kolp. Towards an Agent Architectural Description Language for Information Systems. In *Proc. of the 5th Int. Conf. on Enterprise Information Systems* (ICEIS 03), Angers, France, April 2003.

[GAO94] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting Style in Architectural Design Environments. In *Proc. of SIGSOFT'94: Foundations of Software Engineering*, New Orleans, Louisiana, USA, Dec. 1994.

[Jac03] JACK Intelligent Agents. http://www.agent-software.com/.

[KGM01] M. Kolp, P. Giorgini, and J. Mylopoulos. An Organizational Perspective on Multi-agent Architectures. In *Proc. of the 8th Int. Workshop on Agent Theories, architectures, and languages*, ATAL'01, Seattle, USA, Aug. 2001.

[Min92] H. Mintzberg. *Structure in fives: designing effective organizations*. Prentice-Hall, 1992.

[SG96] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*, Prentice Hall, 1996.

[Yu95] E. Yu. Modeling *Strategic Relationships for Process Reengineering*, Ph.D. thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[WJ96] M. Wooldridge and N.R Jennings, editors. Special Issue on Intelligent Agents and Multi-Agent Systems. *Applied Artificial Intelligence Journal*. Vol. 9(4), 1996.