A Knowledge Modeling Tool for Rule – Based Agents

Marco Repetto, Christian Vecchiola, Antonio Boccalatte
LIDO Lab – Department of Communication, Computer and System Sciences (DIST)
University of Genova
Via Opera Pia 13, 16145, Genova - Italy
{kapsule, nino}@dist.unige.it

Abstract

Different approaches to improve business process have been proposed. One of the most common techniques is based on software agents and workflow technology. A software agent can be adopted to implement and to manage workflow, and to give to an enterprise new functionalities such as job automation and integration among levels.

This paper describes Agent Developer Studio, a tool for modeling and implementing rule-based agents database-oriented. The particular context addressed in this work is the workflow management, and a special focus is kept for agents acting in an environment mainly based on a Relational Database Management System (RDBMS).

The system architecture, the underlying model, and the possible fields of applications of such a tool, will be discussed. An example of a particular application will be also presented.

1. Introduction

The main requirements that a modern manufacturing information system should satisfy are [1]: enterprise integration, distributed organization, interoperability, open and dynamic structure, cooperation, human integration with software and hardware, agility, scalability and fault tolerance. In order to deal with the need for rapid, continuous change, computer science is challenged to develop new interrelated information and communication technologies, able to satisfy the social needs of co-operating user groups, as well as the management requirements of formal organizations [2]. Workflow systems are among the most advertised technologies addressing this trend. Workflow can be defined as the *computerized facilitation or automation of a business process in whole or part*, and the workflow management system as *a system that completely defines, manages, and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic* [3]. Due to their ability to integrate different information systems, for instance plant and business information, workflow management is one of the major applications of software agents in manufacturing.

Technology based on software agents, can be used to implement and to manage workflow, and to give to enterprises new functionality such as jobs automation and levels integration. Software agent, a new paradigm of Artificial Intelligence (AI), can be defined as *a computational system that is situated in a specific environment, and that is capable of autonomous actions in this environment in order to meet its design objectives* [4].

There are two basic ideas of agent-based architecture applied to workflow management [5-6]. The first is from the ADEPT (Advanced Decision Environment for Process Tasks) project by British Telecom labs [7]. The software agents take full responsibility for business process provisioning, enactment, and compensation, where each agent is managing and controlling a given task or a set of tasks. In the second agent-based architecture, called agent-enhanced workflow, agents provide an additional layer to an existing commercial workflow management system. The agent layer takes full responsibility for both provisioning and compensation phases of business process [5].

Conventional programming languages are designed and optimized for the procedural manipulation of data but they are not well suited to solve complex problems where humans usually use abstract and symbolic approach. Although abstract information can be modeled using conventional programming languages, a considerable programming effort is required to turn information into a format suitable for a procedural programming paradigm. One of the results in the area of AI has been the development of techniques that allows the modeling of information at higher levels of abstraction. These techniques are embedded in languages or tools, which allow to build programs resembling human logic. These programs, which emulate human expertise in well-defined problem domains, are called expert systems.

In particular, CLIPS (C Language Integrated Production System) [8] is language to generate expert systems, developed by the Software Technology Branch (STB) in 1984-'86, and it is designed to make easier the development of software to model human knowledge and expertise. CLIPS supports different programming models and has been designed for full integration with other languages such as C and ADA. Procedural code can be defined as external functions and called from CLIPS: in this way, all the tasks needed in order to define a workflow are performed. Traditional features of expert system, such as problem solving and modeling complex scenarios, are potential advantages if applied to workflow implementation.

The followings sections will illustrate how the features of CLIPS can be used to implement agent-based architectures applied to workflow management.

2. Agent Developer Studio

Agent Developer Studio (ADS) is a tool made to develop agents graphically and to implement them as expert systems. Agents are made-up of blocks that are automatically translated into rules: these constitute the knowledge-base of the agent. ADS provides also a platform to run the generated expert systems, the CLIPS enhanced Shell.

ADS consists of:

- a Graphical User Interface (GUI) for designing, editing, configuring and verifying a knowledge model
- enhanced CLIPS shell, improved with new functionalities (e.g., RDBMS query, office automation features)

Current implementation is based on the C++ language, ADO (Active Data Objects) and MAPI (Mail Application Program Interface) libraries.

CLIPS source code is available in C language, so can be executed on different platforms (Windows, Mac, Unix).

ADS agents interact with RDBMS and their knowledge is modeled in order to operate on tables, to access to database and to evaluate stored data and decisions taken according to its knowledge-model.

For this reason the CLIPS shell and also the GUI must communicate with the RDBMS: for the GUI it is necessary to know the schema of tables, for the shell the system needs to access to the data stored into the tables. The Figure 1 shows how the different parts of the system interact: the arrows represent how the communication among the entities takes place. The GUI deals with design and modeling, and the shell with execution. The GUI looks for schema on the RDBMS, produces files that will be run on the shell, and this one interacts with the database when the agent runs.

In the ADS environment a design session begins with a schema definition: a knowledge model schema is a diagram in which the blocks are atomic elements necessary for WFMS agents; then the blocks are connected to assert logic dependencies among them.

The user can set block parameters in order to configure every agent's rule: input data-types, data constraints and actions the agent performs. With these operations, human reasoning is directly mapped to CLIPS behavior, without forcing the user to write the CLIPS code.

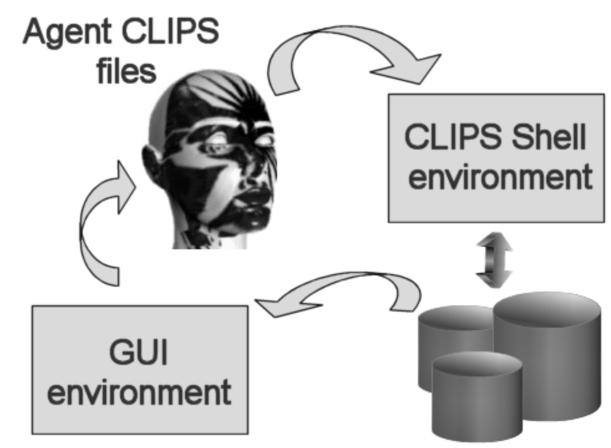

Figure 1. System architecture

Each block receives from the previous block, that is, the parent, all the data types plus its output type (if the parent is a select query block). This is needed because a select operation imports data in the CLIPS environment.

The next step is to compile the schema in order to obtain a translation into CLIPS language. After checking schema and data correctness, ADS performs the translation and put its results in four text files, which will be loaded into CLIPS environment for initializing expert system knowledge-base.

3. Formal Model

In order to illustrate the model, it is necessary to distinguish two environments: the GUI and the CLIPS shell. The goal of the model is to maintain a strong link between these two environments and to allow an easy mapping of objects defined. In this way the process of translation from the agent schema to the expert system that implements the agent, becomes an automatic task.

The model deals with data representation, knowledge-base representation and the process of translation.

3.1 Data representation

In ADS environment, data can have different meanings: records into tables and messages in a queue. These elements are all mapped into facts in the expert system that implements the agent. For this reason in the GUI space is provided a generic class called `CFactModel` mapping a fact in the Shell environment. The inheritance mechanism allows a specialization of the fact definition and it allows building some particular data-types. The actual implementation of the model provides data-types called `CRowModel` that represents a row on a specific table in a RDBMS.

In the CLIPS system data are represented with *deftemplates* construct that are record types. Deftemplates have fields that represent the properties of data, and constraint on these fields can be established in order to perform pattern matching of the rules.

Each data-type in the GUI has a corresponding deftemplate in the shell space, so the mapping is 1:1 and the translation from one space to another can be automatic and simple.

3.2 Knowledge-base representation

A knowledge base is a set of rules describing relations between elements in the domain of knowledge [9]. A knowledge base is represented in the GUI space through blocks and connection between them. Each block is mapped to a rule in the shell environment.

Blocks are the fundamental elements in the GUI. A block can be considered as a unit of process that has some input data, it changes the status of the system according to particular constraints on data, and, depending on the nature of the block, it produces some output.

This structure allows direct rule-based programming, because blocks directly map to rules. Rules are used to represent heuristics, or "rules of thumb," which specify a set of actions to be performed for a given situation. A rule is composed of an *IF* clause and a *THEN* clause. The *IF* clause of a rule represents a series of patterns which specify the facts (or data), causing the rule to be applicable. The *THEN* clause of a rule represents the set of actions to be executed when the rule is applicable.

Constraint on input data of the block are directly mapped to the pattern of the rule associated to that block, and actions in the *THEN* clause are a representation of the actions that change the status of the system and produce some output. Also, this process can be performed automatically.

3.3 Translation process

The translation phase puts in the shell the CLIPS files that implement the agent schema defined in the GUI context. This activity can be divided into three steps:

- definition of the deftemplates corresponding to the data types in the GUI space
- definition of the rules associated to each block in the schema following the order starting from the first block
- building of the file in which data types and rule are loaded into CLIPS shell

The second step is the most time-consuming, because for each rule several tasks have to be performed. First of all there is a check for consistency of constraints on input data of the block, then constraints are translated into pattern. These two operations build the *IF* clause of the rule. The sequence of action are characteristic of each block and actually, the build of the *THEN* clause involves some standard operations performed in order to allow the expert system to evolve in the proper way. The sequence of actions working on input data, producing some output data can be various and the translation of this part is left to each block.

3.4 Execution flow

Agents are executed in the CLIPS shell by the inference engine, which automatically matches facts against patterns and determines which rules are applicable. Actions of applicable rules are executed when the inference engine is instructed to begin execution. The inference engine selects a rule and then the actions of the selected rule are executed (which may affect the list of applicable rules by adding or removing facts). The inference engine then selects another rule and executes its actions. This process continues until there are no applicable rules.

Agent behavior is defined as a sequence of actions performed according to conditions defined on data. It is necessary to guarantee that a sequence of actions that processes a particular instance of data is executed in the right order and it is completed, before the agent starts to process other instances.

CLIPS provides an environment in which the execution is driven by the pattern matching on facts. For this reason some rules can be fired in parallel mode if they have the same pattern. There is the problem of maintaining the right sequence of actions performed by the agent. This aspect has been solved with the insertion of a hidden field that stores the status of execution in each instance of data type in the shell. This field is properly updated by the rules, and this task is the result of the “standard operations” addressed before. A particular rule provides a starting point for the execution and that rule is the one associated to the start block that must necessarily be included in our schema.

4. Order Manager : a practical Example

A simple example will be used to cover all the aspects of the process, starting from the design of the agent’s schema, its realization and the files produced.

The example is an order manager that looks for new orders stored in a database, it checks for each order if it can be changed into an accepted order, evaluating the content of the order and RDBMS status. For each accepted order, the agent sends an order notification to the customer and updates the tables, moving the order from the original table to another. If an order can be satisfied, the agent sends a notification to the customer.

The structure of the database is composed of four tables: INCOMINGORDERS, STORES, CUSTOMERS, ACCEPTEDORDERS:

- **INCOMINGORDERS**

  ORDERCODE integer, primary key

PRODUCTCODE integer
CUSTOMERCODE integer
QUANTITY integer

- **STORES**
  PRODUCTCODE integer
  QUANTITY integer
- **CUSTOMERS**
  CUSTOMERCODE integer, primary key
  NAME varchar
  SURNAME varchar
  EMAIL varchar
  COMPANY varchar
- **ACCEPTEDORDERS**
  ORDERCODE integer, primary key
  PRODUCTCODE integer
  CUSTOMERCODE integer
  QUANTITY integer

This database has been built using SQL Server 7.0 and Access 2000 in order to test the system on different DBMS.

4.1 Project set-up

The first step to do to make an agent schema is to open a new project in which the project name and the project folder have to be defined. ADS system then defines four CLIPS files that constitute the expert system's source code that will be run in the CLIPS shell. These files are:

- *projectname*_Def.clp : stores data definition of objects handled by the agent
- *projectname*_Rule.clp : stores the knowledge-base of the agent in the form of rules
- *projectname*_Main.clp : this file loads in the CLIPS shell the previous files and implements an infinite loop that allow the agent to stay alive
- *projectname*_Pack.clp : stores the code of all the previous files in the order in which they appear (this is not necessary to the execution of the agent but can be useful having all the code in one single file)

It is necessary to define connection parameters, such as database source, with user login information if requested. Then the application automatically retrieves data from database tables. The agent schema is defined dragging and dropping blocks from the blocks toolbar.

4.2 Schema design

Every agent needs a starting point so the ADS application provides a *start-block* that performs a select-query on a specified table. The agent has to look for new orders so the INCOMINGORDERS table is the one that has to be queried. The *start-block* performs a *select* without constraints, but this is not a problem because in this example all the orders have to be handled.

To check if an order can be satisfied a *conditional block* is placed on the schema. The condition implemented by this block is Boolean: the block performs a select query on a specified table with filters, and if the query returns at least one row, the condition is satisfied. Once the block is positioned on the schema, it is necessary to connect it to the previous block with a wire. This operation must be performed for each block added. After connection has been established the properties of each block have to be set starting from the *start-block* and setting first the input parameters, then the preconditions and finally the action. Preconditions are a filter on data that the agent already knows because they work on input parameters, action is used to perform some operation or to instantiate new data in the environment optionally using some filters. For the conditional block particular preconditions don't have to be set because all the output of the start block has to be checked. The constraints of the select-query established in the action parameters are used to check if an incoming order can be satisfied. This is an example of the query performed when the agent is running.

```
SELECT * FROM CUSTOMER
     WHERE
     PRODUCTCODE =
      <!INCOMINGORDERS#PRODUCTCODE!>
     AND
     QUANTITY >=
     <!INCOMINGORDERS#QUANTITY!>
```

tags in the form <*!tablename#fieldname!*> mean that those particular texts are placeholder for values caught in the input parameters. After the *conditional block*, the data flow is split into two branches, one for orders that can be satisfied and one for others that cannot be satisfied.

In the first case are necessary five blocks to implement the following operations:

- insert the order into EXECUTIVEORDERS table (*insert-block*)
- delete the order from INCOMINGORDERS table (*delete-block*)
- update the STORES table with new value of quantity for the ordered product (*update-block)*
- select customer data from CUSTOMERS table (*select-block*)
- send a e-mail to the selected customer to notify him (*mail-block*)

As presented in Figure 2, the path followed by the orders that do not satisfy the condition, is made-up only with the last two previous blocks, because a notification of the order status is sent to the customer.

4.3 CLIPS code generation and Agent running

When the schema is completed, the code generation can be performed. After some preliminary operations of topologic and logic checking of the schema, the CLIPS files addressed before are generated. The files are ready to be loaded in the shell following this order:

*projectname*_Def.clp
*projectname*_Rule.clp

The CLIPS shell is a single process that runs till the exit command, and the main loop activates the inference engine of CLIPS periodically when no rule fires anymore. In this way, the agent continuously queries to the database in order to satisfy new orders, and when one of them can be promoted to a accepted order updates tables, and waits for new orders.

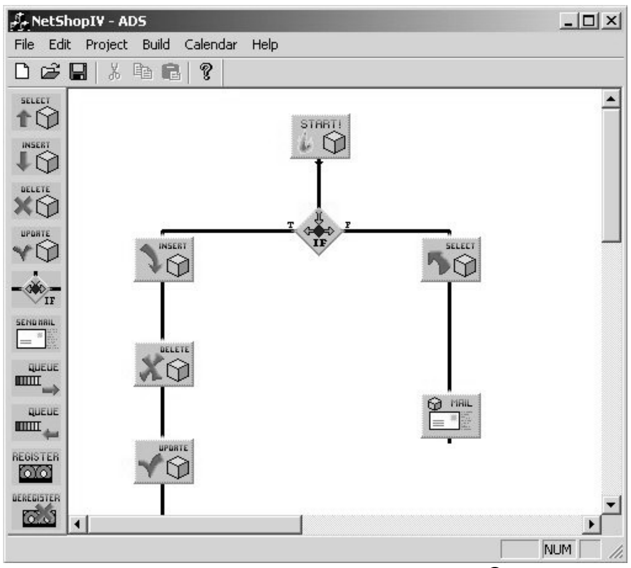

Figure 2. A knowledge schema in ADS graphic user interface

Using some settings of the CLIPS shell the agent's execution flow can be monitored during the running, selecting whether to watch facts, rule-firings or both. These options are also a check to verify that the agent's behavior is correctly implemented.

5. Analysis

The model and the application discussed is only a small part of the ongoing project. In fact there are other built-in functionalities that can be added to the GUI, to the shell and, in minimum part, to the model.

Different start blocks can be provided in order to instantiate different data-types in the agent environment, also different condition blocks can be provided in order to perform more complex checks. The impact of these add-ons does not involve refactoring, because the model remain still the same or there are at least slight changes. A new change instead is the blocks' behavior and the rule generated but new data-types and new behaviors are obtained via sub-classing.

Variation-points are features in the system that can be implemented with different strategies to provide the same functionality. In this case variation points are related to the different strategies used to represent data in the CLIPS shell. Using CLIPS Object Oriented Language (COOL) probably would provide a more efficient knowledge-base model, but the automatic translation would become more complex. Another aspect that has to be taken into account is that the translation process actually checks for the schema's consistence and puts into rules blocks adding some hidden stuff to maintain the right sequence of firing. To have a more elaborate and efficient behavior, the algorithm that implements this process can be changed without affecting the overall structure of the model. This is an important aspect and could be a working point in the following releases.

6. Conclusions

The development of a Multi Agent System (MAS) requests not only skills in the particular problem to solve, but it is necessary to know techniques and tools to build it. So there are two areas, leading to two different professional roles: the expert of the problem, and the software developer. The main goal of the discussed approach is to make possible this decoupling of roles, but it is also necessary to guarantee all the requested support to the workflow manager in realizing his project. These two features have to be provided hiding the agent's implementation.

The approach followed has some vantages: the CLIPS shell is available on different platforms; work time is reduced because the agent's code generation follows immediately the design phase; the possibility to add new features with a low impact on framework is practically infinite; the model used is easy to understand and extendable.

Although expert systems are efficient strategies to solve problems where there is no imperative model, they often need good skills to use. For this reason an interface able to model the agent's knowledge without paying attention to expert system features, but focusing only on the agent's behavior can be a powerful tool.

References

[1] Shen W., Norrie D. H.. Agent Based System for Intelligent Manufacturing: A State of the Art Survey. Knowledge and Information Systems 1999; 129:156

[2] Schäl T.. Workflow Management Systems for Process Organization. Lecture Notes in Computer Science, Vol. 1096. Springer-Verlag, Berlin Heidelberg New York, (1996).

[3] Workflow Management Coalition, "*The Workflow Reference Model*", Available at http://www.wmfc.org/wh2002.htm, 1995.

[4] Wooldridge, M.: Intelligent Agents. Multiagent Systems, G. Weiss editor. The MIT Press, April 1999, 27-77.

[5] Odgers, B.R., Thompson, S.G., Shepherdson, J.W., Cui, Z., Judge, D.W., O'Brien, P.D.: Technologies for Intelligent Workflows: Experiences and Lessons. Proceedings of Agent-Based Systems in the Business Context, AAAI 1999 Workshop, 19 July 1999.

[6] O'Brien, P.D., Wiegand, M.E.: Agent based process management: applying agents to workflow. Knowledge Engineering Review, Vol. 13:2, September 1998.

[7] Jennings, N.R., Faratin, P., Norman, T. J., O'Brien, P., Wiegand, M.E., Voudoris, C., Alty, J.L., Miah, T., Mamdani, E.H.: ADEPT: Managing Business Processes using Intelligent Agents. Proc. DCS Expert Systems 96 conference (ISIP Track), Cambridge, UK 5-23, 1996.

[8] Riley, G.: CLIPS: A Tool for Building Expert Systems. Available at http://www.ghg.net/clips/CLIPS.html (1999).

[9] M D, Fisher, "Expert systems and anthropological analysis" BICA Issue nr. 4, March 1986