Specification Environment for Multi-agent Systems Based on Anonymous Communications in the CIM Context

A. ATTOUI, A. HASBANI, A. MAOUCHE
Laboratoire d'Informatique/ ISIMA
B.P. 125
63173 AUBIERE CEDEX, FRANCE
Tel: 73 40 74 40 Fax: 73 26 88 29
mail: ammar@sp.isima.fr

Abstract
In this paper, we propose a formal specification method for manufacturing systems software development. Our approach is based on rewriting logic and multi-agent paradigm. It proposes a methodology for the analysis and the structuration of the command part of manufacturing systems, in terms of cooperative and specialized agents. Rewriting logic constitutes the formal framework. The approach may be seen under two complementary aspects; the first one consists, in general, in the distributed system conception and the second concerns the use of IAD principles for the representation, the distribution and the knowledge co-operation through a system of cognitive, autonomous and co-operating agents. The method supports modularity and abstraction, follows the great principles of a multi-agent systems approach and supports real time applications.

Keywords
Specification, Validation, Formal methods, Manufacturing Systems, Multi-Agent Systems, , Methodology.

1 INTRODUCTION

It is widely recognized that methodologies and tools used in analysis requirements and specification stages determine directly the quality of the development of software systems. For real-time systems, these methodologies and tools must additionally provide concepts, formalisms, and mechanisms to express at a high level of abstraction the concurrency of multiple computation threads, the synchronization and the communication between these threads and the handling of internal and external events. Inconsistency and incompleteness of the specifications must be detected as soon as possible in order to avoid costly readjustments in the design and development stages. Manufacturing systems belong to the more general class of distributed and real time systems. The intelligent systems designers (Ayel, 1991]) are not out of the way of this evolution; They use both systems with blackboard architecture (Laasri,1989), (Bouzouane,1993), where the contribution to solve a problem goes by a common data structure, and Multi-agent systems, where communications are done solely by message passing. We can distinguish two distribution levels: running distribution (parallel inferences) and data distribution (knowledge) which favours the performance improvement (sequential or parallel execution) (Occello, 1993). Implementation of such systems requires two different stages. The first stage consists in designing a conceptual unambiguous model for specifying the requirements analysis. The second stage consists in deriving an executable model for evaluation and validation of the system. The information provided by this model are very useful for the final definition of the control part of the system.

Numerous methods are available to specify and design such systems. Among these methods we can mention SA/RT, SA/DT, OMT(Hatley,1990), (Fayad,1994). These methods are adequate to capture and describe the behavior of complex systems. But most of them do not offer a formalized framework facilitating the elimination of any inconsistency and incompleteness. Indeed, these methods use multiple different and incompatible models with the same specification. For example, SA/RT uses data flow diagrams, control flow diagrams (finite state machines) and a process activation table. Timing constraints are specified separately in the timing constraints table. In the other hand, Petri nets and their extensions (Bruno,1986), (Peterson,1981) can be efficient when the studied system is not too complex.

Formal methods and techniques have been suggested over the last several years to prove properties about specifications. CCS, Z, VDM, ESTELLE, LOTOS, SDL (ISO,1989), (Courtiat,1991), (Sijelmassi,1991), (Binding,1991), (Busttard,1992), (IS8807,1988), (Vigder,1991), (Coelho,1992), (Hoare,1985), (Lightfoot,1991), are the most prominent ones. In a general way, these formal methods have not been widely used in industrial software development environment for several reasons (Fraser,1994). Among these reasons, we can mention: a formal specification language provides a notation (syntactic domain), a univers of objects (semantic domain), and a precise rule defining which satisfy each specification. This makes them an inappropriate tool for communicating with the end user and requires that software engineers, designers, and implementors master the notation and the conceptual grammar of the language

In this paper, we present an approach based on a unique formalism: the rewriting logic. This approach is dedicated to support the distributed system design in general, and in particular, to the design of multi-agent systems (SIC,1992). In this domain, our contribution is to palliate to some difficulties (of communication, concurrence, and real-times) proper to the cognitive approach by message passing (Bouront,1993). This approach, has not yet been formalized as blackboard approach, nevertheless, it seems avoid some limits of the blackboard approach. Its main features are the following:

- It takes into account an important variety of systems based on a sequential or a concurrent execution model.
- It uses the object model to implement multi-agent systems as in (Cardozo,1993), (Stinckwich,1993).
- It supports modularity and abstraction.
- It offers the possibility for validating the specifications and for generating the code automatically, according to a predefined distributed architecture.

This approach does not necessitate the mastering of the concepts and the foundations of the rewriting logic. It also integrates architectural considerations and is able to support the design of complex systems.

2 FORMAL MODEL FOR MULTI-AGENT SYSTEMS

Our model is based on a formalism that is closed to the one of the MAUDE language (Meseguer,1990). It takes into account the complexity of the information processed within an enterprise, it includes a description of the various entities manipulated, the actions that these entities may undergo, the temporal constraints and the tracability of information. The model is independent of the target language.

An agent will be associated to a module representing both the cognitive body and the resolution strategy. It is made up of rewriting rules basis expressing production rules, facts basis and a communication interface.

Rules and fact basis encapsulate the agent knowledge and transcribe its resolution strategy in regard with its specification. The communication interface allows, to a basic agent, to share its results with others agents of the same abstraction level .

2.1 The formal modules (Description of an agent)

A formal module is a quadruplet (Σ, a, R, S). Σ is the set of symbols for the functions of the module. It permits the formal description of the static part of an agent. Symbols can be simple (i.e. characters) or complex syntactical units. a is the set of structural axioms necessary to achieve the rewritings in a concurrent manner modulo the axioms.
The doublet (Σ, a) is called the signature of the agent. R is a set of rewriting rules. It permits the formal description of the dynamical part of a system. A rule has the form [t] ==> [t'] where t and t' are terms constructed from Σ. The notation [t] is used to indicate that t represents an element of the class of terms modulo the axioms. We will use three structural axioms called ACI : associativity, commutativity and identity. A rule indicates that the current state of the agent corresponding to the configuration t, becomes a new state corresponding by the configuration t'.
A configuration is defined in terms of agents and messages. It is represented through a sentence in the language of the corresponding formal module. More precisely, a configuration at a given time is composed of:

- identified agents. Each agent possesses intrinsic properties and is in a particular defined state.
- messages. Messages are generated by external excitations or internal interruptions. A same message can be sent to different agents.

During the specification stage, each agent can be considered if necessary as composed of other agents. A hierarchical decomposition is thus naturally introduced. An agent can use for its specification other agents. We will use the notions of level and visibility. An agent A is at a level immediately lower of the one of the agent B, if B is directly used in the composition of A. Usually, agent A knows only attributes of agents of its level. In some situations it is necessary that an agent see attributes of agents of lower levels. Such attributes are called visible.

2.2 The signature

For (Σ,a) we use a general formulation (figure 1) which is closed to the one proposed by (Meseguer,1990). This general form of signature integrates different operators. Op < _: _ / _> is the constructor of agents. Op _ = _ permits to affect a value to an attribute. Op _, _ is the syntactical constructor of attribute lists. These lists are used for the designation of the attributes of an agent. Op _ _ is necessary for the construction of distributed configurations. It permits the specification of any configuration. A configuration is composed of agents and messages. This operator has been declared modulo the ACI axioms. Thus, the order with which agents and messages are declared has no influence on the reduction process used further.

```
                    /*Alphabet of the system*/
Type  Agent, Attribute, Attributes, Msg, Configuration, Value, AgentId, ClassId,
                              AttributeId;
        /*Hierarchies and structural relations between the agents*/
              Subtype AgentId, ClassId, AttributeId < Value;
                     Subtype Attribute < Attributes;
                   Subtype Agent, Msg < Configuration;
   /*Operators for constructing the words and the sentences of a formal module*/
                Op < _: _ / _> : AgentId Value -> Object;
                Op _ = _ : AttributeId Value -> Attribute;
     Op _, _ : Attributes Attributes -> Attributes [ Assoc, Com, Id = Nul];
 Op _ _ : Configuration Configuration -> Configuration [Assoc, Com, Id = Nul].
```

Figure 1 The general signature of a formal module

The description of a real multi-agent system consists to instantiate the metatypes of $\sum$ (agent, Attributes, Msg, ...).

2.3 The rewriting rules

Actions of messages on an agent are described through the rules. An agent can receive messages from agents in the same level or in the upper levels of its hierarchy. An agent can send messages to agents in the same level or in the lower levels of its hierarchy. A message can also be intercepted through a rule and routed to any level. A rule signals the occurrence of a communication in which n messages and n agents are involved. All the agents participating in a rule are at the same level. The general form of a rule is given by figure 2.

```
                    /*Syntax*/
    M1M2..Mp<AG1: C1/ listeAt1>...<AGi: Ci/listAti>
         <Aj: Cj/ listAtj>...<Ak: Ck/ listAtk> ==>
           <Aj: Cj/ listAtj>...<Ak: Ck/ listAtk>
 <AGm: Cm/ listAtm>...<AGn: Cn/ listAtn>Mq ... Mr [T]
```

Figure 2 Syntax and effects of a rewriting rule

Effects:
The messages M1M2..Mp are deleted after the execution of the rule.
The states of the agents Aj,..., Ak are modified.
Agents A1,..., Ai which appear only in the left part of the rule, are deleted.
New Agents AGm,...,AGn defined in the right part, are created.
New messages Mq, ..., Mr are created.
[T] is a temporal constraint. It can take:
1: every (T, msg) : to each time interval T, send the message msg;
2: within (T, msg) : after the time T, send the message msg;
3: AT (T, msg) : at the time T, send the message msg;
4:before(T, msg) : before the time T is elapsed, send the message msg.

In this syntax, an agent is represented by the term <idAgent:C/list At>. idAgent is the agent identifier. C is the agent class. listAt is a list of conditions on the attributes of the agent. Attributes which are modified must be visible at this level. At least, T expresses the service time for the execution of the rule. T can be a random law.
This general form permits to specify, at a high level of abstraction, the different conditions of co-operation and synchronisation between agents.

A rule indicates that the system goes from the configuration defined by the left part to a new configuration defined by the right part. A rule can be activated when all the messages of the left part are present and when all the conditions on the attributes of the left part are satisfied. Rules describe the actions of the events associated to the messages. They permit to reason on the state changes of the system and to draw valid conclusions about its evolution. They constitute the formal description of the dynamic aspects of the system.

2.4 Synchronous rules and asynchronous rules

A rule is synchronous if several agents are simultaneously modified through an atomic action. Otherwise it is asynchronous.

An example of synchronous rule is given by the following rule which specifies the carriage of a part by a an AGV (Automated Guided Vehicul) from a turning unit to a control unit:

(Carry by V Part P from T to C) < T: TurningUnit / CurrentOperation: finish>
< C: ControlUnit/ (queue : N) <= 10 > <V: AGV/ state: free>
==>
< T:TurningUnit / CurrentOperation: O> < C: ControlUnit/ queue : N+1 >
<V: AGV/ state: S>

In this rule, the carriage of P from T to C is an atomic operation. The trigger of this rule is:

(Carry Part P from T to C by V) and (T.currentoperation= finish) and (C.queue <=10)

The same problem can be formulated through an asynchronous rule:

(Carry Part P from T to C by V) ==>(TurningUnit T exit Part P)
(AGV V accept part P) (ControlUnit C accept Part P)

This rule deletes the message (Carry Part P from T to C by V) from the configuration and produces three new messages. The rules which intercept these messages can then fire in parallel.

Thus, formulation of a rule in an asynchronous way expresses possibilities of parallel processing.

<u>Internal working rules</u>

These rules have the following form:

Ai ==> Ai, Am...An, Mq...Mr[T]

They permit to express that state changes result from an internal working not visible at this level. They generally concern an agent which can evolve according to a self working. Through the parameter [T], which can be any random law it is possible to express random behaviour.

These rules permit specifications very similar to those proposed by (AGHA,1987) for actors.

3 DEVELOPMENT METHODOLOGY FOR MULTI-AGENT SYSTEMS

The method covers almost the entire life cycle of a distributed system in general, and of a multi-agent system in particular.

Specifications use concepts and notion of concurrent rewriting logic which leads to a coherent description of the system. The automatisation of the passage from one phase to the next permits to kept the coherence and to ensure a uniform life cycle.

3.1 Analysis

This step concerns the definition of agents which will compose the future command part of the studied manufacturing system according to needs and objectives.
The automatic passage from this phase to the specification phase remains very difficult. Consequently (ever though our specification method lightly overlaps the analysis phase), our methodology does not integrate a method of analysis which is already intrinsic to it. But it use may prove efficient, if it is situated after an analysis of needs made with E/R model, SADT or the diagram of data flows.

3.2 Specification methodology

The method is based on a systemic approach for the analysis and the decomposition of the studied systems. Indeed, we are interested in complex reactive systems which can be decomposed in three sub-systems:

- **the physical and logistic sub-system**. This is the part of the studied system composed of physical resources. According to the type of the studied system, these resources can be engines, machines, hardware systems, software systems, etc.
- **the decisional or monitoring sub-system**. It is the set of rules or functions, when applied to the physical sub-system, permits to reach the fixed goals: the decisions, the regulation, etc.
- **the information sub-system**. Its main feature is to establish the connection between the
  two other sub-systems. It intercept the data flows from the physical sub-system, if necessary processes them, and sends the information to the decisional sub-system (figure 3).

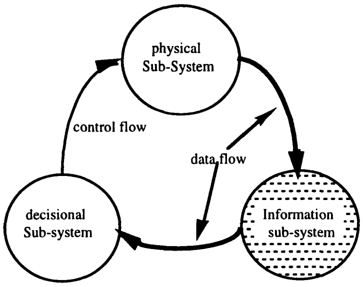


Figure 3 Complex system decomposition in basic sub-systems

The analysis and the design of a complex system can be done with respect of the two following different specification stages which are strongly coupled.

First stage: identification of the physical sub-system resources
Five steps are used:

1- Identification of the physical sub-system objects or resources. It consists to highlight the different components of the studied system. This identification concerns the available objects if the studied system is an existing system, or the resources or objects necessary to build a new system. In this step, the Entity Association model or any other simple formalism can be used for the analysis of the physical part or the static part of the system.

2- For each identified object or resource, precise its interface (cooperation and communication protocol):

- Input data flow messages,
- Input control flow signals or events,
- Output data flow messages,
- Output control flow signals or events,

3- For each object, identify, if they exist, the state variables or visible attributes. These attributes are necessary to write the control or the decisional rules. In general, these state variables are used in the system global synchronisation and monitoring rules (machine state: On, Off, Occupied, etc.).

4- For each resource decide if it is an active resource type or a passive resource type. Active resources, generally, concern an object which can evolve according to a self working (their states result from an internal working not visible at this level.) and realize one or several tasks in accordance with the received control command (robot, workstation, a software active program (server), etc.). These reactive objects interact with their environment by messages exchanges. They are complex systems as well as the studied system.

Passive resources are objects which perform a particular task, but they have not an internal logic which allows them to evolve or to perform actions in an autonomous manner (pallet, machining tools, sensor, captor, database, etc.). The distinction between these two kind of objects is of a nature to facilitate the analysis and the specification of the decisional sub-system in the following stage. At this step, it is important to have an accurate vision of the nature and the type of each component of the sub-system, because, this will determine the structuration of the decisional sub-system as it will be shown in the following stage.

5- For some cases passive resources are data storage means. They are also components of the information sub-system.

second stage: Hierarchical decomposition by level abstraction of the decisional sub-system

This stage uses, also, five steps.

1- associate a ***monitoring agent*** for each active resource. The interface of this agent will be made of input and output control and data flow of its resource (machine command and utilization protocol, software system invocation interface, etc.). The visible attributes or state variables values must be integrated in the agent interface as input or output messages. The agent is the only entity qualified to retrieve or to give the contents of these kind of attributes (encapsulation principle) to the other agents of the same or the upper levels.

2- For each passive resource or object necessary for the implementation of the decisional sub-system, associate an access ***manager agent***. Its interface have to integrate the resource access protocol massages (Database access protocol, captor or sensor access commands, etc.).

3- Identify the input:output control flows of the decisional sub-system.

4- For each input control flow (signal) or input data flow (message) of the studied system, associate an interception rewriting rule ("Handler"). This rule may implicate several monitoring agents and manager agents (synchronous rewriting rules). It can also use visible state variables of the physical sub-system via its associated agents. If an interception rule of a given event or signal have to use a complex logic in plus of the simple synchronisation and the control of the implicated monitoring agents and manager agents, it's advised to associate to this signal or event a decisional functional object which have to be decomposed in next level of the hierarchical decomposition process of the decisional sub-system. This abstract object is called ***expert agent.*** Indeed, to process such events to take a decision, a complex logic must be used. This logic can use some expertise in a particular domain (scheduling algorithms, production planing, etc.). The expert agents use their specific and private data and passive resources. Make these resources or data visible at this level is of a the nature to compromise the readiness and the comprehension of the decision logic of this level.

5- For each expert agent highlighted in the four step, precise:

- the knowledge on its environment in order to complete its interface and, above all, to identify its resources during its decomposition process.
- its expertise.

Then, apply to it this second stage of the method: hierarchical decomposition by level abstraction, only, if this agent has to be created.

third stage: top-down decomposition of the physical sub-system

This stage presents an interest only if some passive or active physical resources have to be defined. In this case, each resource of this kind identified in the first stage, must be considered as a new system to be studied and we apply to it the three stages of the method. For more details, see Attoui(95a)

3.4 Inter-agent Communications

One of the most important factors in the inter-agents communication protocols is the designation of agents implicated in this kind of interaction. To which agent a message will be sent? from which agent a message will be received?

With the variety of messages which transit between an application parallel entities and the diversity of sources, it becomes important and possible to have automatic methods of information filtering. An agent needs a fraction of messages. To have access to these messages without knowing, beforehand, their sources is not easy to make use for the designers of multi-agents system.

In this context, we have defined and made use of a mechanism based on anonymous communication by message passing. No designation of agents implicated in the communication is necessary; whatever it may be, explicit or implicit, direct or indirect (Attoui,1994b). In order to enssure communication according to message content and requested services, we have used a filtering process of messages emitted by anonymous agents. All messages are filtered and oriented towards their addressee.

The filtering is a process integrated to anonymous communication mechanism. The information filtering is strongly linked to information retrieval because they have a common objective: retrieve an information requested by an agent.

Nevertheless, there is a main difference. Information filtering is applied to an incoming data-flow whereas information retrieving is applied on an existing database which may evolve in time. In multi-agents system, the data flow is constituted of messages produced by parallel entities.

In our environment, the inter-agent communication is modelled by a channel. A channel is an abstraction of a physical communication network, so it provides a communication path between several agents. It may be seen as a communication "software bus". At the time of channel declaration, we permit only its later utilisation by the agents. The agents identities and the direction of transfers are not indicated. A channel does not have associated data types. This reduces the number of channels required if data of different types are transmitted between agents.

Figure 4 highlights the logical architecture of a multi-agent application. Each agent is bound to a channel to carry out inter-agents communication. A channel is seen by agents as a logically shared variable. A nested agent Aij can use a communication channel Cij. Two agents situated on different levels of the nested structure can use any common communication channel. For instance, Ax may communicate with Aik (k<>j) by using Ci channel. Cp channel can also be used. This principle applied is identical to the use of variables by nested procedures. Then the scope of the channel for agents is the same as the scope of variables for procedures in the classical programming languages.

Anonymous communication mechanism allows solving problems of the agents designation and the messages description. Rewriting rules enssure one of "languages acts" characteristics namely the correspondence between messages and actions to do.

In the specification level of multi-agents system behaviour, messages specified in the left part of rules represent, in reality, the profiles of these messages. No designation, explicit or implicit, of agents is specified in these profiles. Any message respecting a profile will be received by an agent independently of the sender. A complete anonymity on sender identity and a total transparency on its localisation in the communication network are ensured by the anonymous communication mechanism.
The anonymity is also applied in emission. Indeed, the right part of a rule specifies the actions to execute when conditions described in the left part are satisfied. These actions include the sending of messages.

With this anonymous communication mechanism, interrogation, request and supply messages may be exchanged without any designation of the concerned agents. More details on this macanism can be found in (Attoui,1994b), (Maouche,1994)

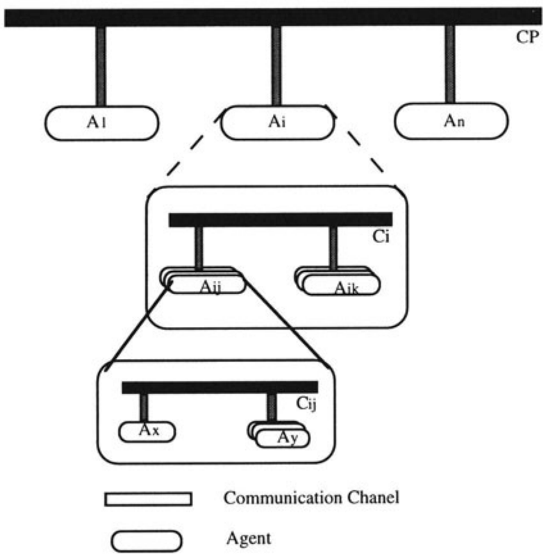


Figure 4 Hierarchical multi-agent architecture

4 Verification and validation

The inference engine of VALID environment allows the verification of the specification previously constructed with the graphical editor.

The user has to submit to the reduction engine the system description using objects and inference rules. Each test session is based on the notion of "scenario". A scenario is a foreseeable evolution of the system behavior from an initial state to a final state. (a state is defined by attribute values of the instanciated objects that constitute the real system).

The user specifies the state in which must be the system either at the beginning of the scenario (initial state) and at the end of the scenario (final state). If it is possible he can also specify several undesirable states of the system for this scenario.

The inference engine has the relevant information to perform a syntactical reduction on the specified objects. This reduction will respect the principle of locality and visible levels between objects. We remind that a message sent by an object can be intercepted by its father, its sons or its brothers in the specified hierarchy.

We can make either a global verification of the hole specified system or a partial incremental verification by respective levels. So we have to define for the engine the object that we consider as "root" (beginning object) and the scenario (initial and final state) associated to this object. This verification can detect:

- deadlock situations : the reduction process reaches an intermediate configuration not foreseen by the user and can no more evolve.
- undesirable boundless cycles : the system reduces in cyclical manner the same set of rules; so the engine reaches the maximal inference number specified by the user.
- impossibility to reach the final state indicated by the user.
- undesirable state (if specified)

This automatic detection is performed in an interactive way with the user. Each time a problem is detected, the system can show the historic of the behavior from the initial state. This historic contains the list of the explored configurations an the rules triggered since the beginning of the verification and the possible errors messages. The step by step inference mode is also available, it is very useful for a detailed supervision of a complex system reduction.

The temporal constraints in the simulation process, and especially the integration of the various temporal constraints primitives defined by the VALID syntax, produce time references in the historic file.

5 APPLICATION TO A CONCRETE INDUSTRIAL WELDING LINE

As an illustration of our approach, we consider the example of a welding line for electrical motors (figure 5). The configuration which is considered corresponds to the one of a real line installed by a French automobile manufacturer (Kellert,1990).

There is four similar welding stations for welding the motors and two types of conveyors for accessing it: a main conveyor to permit the routing of the motors along the line and several bi-directional conveyors to permit access to the welding stations (one for each station). Different workings of the line can be considered and several policies can be studied. The objective is to maximise the putting through of the line. We only give the description of a policy called “ordered policy”.

Unwelded motors must access to one of the four stations to be welded. They circulate on the main conveyor and they can take the elevator and the bi-directional conveyor leading to a station if there is space available in the queue of this station; otherwise they continue their routing on the main conveyor.

At the output of a unit, a welded motor must go down the bi-directional conveyor in order to output the line through the main conveyor. In any case, there can be only one part in an hashed section (a bi-directional conveyor and the two elevators situated at each of its extremities). An untreated motor can reach the end of the line. In this case it is recycled at the line input through the recycling conveyor. Figure 5 shows the schema of this line.

The system can be described through three levels (figure6). An agent type called "Unit" has been introduced to characterise one section of the line. A section is composed of a welding station and the corresponding means for accessing it: a section of the main conveyor and the critical section composed of the bi-directional conveyor and the elevators at its extremities.

Each level of rules represents the part of the decisional subsystem dealing with this level. This approach permits a hierarchical decomposition of the decisional subsystem as well as the physical subsystem. The locality and the encapsulation principles are naturally respected.

For instance, rule "R0" of the Welding Line level is used to routing parts from the last unit "Unit4" to the terminal Unit because the decomposition leads to a structure of the terminal unit which is different from that of the other unit. On the other hand, the rule "R3" of the Unit level is used to propagate the internal message to the upper level.

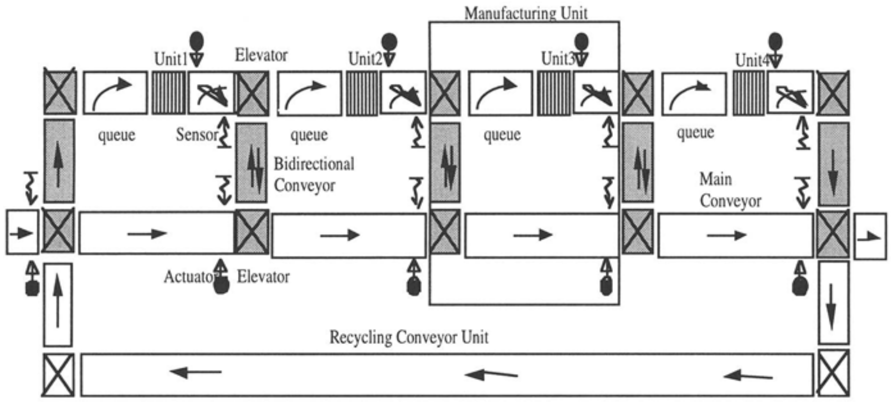
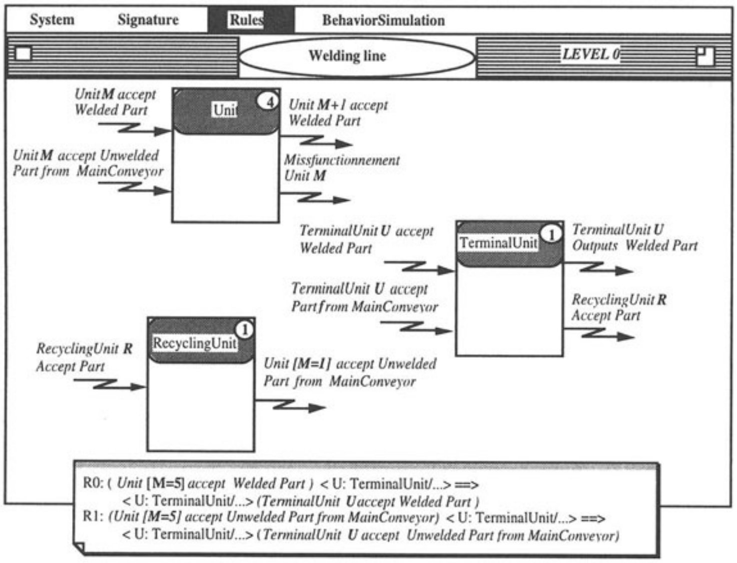


Figure 5 Schema of the Welding Line

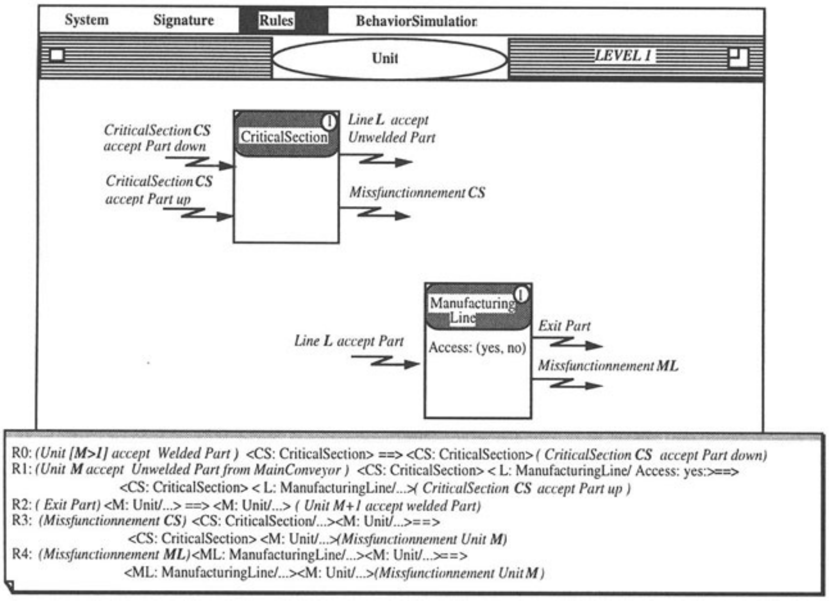
It is important to note that when a free variable [I] is used in a message, this variable must be instanciated to all the possible values of the corresponding parameter. For instance, (Elevator [I] is free) represents both (Elevator E1 is free) and (Elevator E2 is free) messages.

A set of abort messages (Missfunctionning CS, Missfunctionning ML, ...) are used at each level. They allow agents to perform recovery operations which are the most critical operations in this kind of real time systems. Temporal constraints ([t1: (Missfunctionning CS)]) can be used to express that the execution of rules must not exceed the given deadline value, otherwise, the following message is generated. Finally, rules R0 and R1 of the Unit level specify the ordered policy mentioned above.

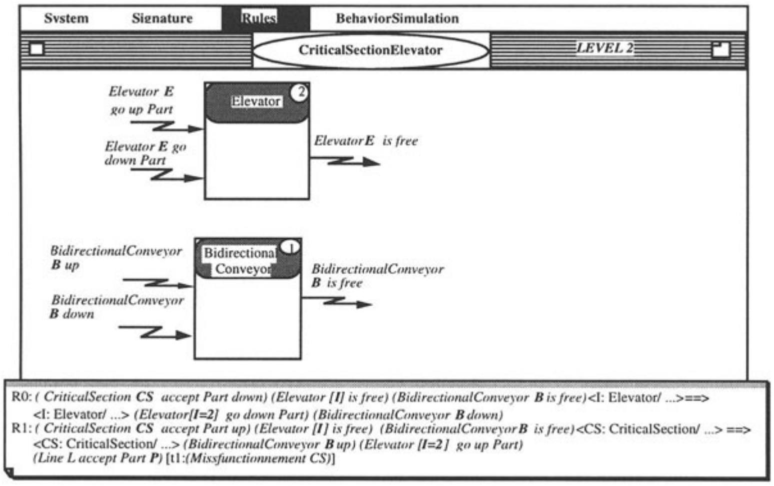
The figure 7 give a VALID session for the description of the first level of this system.


System
Signature
Rules
BehaviorSimulation
Welding line
LEVEL 0
Unit
4
Unit M accept Welded Part
Unit M accept Unwelded Part from MainConveyor
Unit M+1 accept Welded Part
Missfunctionnement Unit M
TerminalUnit
1
TerminalUnit U accept Welded Part
TerminalUnit U accept Part from MainConveyor
TerminalUnit U Outputs Welded Part
RecyclingUnit R Accept Part
RecyclingUnit
1
RecyclingUnit R Accept Part
Unit [M=1] accept Unwelded Part from MainConveyor

```
R0: ( Unit [M=5] accept  Welded Part ) < U: TerminalUnit/...> ==>
        < U: TerminalUnit/...> (TerminalUnit  U accept Welded Part )
R1: (Unit [M=5] accept Unwelded Part from MainConveyor) < U: TerminalUnit/...> ==>
        < U: TerminalUnit/...> (TerminalUnit  U accept  Unwelded Part from MainConveyor)
```

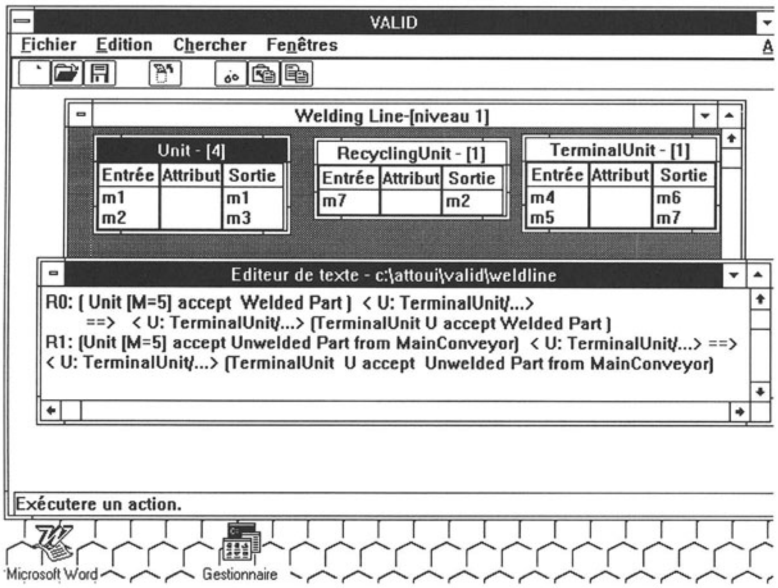


```
R0: (Unit [M>1] accept  Welded Part )  <CS: CriticalSection> ==> <CS: CriticalSection> ( CriticalSection CS  accept Part down)
R1: (Unit M accept  Unwelded Part from MainConveyor )  <CS: CriticalSection> < L: ManufacturingLine/ Access: yes:>==>
                  <CS: CriticalSection> < L: ManufacturingLine/...>( CriticalSection  CS  accept Part up )
R2: ( Exit Part) <M: Unit/...> ==> <M: Unit/...> ( Unit M+1 accept welded Part)
R3:  (Missfunctionnement CS) <CS: CriticalSection/...><M: Unit/...>==>
                   <CS: CriticalSection> <M: Unit/...>(Missfunctionnement Unit M)
R4:  (Missfunctionnement ML)<ML: ManufacturingLine/...><M: Unit/...>==>
                   <ML: ManufacturingLine/...><M: Unit/...>(Missfunctionnement Unit M )
```



R0: (*CriticalSection* ***CS*** *accept Part down) (Elevator [I] is free) (BidirectionalConveyor* ***B*** *is free)*<I: Elevator/ ...>==>
<I: Elevator/ ...> *(Elevator[****I=2****] go down Part) (BidirectionalConveyor* ***B*** *down)*
R1: (*CriticalSection* ***CS*** *accept Part up) (Elevator [I] is free) (BidirectionalConveyor****B*** *is free)*<CS: CriticalSection/ ...> ==>
<CS: CriticalSection/ ...> *(BidirectionalConveyor* ***B*** *up) (Elevator [****I=2****] go up Part)*
(Line L accept Part ***P****)* [t1:*(Missfunctionnement CS)*]

Figure 6 Description of the welding line


VALID
Fichier Edition Chercher Fenêtres
A
Welding Line-[niveau 1]
Editeur de texte - c:\attoui\valid\weldline
Exécutere un action.
Microsoft Word
Gestionnaire

Unit - [4]

Entrée	Attribut	Sortie
m1 m2		m1 m3

RecyclingUnit - [1]

Entrée	Attribut	Sortie
m7		m2

TerminalUnit - [1]

Entrée	Attribut	Sortie
m4 m5		m6 m7

```
R0: [ Unit [M=5] accept Welded Part ] < U: TerminalUnit/...>
      ==> < U: TerminalUnit/...> [TerminalUnit U accept Welded Part ]
R1: [Unit [M=5] accept Unwelded Part from MainConveyor] < U: TerminalUnit/...> ==>
< U: TerminalUnit/...> [TerminalUnit U accept Unwelded Part from MainConveyor]
```

Figure 7 Snapshot under windows (Welding line level 1)

6 Conclusion

The specification and validation approach for manufacturing systems software development presented in this paper has the advantage to enhance the insight into and understanding of software requirements, helps clarify the customer's requirements by revealing or avoiding contradiction of specifications and ambiguities in the specifications, enables rigorous verification of specifications and their software implementation.

Verification of specifications would increase specification quality there by reducing life cycle costs. This approach is based on rewriting logic and multi-agent paradigm.

The development process has two main stages:

-The first one is a specification stage with a specific methodology for the analysis and the structuration of the command part of manufacturing systems, in terms of co-operative and specialized agents. The result is a conceptual model. Rewriting logic constitutes the formal framework.

-The second stage is a verification and validation stage based on a syntactical reduction engine.

After this two main stage it's possible to obtain the translation of the conceptual model into an executable model which can be directly used as a prototype. The executable model is generated into a target programming language (C++, ADA, VHDL...).

The environment for this approach is composed of a graphical editor for agents and rewriting rules description, a distributed inference engine for rules activation and a generator for the automatic translation of the formal specifications into a target programming language.

The approach and the environment have been used to develop a multi-agent application for a specific manufacturing system. We have used it for the specification of different kinds of applications and systems including manufacturing systems (Attoui,1994a) and transactional information systems (Attoui,1994b).

Although our environment constitutes a step towards a multi-agent architecture, it remains far from true multi-agent systems such as they are theoretically defined. Its evolution depends in effect on an integration of the modelisation of phenomenon like intentionality, rationality, commitment and representation of beliefs. Those phenomena have not yet attained the stage of concretisations.

References

AGHA G., HEWITT C. (1987) Concurrent Programming Using Actors: OOCP87, Yonezawa, MIT Press, .

ATTOUI A., SCHNEIDER M (1994a) Valid: An Environment Based on Rewriting Logic for the Formal Modelling of Manufacturing Systems: CIMPRO'94, Rudgers' Conference on Computer Integrated Manufacturing in the Preocess Industries, New Jersey, USA, April 25-26.

ATTOUI A., SCHNEIDER M (1994b) A Formal Approach for Prototyping Distributed Information Systems: IEEE International Workshop on Rapid System Prototyping, Grenoble, France, 21-23 June.

AYEL J. (1991) CIMES, un système d'intelligence artificielle distribuée pour la supervision en continu des activités de gestion de production: Thèse de Doctorat, Université de Savoie.

BINDING C., SARIA H., NIRSCHI H.(1991) Mixing LOTOS and SDL Specifications: FORTE'91, Sydney, 12-22 Nov.

BOURON T.(1993) Structure de communication et d'organisation pour la coopération dans un univers multi-agents: Thèse de Doctorat, Université de Paris VI, LAFORIA 93.04.

BOUZOUANE A.(1993) Un modèle multi-agent basé sur le tableau noir: application au pilotage d'une délégation d'assurances: Thèse de Doctorat, Ecole Centrale de Lyon.
BRUNO G., MARCHETTO G.(1986) Process translatable Petri nets for the rapid prototyping of process control systems", IEEE Transactions on Software Engineering, SE-12, February 1986.
BUDKOWSKI S.(1992) Estelle Development Tooltest (EDT): Computer Network and ISDN Systems, Special Issues on FDT Concepts and Tools, Vol.25, N°1.
BUSTTARD D.W., NORRIS M.T., ORR R.A., WINSTANLEY A.C.(1992) An Exercise in Formalizing the Description of Concurrent Systems: Software Practice & Experience, Vol 22, N° 12, Dec.
CARDOZO E.(1993) Using the object model to ilplement multi-agent systems: IEEE International Conference, Boston.
COELHO DA COSTA R.J., COURTIAT J.P.(1992) A True Concurrency Semantics for LOTOS: FORTE'92, Lannion (France), 13-16 Oct.
COURTIAT J.P., DIAZ M., MAZZOLA V.B., DE SAQUI-SANNES A.(1991) Description formelle de protocoles et de services OSI en Estelle et Estelle*- Expérience et méthodologie: CFIP' 91.
SIC 5TIMC/IMAG, Grenoble (1992) Modèles de connaissances et systèmes multi-agents: Journée Systèmes Multi-Agents du PRC-IA, 18 Déc. 1992, Nancy.
KELLERT P, FORCE C. (1992) Knowledge Model Building of Manufacturing Systems with SADT: 8th International Conference on CAD/CAM Robotics and Factories of the Future, Metz (France), 17-19 Août.
ISO 9074 (1989) Information Processing systems- OSI Estelle: a Formal Description Technique Based on an Extended State Transition Model.
FAYAD M. E.(1994) Objects Modeling Technique (OMT): Experience report: Journal of Object-Oriented Programming, Nov-Dec.
FRASER Martin D.(1994) Strategies for Incorporating Formal Specification in Software Development: communications of the ACM, N°10, Vol.37, October.
LAASRI H., MAITRE B.(1989) Coopération dans un univers multi-agents basée sur le modèle du blackboard: étude et réalisation: thèse de Doctorat, Université de Nancy 1.
LIGHTFOOT DAVID (1991) Formal Specification Using Z': The Macmillan Press.
IS 8807 (1988) LOTOS, a formal description technique based on the temporal ordering of observational behavior, December 88.
MAOUCHE A. and ATTOUI A.(1994) A programming environment for distributed applications: Proceedings of the 5th international training equipment conference and exibition, The Hague, The Netherlands, April 26-28.
MESEGUER J.,(1990) A Logical Theory of Concurrent Objects: Concur 90 Conference, Springer Verlag, Amsterdam, August 1990.
OCCELLO M.(1993) Blackboards distribués et parallèles: application au contrôle de systèmes dynamiques en robotique et en informatique musicale: Thèse de Doctorat, Université de Nice, Sophia-Antipolis 93.01.
PETERSON J.L.(1981) Petri Nets Theory and the Modelling of Systems: Prentice-Hall, Englewood Cliffs NJ.
SIJELMASSI R. and STRAUSSER B. (1991) NIST Integrated Tool Set For Estelle: Formal Description Techniques, Quemada (ed), North-Holland.
STINCKWICH S. (1993) Modèle et environnement objet dédié aux systèmes milti-agents: Premières journées IAD & SMA, Toulouse, Avril.
VIGDER M. (1991) Using LOTOS in a Design Environment: Proceedings FORTE'91, Sydney, 12-22 Nov.