A Pattern System to Supervisory Control of Automated Manufacturing System

Paulo Cézar Stadzisz[1], Jean Marcelo Simão[1,2] & Marcos Antonio Quináia[1,3]

[1] Federal Center of Technological Education of Paraná
Post-Graduation Program in Electric Engineering and Industrial Computer Science
Av. Sete de Setembro, 3165 - CEP 80.230-901 - Curitiba-PR – Brasil
http://www.cpgei.cefetpr.br
{simao, quinaia}@cpgei.cfetpr.br
stadzisz@lit.cpdtt.cefetpr.br

[2] Université Henri Poincaré (UHP)
Centre de Recherche en Automatique de Nancy (CRAN)
Présidence - 24-30, rue Lionnois BP 60120 - 54003 Nancy Cedex
http://www.cran.uhp-nancy.fr
simao@cran.uhp-nancy.fr

[3] State University of Center-West
Rua Presidente Zacarias, 875 - CEP 85015-430 - Guarapuava - PR
http://www.unicentro.br/
quinaia@unicentro.br

Abstract

Software patterns represent a promising research area in reason of the benefits happened of its application, mainly in terms of productivity reached with the reutilization. In automatics, patterns can be applied to recurring problems involving many types of computational systems. A complex domain of application, for which patterns can bring great contribution, is the Supervisory Control of Automated Manufacturing Systems (SC-AMS). This article proposes a system of patterns that aim to be applied in SC-AMS domain. The system is composed by an architectural pattern and three design patterns.

1. Introduction

Nowadays, a useful technique to compose computational systems is the *architectural pattern*. It expresses an organization or structural scheme, foreseeing a set of predefined subsystems, specifying its responsibilities and including rules and general principles to their organizations and relationships [6]. In fact, as general principle, the proposition of an architectural patterns is not a simple task, once a trade-off between efficiency in the performance of the instances and generality of the solution is needed.

To obtain a better organization and reusability degree in architectural patterns, a good practice is to define its subsystems in terms of design patterns, once these last ones are already well specified and possibly tested.

Architectural patterns based on the design patterns, can be applied in many application domains, as in telecommunications and automatics. In automatics, patterns are applicable, for

example, on the development of Supervisory Control of Automated Manufacturing Systems (SC-AMS). In fact, considering the typical complexity and dimension of SC-AMS, the development and use of architectural patterns can bring an important contribution to the developers.

Despite the numerous studies evolving SC-AMS [8][10][18][19], a lack of specific researches to the development of architectural patterns to these computational systems is noted [22]. This lack is especially related to aspects of the composition and execution of the control decision and consequent co-ordination of elements in the factory [24].

The conceiving process of an architectural pattern to SC-AMS is not a simple task because besides conceiving a strategy of factory control, it is necessary to generalize it in a set of situations of similar factory control.

Some approaches have been proposed in the literature as computational architectures or same as patterns to compose (in a certain way) SC-AMS [5][11][16][22][23], but none as architectural pattern composed by design patterns, regarding and solving the decision and co-ordination issue.

In this paper it is proposed an architectural pattern to this important area in computation and automatics called as Supervisory Control of Automated Manufacturing System (SC-AMS). The architectural pattern is based on design patterns, which are improvements of a computational architecture, which proposes strategies to effectively solve issues pertinent to SC-AMS, as the Monitoring & Command and the Regency (including the Decision and Co-ordination) [24][25].

The solution is agent based, where the agent classes specify a Generic Rules Based System (GRBS) [25]. Each instance of the architectural pattern is an Expert System (ES) with an advanced inference process, reached by the agent collaboration that results in incremental time growth in relation to the number of rules.

The proposed patterns are conceived from the analysis of supervisory controls of factories, including the simulated factory, modeled in the ANALYTICE II simulation tool [24]. ANALYTICE II allows expressing the fundamental characteristics of real industrial systems [14][23].

The architectural patterns is described following the POSA [6] format, whereas design patterns are presented as a mix of the two approaches very used called Alexandrian from [1] and GOF from [13].

The organization of this article is the following: section 2 is an overview about SC-AMS and its context; in section 3 there is an explanation over the design pattern Monitoring and in section 4 another explanation over the design pattern Command, while in section 5 presents the design pattern Regency and, finally, the section 6 presents the architectural pattern of SC-AMS in function of the presented design patterns.

2. An overview of SC-AMS

Before propose design patterns and an architectural pattern to SC-AMS composed by them, would be interesting a contextualization more detailed about Automated Manufacturing Systems (AMS), as well as about the Supervisory Control to AMS. In this sense, as example, this section presents simulated manufacture cell, its features and the related computational decisional system (specially the Supervisory Control).

The presented manufacture cell in Figure 1 is a system simulated in ANALYTICE II tool [14][23]. This manufacture cell is composed of various machines and their function is to produce fictitious parts of the types A and B.

Each processed part in this AMS has a *process plan* generated in another decision system called *Planning*. The plan specifies which machines the part must visit and which operation must be carried on through it [5][15]. The *process plan* for A part is {<Store> <Table 1> <Machine-Tool> <Table 2>} and for B parts is {<Store> <Table 1> <Table 3> <Lathe> <Table 3>}. There could still be an alternative of manufacture in the process plan, in case of an existing *Dynamic Scheduler* (with a *dispatcher*) to carry out the elections in execution time [23].

The *Supervisory Control* software role is to make the constituent elements of AMS (e.g. lathes and robots) work in a harmonic way to carry out the manufacture of the parts following the *process plans* [18][19]. In a general manner, the elements of an AMS can be classified in equipment, hierarchical elements and process elements.

A common division of the equipment is to classify them as execution (carry out operations over parts), transport (carry out the transport of parts) and storage (carry out storing parts). In the proposed example both Lathe and Machine-Tool are classified as execution equipment, while Puma, Kuka 386 and ER III as transport and, finally, Store and Tables as of storage.

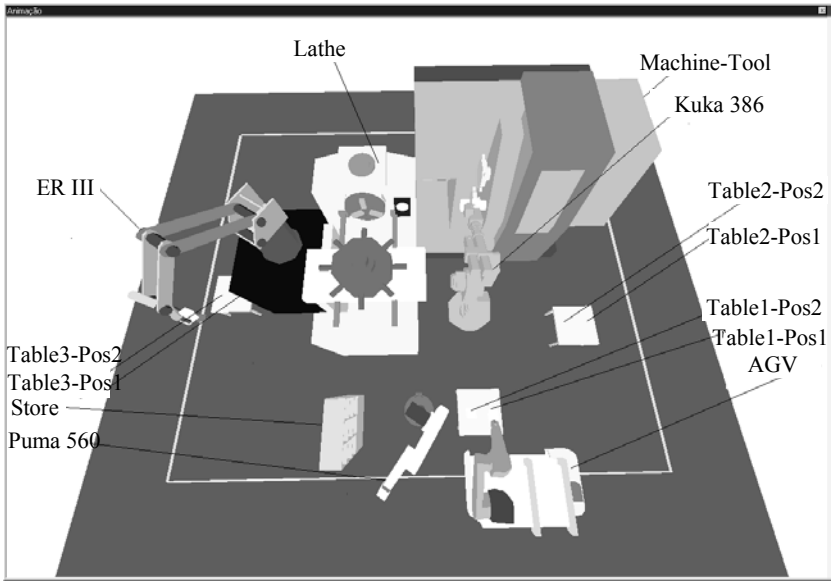

Figure 1 - Manufacture cell simulated in ANALYTICE II

The hierarchical elements are subsystems of an industrial plant, as the workstation (i.e. an equipment set), the manufacture cell (i.e. equipment set and workstations) and the plant (i.e. equipment set, workstations and cells). As example, the AMS illustrated in Figure 1, could have three workstations {<Lathe> <Table3> <ER III>}, {<Machine-Tool> <Table2> <Kuka 386>} and {<Store> <Table1> <Puma>}. The AMS as a whole could be considered as a composite cell by the three stations and the equipment of transport < AGV > (i.e. auto-guided vehicle).

This hierarchical division provides the SC-AMS development in several levels, known as "Hierarchical Supervisory Control" [15]. For example, a Hierarchical SC can determine that some parts go to a cell and not to another one. Once the parts are in the cell, another coordination level of this SC-AMS will determine which elements of that cell will process the parts.

The last type is the process element, which includes the parts (or products), the lot of parts and the pallets. One lot of parts consists of a parts group of the same type that advances in conjunction in the manufacture system. One lot has a processing priority and a production plan (to know which lot must visit which cell), allowing extending the scopes of supervisory control. Finally, one pallet is an element on which one or more parts (depending on the model) are placed for the purpose of protection and standardization in the transport. The pallets are limited resources in the AMS. Depending on the morphology of the parts, the AMS may not use pallets, as occurred in the studied example.

3. Design Pattern: Monitor

3.1 Intent

The intent is to propose a design pattern, called Monitor, as a generic solution to facilitate the creation of monitoring module in the design of the Supervisory Control of AMS.

3.2 Context

In the scope of Supervisory Control of AMS there exist the *monitoring*, which consist in to observe the discrete states of factory elements. The context of this design pattern Monitor is proposed a generic solution (regarding the reusability) to monitoring problem in SC-AMS. The idea is generically represent and specify the monitoring of the factory elements, in terms of their attributes.

To solve the question of monitoring it is needed to monitor the discrete states of the factory elements (e.g. equipment, work-stations and manufacturing-cells) and notify these states to interested elements (e.g. specially the Regency). In fact, to know these states it is fundamental to allow carry out the Regency and consequently the Command [5][24] as is argue after.

More detailed examples of AMS elements are equipment (e.g. robot, lathe and auto-guided vehicle), hierarchical elements (e.g. station-works, manufacturing cells and plants) and process elements (e.g. parts, lot of parts and pallets).

Each AMS element has attributes that specify its characteristics. As an instance, the robot can have an attribute to specify its state of work (i.e. free or busy) or another to specify the state of operation (e.g. turned in, turned off or out of order). All these states must be monitored in the SC-AMS and the Regency keep track of it.

3.3 Problem

The AMS elements, in the line of time, can assume different discrete states (e.g. robot moving, robot stopped, lathe free and lathe processing) to each attribute. These states can have strong influence over the process of decision (inside of the Regency), and then it is fundamental to monitor them. The monitoring problem consists in observing the most diverse equipment discrete states (that can be viewed as facts) and informing them to other elements of the Supervisory Control, specifically to the decision elements, in a standardized way [2][3] [5][7].

In a more detailed way, the main forces founded in monitoring problem are:

- Interface with a lot of different kind of elements (e.g. production cells, equipments and products) to know its discrete states.
- Deduce some discrete state when the monitored element does not have a direct feedback.
- Standardize the discrete states in a way that other elements (e.g. Regency) can understand and work with them in an easy way and in a high level.
- For each element, separate the standardized discrete states (that are correlated) in little sets (that can be called "attributes"). As example, in the case of a robot, the attribute "general state" can assume the states "busy or free" and the attribute "gripper" can assume the state "open" or "closed".
- Quickly inform (notify) the interested elements (and only the interested ones) about the discrete states (or facts) of elements attributes, having as objective to allow the system to be more reactive.

In terms of pattern, the problem is to find a generic way (respecting a trade-of with the applicability) to carry out the monitoring in agree with these cited forces.

3.4 Solution

To expose the solution, it is proposed the use of computational (classes of) agents. These agents are weak-deliberative, cognitive, reactive and cooperative. The solution comes from agents responsible by monitor each viewed element and directly notify the interested ones, for example other agents from the Regency.

In the sense of determining the meaning of the agent in this work, a computational agent can be defined as a software module, with high degree of cohesion, with well-defined scope, with autonomy and taking part in a certain context whose changes are perceived by the agent. These perceptions may change the agent behavior and it may promote other changes in the context [12] [20][21][26].

The referred agents are cohesive objects instanced from a hierarchy of classes created to treat classes of factory's elements. In fact, in the pattern instance, the instantiated agents (from low levels of the hierarchy of classes) permit to better specify specific characteristics, whereas the higher level of classes of agent gives the generic behavior of them. Each agent captures the states of the monitored elements by interfacing with feedback elements (e.g. sensors, hardware and software) or by deduction of states using determined artifice (e.g. watchdogs or information correlation) [24][25].

3.5 Structure

The agents responsible for monitoring are divided into two main classes (of a hierarchy of classes) entitled as *FBA* (from *Fact Base Agent*) and *AT* (from *Attribute Agent*), which the instances are respectively called ***fba*** and ***at***.

Each type of feature observed regarding an element is kept by an ***at***, e.g. the state of work from a robot (free or transporting) or the general state of this same robot (active or out of order). While the whole element (e.g. a robot) is managed by a ***fba*** (which computationally represents the element) that aggregates the concerned ***at***s, monitors the information from the element, standardizes the information (e.g. in a predefined set of symbols) and sends the standardized information to the interested and aggregated ***at***s.

The name ***fba*** was chosen considering that each discrete state observed by the aggregated ***at*** is, also, fact. Then, the set of ***fba*** with its ***at***s is considered as base of facts, like those from Expert System (ES).

In the diagram of Figure 2, the FBA is specialized in equipment-oriented, hierarchy-oriented and process element-oriented agents. And each level can be specialized in more specific levels, as the case of the Equipment_FBA a possible derivation is the classes to treat equipment to store, process and transport the parts.

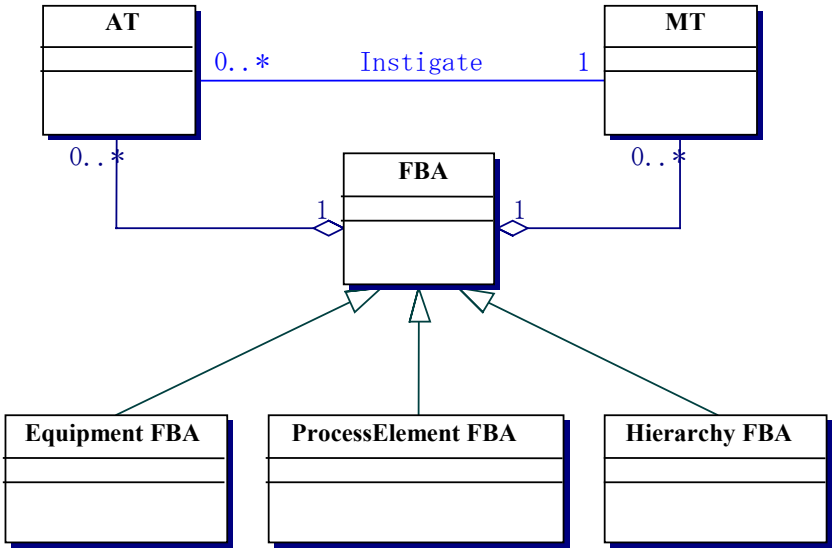

Figure 2 - Class Diagram to Monitoring.

3.6 Dynamics

A scenario for the execution (in a generic way) of the structure previously exposed could be the following:

- The ***fba*** monitors the characteristics of the elements of AMS and standardizes the information.
- The ***fba*** notifies the interested ***at***s responsible for maintaining, one by one the state of an attribute of the monitored elements.
- The ***at***s notify the pertinent Decision Elements and wait by a confirmation about the information treatment from these Decision Elements.

3.7 Consequences

The adoption of the Monitoring design pattern brings the following benefits:

- It "makes easier" to rewrite the monitoring component to work in a new SC-AMS. All the code that has to deal with specific characteristics of the environment elements is concentrated in the more specific levels of the hierarchy that has a standard interface (i.e. ***at***s). If it is required a specific change in the environment elements, it is expected that the changes in the code will be restricted to the code of the low levels. This allows a "quicker"

adaptation (in terms of project) to the new environment with the largest reuse of existing code.

- In sense of the instanced solution:
  - Monitoring is encapsulated in well-contained elements, with functional independence in relation to the other SC-AMS elements.
  - Monitored information is mapped as a set of symbols common to all the other SC-AMS elements, transforming the heterogeneous ones in homogeneous ones.
  - The use of ***at***s brings a special advantage, the notification mechanism that permits notifying the changes happened to the Decision Elements, avoiding traditional searches looking for states or facts.
- As liability:
  - To a simpler AMS this solution can be so robust and maybe could be not compensatory use it. Therefore, the solution is indicated to complex AMS where there exists a great number of information to be processed (monitored).
  - To compose the Monitoring agent-classes in lower level demand expert people and high level of technology integration, being imperative (to real case) study the solution applicability with the actual technology.

4. Design Pattern: *Command*

4.1 Intent

The intent is propose design pattern, called Command, as a generic solution to facilitate the composition of command module in the design of the Supervisory Control of AMS.

4.2 Context

The context of the design pattern Command consists in to specify (in a general way) the send of commands to some kinds of elements (e.g. cells, workstations or equipments), using appropriate protocols and information (e.g. process parameters). It is also part of the context some synchronization of commands given by the co-ordination (from Regency) to the factory's elements.

4.3 Problem

The force in the Command question consists in to give commands to the factory's elements targeting some activities (e.g. a lathe machining a part). However, these commands must be exposed in high level of abstraction (to facilitate the instigation from co-ordination) and after, each command, must be transformed into a command of low level (comprehensible by the commanded element), respecting specific protocols and with the appropriate parameters, to be sent to the target element. All this process is called command-refinement.

Another problem (or force) pertinent to the Command is the synchronization of activities ordered by the co-ordination. The synchronization occurs when an element will

receive an order, but it will not be executed because the element depends that another task be finished before (in one other element, which is its cooperator).

In the terms of design pattern, this problem must be exposed in a generic way, but also respecting a trade-off with the specific aspects needs to "easily" create instances of Command.

4.4 Solution

As solution to the command-refinement it is proposed generic class of agents, permitting derive more specific classes (therefore, a hierarchy of classes), which the consequent agents instantiated can work with specific and specialized knowledge. In fact, in the instances, there is an agent to treat each command applicable over an element of the factory and these agents are aggregated in the same ***fba*** responsible by the monitoring process of this element.

The synchronization is carried out by the ***fba***, once that the solution is modeled (encapsulated) in classes of high level in the *FBA* hierarchy. In a generic way, this solution consists in knowing what the prerequisites to an activity are, and always that a prerequisite is not available, it must consider the possibility of synchronization. In this case, the ***fba*** asks to its collaborator if, in a determined low space of time, someone will make the prerequisite true. If the response is positive then the ***fba*** wait for its collaborator, or else this is the beginning of the solution to the fault detection by the correlation.

A didactic instance of synchronization (in a specific case) is a ***fba*** responsible for a machine that receives an order to process a part, but the part is not yet in its scope because a recent order given by a robot to transport the part to it is still in execution. Therefore, this order given to the machine will be possible to be executed in few instants of time, demonstrating the importance of the ***fba***s communication to know the future possibility of execution and making the consequent synchronization of their activities.

4.5 Structure

Each agent responsible by a command-refinement is called ***mt*** (acronym of method agent), instanced from the some class derived from the *MT* (i.e. *Method Agents*). As more specialized is the agent, more levels of derivations can have its class. In other words, the specialized knowledge to apply a command over an element is encapsulated in a ***mt*** (from a low level class in the *MT* hierarchy).

Also are the ***fba***s, being each one responsible not only by the monitoring and the synchronization problems, but also by aggregate each ***mt*** that treat some command to the element over its responsibility.

4.6 Dynamics

In a general way, the dynamic of the Command design part is:

- A ***mt*** is activated by someone (in fact, by an ***oa*** - order agent - defined inside of next pattern) as a high level order.

- The ***mt***, once activated, translate the high level order in low level order dependent of the context, i.e. dependent of the specific knowledge pertinent to the element that will receive the order.
- The ***fba***, that aggregate the ***mt*** in question, verify the prerequisite and solve any possible synchronization.
- The ***mt*** gives the low command to the ***fba*** that "transport" it to the equipment, respecting the specific protocol and some possible synchronization.

4.7 Consequences

- The specific knowledge about command of AMS' elements is encapsulated in agent instantiated from a low level class (derived "from" the root class *Method Agent*), generating functional independence.
- As the specific knowledge and responsibility about each command of an element is embedded in a ***mt***, then the ***oa*** needs only activated the ***mt***, which out considers its specific details.
- The synchronization is made by the cooperation of ***mt***s, following a generic idea.
- The command and the monitoring are modeled inside of the same *FBA* class (or hierarchy of *FBA* classes), but independently because the subclasses *MT*s and *AT*s encapsulate the most responsibilities of each one.
- As liability, to develop the interfaces between the agents (from low level *MT*s) and the targeted elements (e.g. equipments) is still hard and dependently of specific knowledge and technology from the element. It is imperative (to real case) study the solution applicability with the actual technology state.

5. Design Pattern: *Regency*

5.1 Intent

The intent is to propose a design pattern, called Regency, as a generic solution to facilitate the composition of the "decision", "conflict-solution" and "co-ordination" integrated modules in the design of the Supervisory Control of AMS.

5.2 Context

The Regency (Decision, Conflict-Solution and Co-ordination) in SC-AMS.

The Regency responsibility is, regarding the facts monitored, to decide if some actions (pre-determined by the Planning) can be executed, resolve possible conflicts (using many information, included the arbitration from Scheduling) and co-ordinate the actions (pre-determined by the Planning), as well as given the orders that make part of the each action.

5.3 Problem

The problem can be divided in three sub-problems: decision, conflict and co-ordination.

Concerning to decision, the problem consist in relate or correlate observed facts by the monitoring (respecting the ways allowed by the Planning), make a logic calculus with the result of relations (and correlations) and decide what co-ordinations can be execute based in the resulting of the calculus and in the alternatives proposed by the Planning.

Related to Conflict, the problem is to identify conflicts (i.e. to know when there are two or more alternatives mutually exclusives) and solve it (i.e. to choose an alternative). More precisely, it is necessary identifies conditions where there are elements in competition by shared resources (e.g. a robot) and, based in some kind of parameter (e.g. from Scheduler), to decide what is the better option.

Finally, about Co-ordination, once having the conflict solved, it is needed to co-ordinate the orders (pre-defined) to instigate a certain number of (high level) commands [24].

In terms of design pattern, it is needed propose a generic solution to solve the Regency (Decision, Conflict and Co-ordination) problem, where to create instances be needed only give specific information to guide the generic solution.

5.4 Solution

The solution embedded in design pattern Regency is a sharing of responsibility, being the regency solved by a lot of computational (weakly-deliberative, cognitive, cooperative and reactive) agents, instantiated from a group of classes, that implement the knowledge of rules and also implement a conflict solver.

The solution generality is met in the structure of the group of agent classes, which allow instantiated agents only with the knowledge gave in rules, in a straightforwardly way, in the scope of the Supervisory Control targeted. In other others, the same structure can be used to any SC-AMS, being only needed give the parameters (e.g. knowledge of rules) to the generic structure. Still, is the knowledge of rules that will make to respect the restriction of Planning and Scheduling, once that this is specific to each system.

Each of these agents from rule is divided in others two to treat the condition and the action. The set of condition is the Decision sub-pattern[1] and the set of Action is the Coordination sub-pattern. Still there is the agent called Conflict-Solver justly to work over the conflict question.

In fact, the structure and interaction of agents compose the main solution of the regency. This solution is a new approach if compared with a lot of others solutions [23] [25].

5.5 Structure

The regency model is composed by the class *RA* (from *Rule Agent*) and *SA* (from *Solver Agent*). The *RA* instances are called ***ra***s and the *SA* instance is the ***sa***. The class *RA* has an aggregation relation to class *CA* (from *Condition Agent*, whose instances are the ***ca***s) and

[1] A sub-pattern is a well-identified and well auto-contained part of a pattern, but that cannot be separated because the cohesion with others parts of its pattern.

the *AA* (from *Action Agent*, whose instances are the ***aa***s). A ***ca*** is responsible by a fraction of the decision, as well as, an ***aa*** is responsible by a fraction of the coordination.

A ***ca*** is connected with ***pa***s (that are instances of *Premise Agents*), which collaborate with it to carry out its responsibilities. Each ***pa***s has the discrete value of an ***at*** (received by notification) called *Reference*, a logical operator (to make comparisons) called *Operator* and another value, called *Value,* that can be a constant. The ***pa*** makes a logic calculus comparing the *Reference* with the *Value*, using the *Operator*. The *Value* can be, still, other ***at*** value permitting, therefore, to correlate values of ***at***.

An ***aa*** is connected with ***oa***s (that are instances of *Order Agents*), which collaborate with it to carry out its responsibility. Each ***oa*** instigates changes in the factory elements by means of ***mt***s activations.

The way used to express the knowledge of the agents is a set of well-structured rules (oriented by agents attributes) as the exemplified in the Figure 3. In fact, the proposed approach is also a new way to compose the expert system, once each instance of the all architecture pattern is itself an expert system carried out by distributed agents [25].

The rule in Figure 3 is carry out by a ***ra*** (and its ***ca*** and ***aa***). The ***ca*** has the cooperation of tree ***pa***s and the ***aa*** has the cooperation of two ***oa***s. These agents make a robot transport a part from storage to a workstation when this workstation is free, the robot is free and the storage has a part.

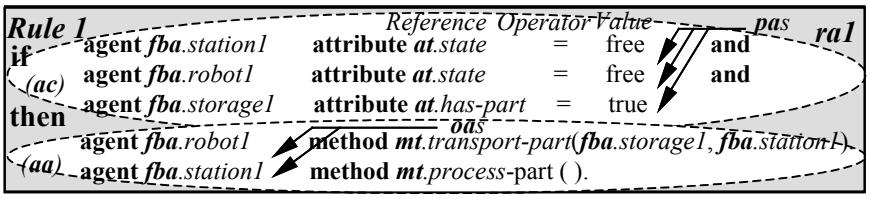

Figure 3 - Knowledge of agents in a rule format.

Expert agents that would create the ***ra***s can extract the knowledge from the rule. A way to implement this kind of agents is using linguistic comprehension or a friendly environment to rules composition.

Still there is the *SA*, which is created to generated an instance to work in the conflict moments. A conflict is established when two ***ar***s are in true state have an exclusive premise. An exclusive premise is one that has the *Reference* as being the "expression" of an exclusively shared resource, e.g. a robot that can serve two workstations, but in different times slice.

The ***sa*** is structured by a mechanism where each conflict established by *Premises* has a sub-agent responsible by taken the priority of the ***ra***s or other decision parameters (if the priorities are the same) and resolve the question. These alternative parameters can be, as instance, the values specified by a dynamic scheduler.

The Figure 4 is a UML class diagram of the proposed design pattern, where all the relation of the class agents (stated above) is expressed, included the class *SA*. These classes will allow to instance objects, which are a way to implement agents

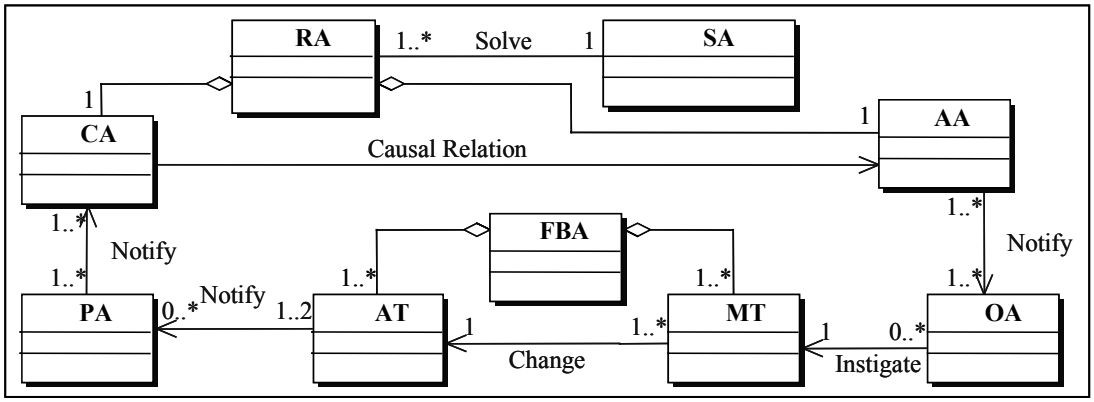

Figure 4 – Regency class diagram.

5.6 Dynamics

The ***pa***s receive notifications from the ***at***s (i.e. from its *References* and, when it is the case, from its *Values*) about the state change, once that the ***at***s know what ***pa***s have interest in its state. After the ***pa*** has received the notification with the new state, it uses this information to make a comparison (i.e. logical calculus), generating a boolean value to itself.

If the new boolean value is different from the last one, this is notified to the interested ***ca***s, that use this boolean value to make or re-make a logical calculus by conjunction with the boolean values of all connected ***pa***s. If the result of this calculus is true, then the ***ca*** put it respective ***ra*** in a true value.

After the ***at*** has notified all interested ***pa***s, it wait by a confirmation that the information was propagated by the ***pa***s. But the ***pa*** only confirms the propagation after the interested ***ca***s have confirmed their propagations. Evidently if (after a predetermined time) someone has not confirmed, it need to solve the problem (e.g. to notify again). This guarantees that all interested ***ra***s will be contemplated by the new facts.

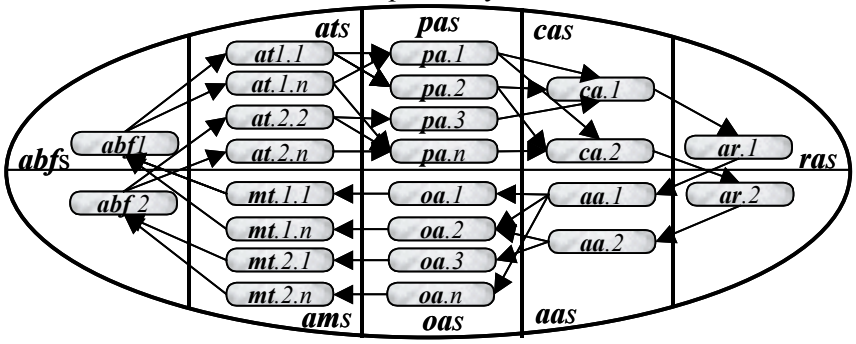

Figure 5 – The dynamic collaboration of agents (ellipses) using notifications (arrows).

When a ***ra*** has value true, it ***aa*** is passible of execution. To an ***aa*** be executed, its ***ra*** firstly verifies if all the ***at***s referenced in the collaborative ***pa***s are with the propagation confirmed. Then, after the resolution of a possible conflict, the ***ra*** activate its ***aa***. The ***aa*** is executed by the activation of the connected ***oa***s. Each ***oa*** instigates works in the ***mt***s.

The Figure 5 represents the notification process allowed by the agent structure of the architectural pattern, which this design pattern makes part.

Other relevant dynamic is the conflict identification. Once that a ***pa*** with exclusive attribute as *Reference* has been approved, and it collaborates to prove a rule it has a counter incremented. This counter represents the number of rules that it collaborates to be approved. Being this counter greater than one, the ***pa*** by itself notifies the ***sa*** to resolve the conflict. Once that ***sa*** is notified, some of its subagents take the priorities rules to decide impasse. But, if the priorities are the same, the ***sa*** demand a solution for whoever (e.g. the Dispatcher from Scheduler) or, in the absence of one interlocutor, it can choose randomly (being a default politics). The rule choose has the true from the exclusive premise confirmed and being approved, while the others have the true from the exclusive premise disapproved and, consequently, are disapproved too.

5.7 Consequences

The adoption of the Regency design pattern brings the following *benefits*:

- It makes easier to express the causal relation (to carry out the decision and coordination) by means of rules that work over objects attributes (or other methods that have mapping to this kind of rules, e.g. Object Petri nets) [4].
- It distributes the responsibilities in agents inside of the net.
- It resolves a complex problem with generic and simple classes of agents, where the complexity solution coming from the relations and cooperation among the instantiated agents that works following the relation established between the classes, mainly the relation called notification mechanisms.
- It better carries out the IE (Inference Engine), once that in an architectural instance the inference process works by notification relationship between agents, where the computational complexity is incremental in reaction to the number of the premises, because only the interested agents are notified and it is possible share information (by the share of ***pa***s among ***ra***s).
- It allows quickly identify the conflicts and resolve them by many ways.
- It promotes a well conjunction, cooperation and function separation of the decision and coordination, as well as, a good cooperation between monitoring and decision and between coordination and command.
- The respect about the determination of the Planning and Scheduling are implicit in the rules composition, letting the SC model more independently of this relation.
- As liability, in fact, it is a little complex to understand all the cooperation among the agents. But, happily, it seen “easy” to apply the solution only understand as compose the rules (it considered that the Monitoring & Command can be composed by expert people).

6. Architectural Pattern: Supervisory Control

6.1 Intent

Define the SC-AMS in three Designing Pattern: Monitor, Command and Regency. Each one carries out macro-functions in the subject system and works in an interactive way with each one, forming the whole Supervisory Control. The idea is “divide to conquer”, i.e.

divide the SC-AMS allow better understand its functions and presents solutions more functionally independent.

6.2 Motivation

In this section it is proposed an architectural pattern to an important area in computation and automatics, known as Supervisory Control of Automated Manufacturing System (SC-AMS). Effectively, contributions to conceive the systems in supervisory control are necessary due to the development complexity of this kind of computational system.

6.3 Known Uses

The ideas of the proposed architectural pattern can possibly be used in Supervisory Control of Automated Manufacturing System (SC-AMS) and it has been used in SC of emulated AMS. Also, there are efforts to demonstrate the model generality, as well as the major applicability of the solution [25].

To be more specifically, the robustness of the constituted architectural pattern, as well as the efficacy of the instanced systems of this pattern, have been observed inside the supervisory control systems applied over the industrial plant simulations made in ANALYTICE II. These tests include the presented plant as an example in this work (in section II).

6.4 Structure

The architectural pattern is composed looking for the maximum high degree of functional independence between the parts (i.e. design patterns). To each design pattern, it was adopted a policy "divide to achieve", being the functions distributed in separated elements with simple action, maintaining the complex cooperation among them.

The diagram of Figure 6 shows the structure of the solution proposed. These elements present the follow (generic) dynamic:

- The Monitor knows the states of the factory's elements (e.g. equipment) and notifies them to the Regency.
- The Regency, respecting the Planning and Scheduling, decides what to do (based in a set of options and solving possible conflicts among alternative solutions) and when to do the action to start the work, and make the coordination of orders to the factory's elements.
- The Command, instigated by the orders from the Regency, effectively gives the command to the element (with all needed parameters) and can also make some needed synchronizations.

After that, the factory elements receive the commands, the Monitor makes new observation, instigating the Regency and, consequently, stimulating the Command to become a cycle or work regime.

6.5 Problem Forces

The plant's elements need to receive discrete orders to carry out actions that allow the factory to work. However, these orders must be given in the appropriate moment, respecting the decision elements (planning and scheduling) and the viabilities of the elements (i.e. its discrete states), resulting in a harmonic interrelation among the commanded elements.

In terms of Architectural Pattern, all these functions exposed above must be modeled in a generic way, but allowing easy instantiation and generate robust, efficacy and efficient instances.

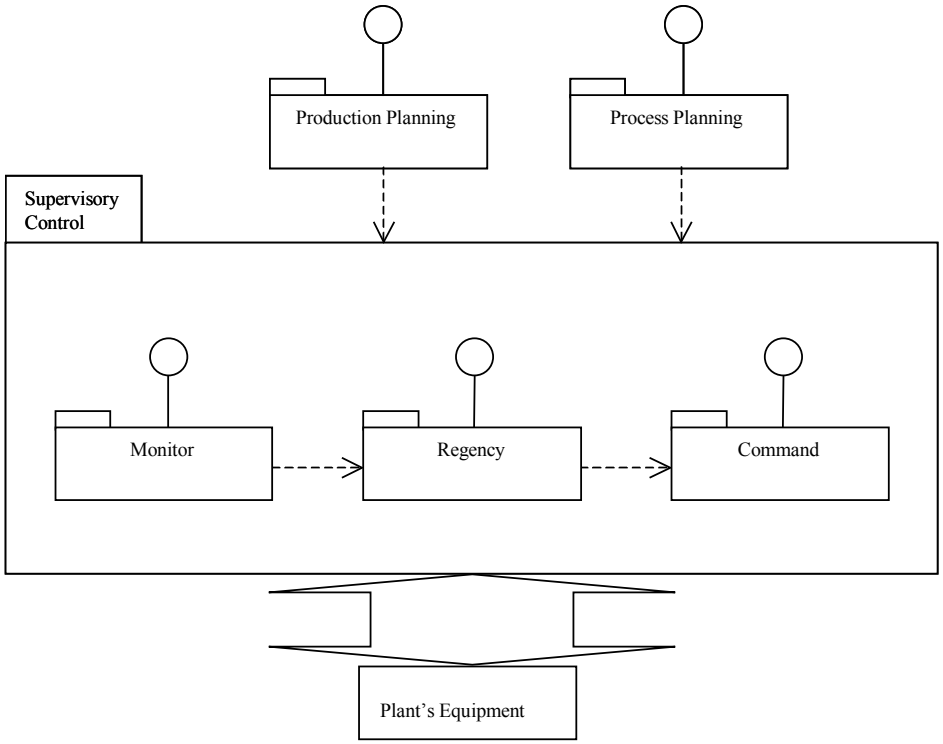

Figure 6 - Supervisory Control architectural pattern structure.

6.6 Benefits

The proposed architectural pattern is an improvement of the essay presented in the last SugarLoafPLoP 2002 [24]. The evolution is met in the specification of the architectural pattern in terms of design patterns, as well as the own advancement of the solution, like in the specification of the Conflict-Solver or in the aggregation of this conflict solver with the Decision and Co-ordination inside the unique element called Regency.

The solution presented includes concepts of artificial intelligence, once the model adopted is a kind of generic rule based system (GRBS), which instances allow carrying out CS-AMS. This model employs the agent concept in the instantiation of classes and uses an advanced and unique inference mechanism, by means of notification, reached by the agent collaborations (that permit the knowledge expansion) with an incremental time in the inference process.

In fact, the class agent concept utilization allows abstracting sub-systems that are cohesive, allowing creating well-defined frontiers and specifying the interrelation among them. As consequence, this agent-based solution still facilitates the archetype exposition in terms of design pattern. And then, the design pattern use make easier the reutilization, once the ideas are better explained inside a well-known standardization.

The utilization easiness is more evident observing the process to conceive instances. The instantiation of the Monitoring & Command takes place by the derivation of classes from the predefined generic hierarchical classes. Actually, in the case of the physics elements (e.g. equipment and its controls devices), this job would be easier if there were a well-defined way (e.g. protocol) to communicate with a computer, or else some artifices should be applied (e.g. deduction or sensors). While, the instantiation of the Regency is divided into the Decision & Co-ordination and Solver Conflict. To the first one it is enough to express the dynamic by rules (or other kind of compatible expression, like object Petri net) and transfer the knowledge to the predefined agent. To the last one it is possible to use the default solution specified, as well as, derive another one (like use of dispatcher agent).

As the architecture has well-defined interfaces, this facilitates the work of Planning and Scheduling, once they have to generate rules in the format predefined and standardized. Also, the incremental inference engine solution permits the use of a great number of alternatives without great effect over the SC-AMS performance.

6.7 Liabilities

Still, it was not developed a complete study about the availability (or weak features) of the solution to real cases in the industry.

6.8 See Also

As parallel work, it is being realized experiments to demonstrate that the architectural pattern can be viewed as a Petri net player, because it is known that exist a strong similarities between the syntax and applicability of rules of expert systems and Petri nets [4]. If the instances of the proposed Architectural Pattern can play any kind of ordinary Petri net, this is an interesting way to demonstrate the possible major range of applicability of the solution, once that Petri net are applicable to great number of discrete event controls.

Still in the theme of generality, one article was proposed in a congress called Logic Applied to the Technology – 2002 [25]. The article underlines a computational architecture as a generic and advantaged alternative form to compose expert systems. The idea consists basically in the use of more generic levels of the Monitoring & Command, as well as the use of the Regency. However, the article was not presented as an architectural pattern and even the architecture was less developed.

Another aspect already developed (and being improved) is a solution to compose rules oriented to class, and not only to objects, following and improving this good practice already known in the literature. However, it is still necessary to write this solution (called Formation Rules) in terms of a design pattern, in agreement to the explained architectural pattern.

An objective, as future work, is to (study the possibility) and applies the proposed architectural pattern to real systems. Future work also includes: (i) defining a distribution computational model of the design pattern; (ii) refining the framework, enveloped by a friendly computational environment to constitute the Expert Systems (to SC-AMS), following

the proposed architectural pattern; (iii) exposing the design patterns, from the proposed structural pattern, in terms of the existent standardizations in the literature, like the Gamma's Patterns [13]; and (iv) developing other architectural patterns for the conception and realization of other decision systems to the AMS, e.g. Planning, Scheduling and Fault Supervision, in an integrated way with the proposed architectural pattern to SC-AMS.

7. References

[1] ALEXANDER C., ISHIKAWA S. and SILVERSTEIN M., *A Pattern Language: Towns Buildings, Constructions*, Oxford University Press, New York, 1977.

[2] AARSTEN A., BRUGALI D. and MENGA G. *Designing Concurrent and Distributed Control Systems: an Approach Based on Design Patterns.* In Communications of the ACM - Special Issue on Design Patterns. 1996.

[3] AARSTEN A., ELIA G. and MENGA, G. *G++: A Pattern Language for the Object Oriented Design of Concurrent and Distributed Information Systems, with Applications to Computer Integrated Manufacturing* . In Pattern Languages of Program Design. Coplien, J. e Schmidt, D. (eds.). Addison-Wesley, 1995.

[4] BAKO V. and VALETTE R. *Towards a decentralization of rule-based systems controlled by Petri Nets: an application to FMS*. Fourth International Symposium on Kowledge Engineering, Barcelona - Spain. 1990.

[5] BONGAERTS L. *Integration of Scheduling and Control, In Holonic Manufacturing Systems*. (Ph.D. Thesis) KatholiekeUniversiteit Leuven, 1998.

[6] BUSCHMANN F., MEUNIER R., ROHNERT H., SOMMERLAD P. and STAL M. *Pattern-Oriented Software Architecture - A System of Patterns*. Wiley and Sons Ltd., 1996.

[7] BRUGALI D., MENGA G and AARSTEN A. *The Framework Life Span: A Case Study for Flexible Manufacturing Systems*. In Communications of the ACM. Out 1997.

[8] CHAAR J. K., TEICHROEW D. and VOLZ R. A. *Developing Manufacturing Control Software: A Survey and Critique*. The International Journal of Flexible Manufacturing Systems. Kluwer Academic Publishers. Manufactured in The Netherlands, pp. 53-88. 1993.

[9] COPLIEN J. and SCHMIDT D. (eds.) *Pattern Languages of Program Design*. Addison-Wesley, 1995.

[10] CURY J. E. R., De QUEIROZ M. H. e SANTOS E. A. P. *Síntese Modular do Controle Supervisório em Diagrama Escada para uma Célula de Manufatura*. V Simpósio Brasileiro de Automação Inteligente, Canela, RS, Brasil. 2001.

[11] FLETCHER M., BRENNAN R. W. and NORRIE D. H. *Modeling and reconfiguring intelligent holonic manufacturing systems with Internet-based mobile agents*. Jounal of Intelligent Manufacturing, 2003. Kluwer Academic Publishers in The Netherlands.

[12] FRANKLIN S. and GRAESSER A. *Is it an Agent, or Just a Program? A Taxonomy for Autonomous Agents*, Institute for Intelligent Systems – University of Memphis, In Proceedings of the Third International Workshop on Agent Theories, Architectures and Languages, Springer-Verlag. 1996.

[13] GAMMA E., HELM R., JOHNSON R. and VLISSIDES, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994.

[14] KOSCIANSKI A., ROSINHA L. F., STADZISZ P. C. and KÜNZLE L. A. *FMS Design and Analysis: Developing a Simulation Environment*. In Proceedings of the 15th International

Conference on CAD/CAM, Robotics and Factories of the Future, Águas de Lindóia, v.2. p.RF25 - RF210, 1999.

[15] KÜNZLE L. A. *Controle de Sistemas Flexíveis de Manufatura - Especificação dos níveis equipamento e estação de trabalho* (Dissertação de mestrado) CEFET/PR, 1990.

[16] LANGER G., SORENSEN C., SCHNELL J. and ALTING L. *Design of a Holonic Shop Floor Control System for a Steel Plate Milling-Cell*, In 2000 Int. CIRP Design Seminar on Design with Manufacturing: Intelligent Design Concepts Methods and Algorithms, Israel, 2000.

[17] MCFARLANE D., SARMA S., CHIRN J. L., WONG, C.Y. and ASHTON, K. *The Intelligent Product in Manufacturing Control and Management*. IFAC 2002, 15th Triennial World Congress, Barcelona, Spain. 2000.

[18] MENDES R. S. *Modelagem e Controle de Sistemas a Eventos Discretos - Manufatura integrada por computador*, Belo Horizonte, Fundação CEFET-MG, 1995.

[19] MIYAGI P. E. *Controle Programável – Fundamentos do Controle de Sistemas a Eventos Discretos*, Edgard Blücher, 1996.

[20] PRADO J. A., ABE J. M. and ÁVILA B. C. *Inteligência artificial distribuída; Aspectos*. Série Lógica e Teoria da Ciência, Instituto de Estudos Avançados – Universidade de São Paulo. 1998.

[21] SCHMEIL M. A. H., *Sistemas Multiagente na Modelação da Estrutura e Relações de Contratação de Organizações*. Faculdade de Engenharia Universidade do Porto. (Tese de doutorado). 1999.

[22] SCHMID H. A. *Creating the Architecture of a Manufacturing Framework by Design Patterns*. Fachbereich Informatik, Fachhochschule konstanz. OOPSLA'95, 1995.

[23] SIMÃO J. M. *Proposta de uma arquitetura para sistemas flexíveis de manufatura baseada em regras e agentes*. (Dissertação de mestrado). CPGEI/CEFET-PR. Brasil, 2001.

[24] SIMÃO J. M., QUINAIA M. A. e STADZISZ P. C. *Um Padrão Arquitetural para Sistemas Computacionais de Controle Supervisório*. In Segunda Conferência Latino-Americana em Linguagens de Padrões para Programação (SugarLoafPLoP), Itaipava, RJ, Brasil, 2002.

[25] SIMÃO J. M. and STADZISZ P. C. *An Agent-Oriented Inference Engine applied for Supervisory Control of Automated Manufacturing Systems*. Advances in Logic, Artificial Intelligence and Robotics, Volume 85, IOPress Ohmsha - 3rd Congress of Logic Applied to Technology - LAPTEC 2002, São Paulo, Brazil. 2002.

[26] YUFENG L. and SHUZHEN Y., *Research on the Multi-Agent Model of Autonomous Distributed Control System*, In 31 International Conference Technology of Object-Oriented Language and Systems, IEEE Press, China. 1999.