15th Triennial World Congress, Barcelona, Spain

COCA – MODEL OF DESCRIPTION AND SYSTEMS RECONFIGURATION USING FUZZY LOGIC

Cyrille PETITJEAN, Jean Marc PERRONNE, Michel HASSENFORDER

Laboratoire MIPS, Equipe MIAM, Projet COCA, Université de Haute Alsace
ESSAIM - 12, rue des Frères Lumière, F-68093 Mulhouse Cedex, France
{c.petitjean, jm.perronne, m.hassenforder}@uha.fr

Abstract: This paper proposes software architecture about conception and reconfiguration for the control of complex systems. This approach is based on a component description model. This model empowers the design of a system as a hierarchic composition of components. Dynamic insertions and removes of components in the system are also allowed. With this architecture, each component can be described with uncertainty. Thus, in the case of faulty behavior, a fuzzy evaluation of the system reconfigurability seems appropriate. To illustrate this analysis and this description, an example of the reconfiguration of a trajectory tracking is proposed.

Keywords: Fault Tolerant Control, Reconfiguration, Fault Accommodation, Fuzzy Analysis, Software Architecture, Object Oriented Programming.

1. INTRODUCTION

The increasing needs of flexibility, modularity, adaptability and safety in the industrial processes are today the source of numerous works.

A research area concerning these problems has particularly developed during these last years. It provides the notion of fault tolerant control and fault tolerant systems. A survey (Patton, 1997) proposes a state of the art in the field of fault tolerant control.

In this frame, the COCA (COmponent CAncelling) project objective is to design software architectures. They have to support all the reconfiguration processes induced by a faulty system in order to maintain the integrity of the system and the control strategies.

When a fault occurs, the reconfiguration process proposes automatically an alternative solution in terms of replacement and/or in terms of internal reconfiguration of the components of the system. The aim is to maintain the system availability to achieve objectives.

The led works try first to develop a description model. This model depicts the studied system. In a second step, an analysis method should be defined. For each sampling time, an evaluation of the reconfiguration possibilities for the defective system should be proposed. At third, (if necessary), a remedial action, allowing a correction of the faulty system, is achieved.

To determine the reconfiguration strategies, this paper proposes a modern architecture of dynamic and adaptive composition. This architecture has to provide:

- A system description in terms of components.
- According to the system components composition, an analysis of the possible reconfiguration strategies should be provided. The use of fuzzy algorithms should increase the precision of the decision.

This article is organized as follows: a first part presents the context of fault tolerant systems and of components reconfiguration. A second part exposes our approach by describing the chosen description model, its limits and its needs. The third part explains the reconfigurability analysis and the reconfiguration using a fuzzy logic way. Finally, a reconfiguration analysis of a trajectory tracking illustrates the presented concepts.

2. RECONFIGURATION BASICS

2.1 Problem analysis

In the case of industrial production systems and in embedded systems, the control of complex process is based on non-robust systems (Hoblos, *et al.*, 2001).

When a fault occurs, according to the context, three adaptative responses should be distinguished (Blanke, 2000) :

- Fault accommodation : Finding a corrective action to accommodate the system controller to the default. The system is not informed of the taken action.
- Controller reconfiguration : Revising the objectives of the current system. New achievable objectives are given.
- System reconfiguration : Reconfiguring the system himself . New system and objectives are set up to replace the faulty system.

These three cases implies the three following needs :

- A systems description model. This description must be an image of the controlled system and must introduce the properties of dynamic composition and dynamic reorganization.
- A method for the reconfigurability analysis. This analysis should be an image of needs and reconfiguration abilities of the faulty system.
- A reconfiguration process. This process performs the results given by the analysis and the evaluation of the system reconfigurability. These results are associated with a decision in the aim to adapt the system to the occurred fault.

2.2 Previous works

The automated systems and the control of industrial processes are a large area of experiments. Numerous works propose results; particularly, a model of systems description focuses on a notion of services versions (Gehin, *et al.*, 1999a). Based on this description, analysis methods for the system reconfigurability evaluation are proposed : a graphic approach (Staroswiecki, *et al.*, 1999), a bottom up approach (Gehin, *et al.*, 1999b) and a formal approach (Gehin, *et al.*, 1999c). The figure 1 illustrates the structure of the chosen model.

From user point of view, a component (the version) supplies a service. This component can be combined with others to supply a higher-level service. These services are the result of a sequential or parallel combination of lower levels services.

The system reconfigurability analysis is primarily based on this description model. Three approaches are already proposed in this scope to analyze the system reconfigurability and the reconfiguration possibilities: a graphic approach, a bottom-up approach and finally a formal approach.

```
<model> :=      <state/transition graph(EM,τ)>
<EM>    :=      <set of exploitation modes>
<τ>     :=      <set of transitions>
<transition>:= < condition,
                 input mode,
                 output mode>
<exploitation mode> := <list of missions>
<mission> := < ordered list of services >
<service> := < ordered list of versions >
<version> := < consumed variable(s),
               produced variable(s),
               procedure,
               activation conditions,
               hardware resource(s)>
```

Fig. 1. Description of an automated system.

2.3 Limitations and evolutions

Within the framework, two limitations appear:

- The composition of components does not lead to a unified approach of the system.
- A drastic determinism: the availability of the components is described only in a binary way; the states associated to the hardware resources can be only available or faulty. This appears within the framework of the reconfiguration processes by the fact that a system element is considered only as active or inactive.

The purpose of the CoCa project is to develop software architectures, which allow:

- A description of components with a certain uncertainty and an introduction of expertise concerning the availability of the system. The use of the fuzzy logic seems appropriate.
- An evaluation of the reconfiguration possibilities, based on fuzzy evaluation and fuzzy algorithms.
- A reconfiguration process where a system can be modified as needed. It chooses the best strategy in order to adapt the system.

3. DESCRIPTION MODEL ARCHITECTURE

3.1 Software architecture and design patterns

The conception of modular software architecture, allowing a description of the manipulated systems, is based on the use of design patterns. (Gamma, *et al.*, 1995) describes in detail the utility of object-oriented techniques in the field of the software design. (Booch, 1992) proposes a complete description of the concepts, methods and applications bound to the object-oriented programming (OOP).

A system can be described by a Directed Acyclic Graph (DAG) of components, which can be himself simplified as a hierarchical composition of

components. The Composite design pattern allows such a construction (Gamma, *et al.*, 1995).

With the model based on components, the recursive evaluation of every component gives respectively the production (behavior) of the service, the production (how far from the end) of the mission, and the production of the complete system. Still there, the Interpreter design pattern allows this construction (Gamma, *et al.*, 1995).

The figure 2 describes the architecture of this two design patterns using the Unified Modeling Language (UML) (Muller, 1997).

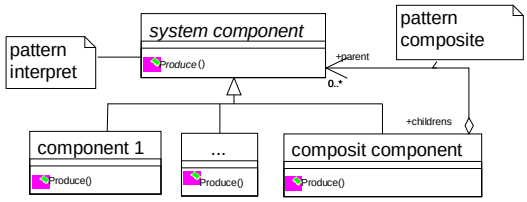

Fig. 2. Composite & Interpret design patterns

The description model of a trajectory tracking illustrates the application presented in the section 5. The figure 3 describes an example of a composition of objects representing trajectory elements. A trajectory can be indeed described as a composite structure of these elements.

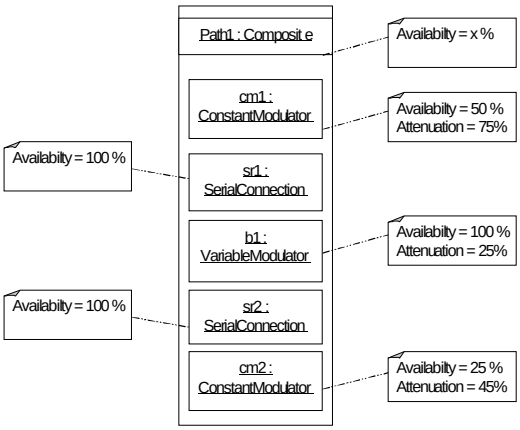

Fig. 3. Example of a path.

3.2 Basic Elements of the model of description

An element is described by means of a tupple <consumed variable(s), produced variable(s), law(s) of production>. The abstract class *Element* defines an interface for the elements, which compose the model.

This interface declares the abstract behavior of the various elements of the composite structure; it defines the transformation law between the input variables and the output variables. Every sub-class has the responsibility to implement a concrete production relation (*Produce* method).

The *availability* attribute traduces the physical availability of each element. This attribute results from the analysis of appropriate diagnosis process for all elements. It is used during the stage of the reconfigurability evaluation.

Through the *Composite* and the *Elements*, the composition of the different elements describes complex systems models.

The concrete class *Source* defines the source(s) feeding the system.

The class *Storage* defines element having a capacity to store the manipulated entities.

The abstract class *Modulator* defines element, which modulates the stream between elements by a modulation factor. The real behavior is given through concrete implementations; the first two propose *constant* and *variable* modulation factor.

The abstract class *Connector* defines the connections between elements. Two possibilities are proposed for *serial* and *parallel* connections.

The figure 4 illustrates the complete software architecture developed within the framework. This approach allows all latitudes for the designer to enhance or to specialize the behavior of these basic elements.

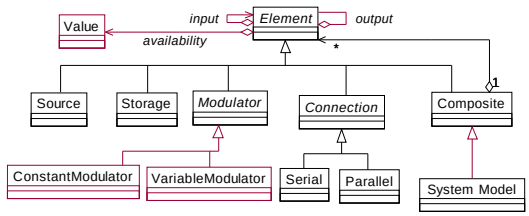

Fig. 4. Composite structure of the description model - class diagram

3.3 Illustration

The construction of the proposed example (figure 3) is following:

```
ConstantModulator cm1 =
       new ConstantModulator(75, 50);

VariableModulator b1 =
       new VariableModulator(25, 100);

SerialConnection sr1 =
       new SerialConnection(cm1, b1);

ConstantModulator cm2 =
       new ConstantModulator(25, 45);

SerialConnection sr2 =
       new SerialConnection(b1, cm2);

Composite way= new Composite();
way.add(cm1);
…

way.produce();
```

This composite describes and models a path from a point to the other one (points are not described here).

4. RECONFIGURABILITY ANALYSIS AND FUZZY LOGIC

At first, this section describes an approach to provide reconfigurability analysis to the systems. The fuzzy logic analysis method, organized to implement the reconfiguration algorithm, is presented in a second time.

4.1 Systems Reconfiguration and Fuzzy Logic

For the modification of each element of the system (The addition, the retreat or the modification of an element), an analysis of the system reconfigurability is achieved. Furthermore, during this process, a factor of availability concerning the use of the elements could be interesting. To increase the robustness of the analysis and the progressiveness in the evaluation, the reconfiguration process works as depicted in figure 5.

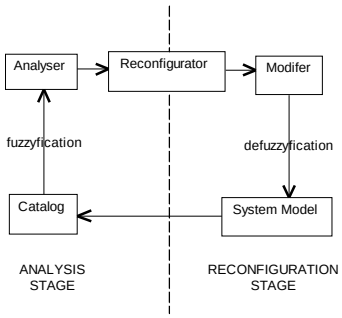

Fig. 5. Fuzzy analysis and fuzzy reconfiguration

The chosen structure to implement these imperatives is the following:

- A modification of an element implies an evaluation of the nature and of the state for all components in the system. The fuzzyfication stage is involved in the perception task of the *Analyzer*. According to the result, a reconfiguration will be done or not. This component just analyzes the needs of reconfigurability when a fault occurs.
- The reconfiguration component (*Reconfigurator*) takes automatically care of the outputs of the *Analyzer*. This component activates, if needed, a concrete modification at the system level. It chooses the actions to implement.
- The *Modifier* component gives some concrete behaviors for the actions, which should be implemented for the system reconfiguration. These actions were deducted from the analysis of the *Reconfigurator* component.

The *Analyzer* output provides to the *Reconfigurator* component the information about the nature of the action to take. According to this information, *Reconfigurator* will implement concrete reconfiguration behaviors. Then this component gives to the *Modifier* these behaviors and then the *Modifier* update the model of the system to avoid the consequences of the fault. This paper focuses only on an analysis of the faulty system. Thus, the description of these two last components is not describes here.

4.2 Fuzzy inferences system

The figure 6 illustrates the architecture proposed by (Perronne, 2000). The designer can create simply in the semantics of the fuzzy logic experts, a complete fuzzy inferences system.

With this architecture, the fuzzy operators and the fuzzy rules involved in the reconfiguration stage can be expressed. The designer has to specialize some classes as particular operators or membership functions to adapt the possibilities offered by this architecture to the particular case of the problem to resolve.

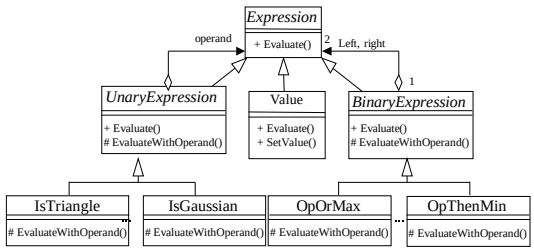

Fig. 6. Composite structure of a fuzzy expression

4.3 The Analyzer: Fuzzy evaluation of elements

A multivalent logic is required to take into account the uncertainty of the state and the availability of an element. Each element of the framework has these skills. Each element is referenced in a component called Catalog; it gives a way to find them.

The application of the fuzzy rules of the *Analyzer* determines the nature of the modifications that should be done. After the aggregation, the four output possibilities are:

- Nothing.
- A fault accommodation.
- A revision and an adaptation of the objectives.
- A complete reconfiguration of the system.

5. TRAJECTORY RECONFIGURATION EXAMPLE

5.1 Presentation

The works of reconfiguration led within the project CoCa are materialized through two mobile robots. The manipulated concepts are shown through this window.

The figure 7 presents the graphic user interface of the developed application.

The description of a trajectory tracking is the following one: The user has to specify start-points, end-points, intermediate-points and finally the speed of the mobile. Each point is an element; it can be modified, added, removed, or replaced at any time.

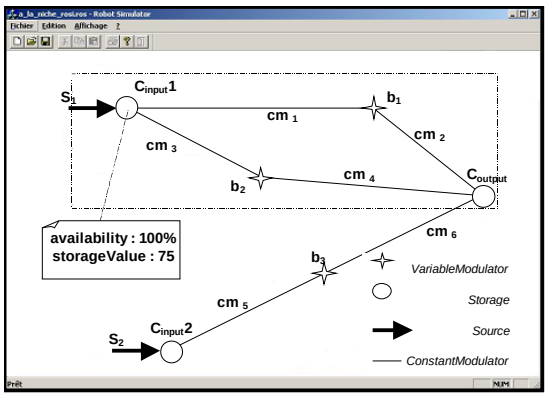

Fig. 7. User interface of the application of reconfiguration of a trajectory tracking

5.2 Model construction

According to the presented architecture, a description of the proposed system for a trajectory tracking is:

- Two *Sources* elements (S_1 and S_2) describe the initial speed of mobile V_0, considered here as constant. It allows thus the movement of the mobile platform {Si.deliver(aValue)}.
- Two *Storage* elements (C_{input1} and C_{input2}) associated respectively to sources S_1 and S_2, describing the possible start-points {Ci.store}.
- A *Storage* element (C_{output}) describes the end-point of the movement {C_i.store}.
- A set of *VariableModulator* describes the intermediate points (b_i, i ∈ N*) {b_i.letPass(aValue)}.
- A set of *ConstantModulator*, image of the movement-resistance between every intermediate points. {cm.letPass(aValue)}.
- A set of *Connectors* allows the parallel and\or serial interconnections between elements.

All connectors are considered as always available. During the elaboration of the model, the availability of every created element is initially fixed to 100 %.

Let us consider the composition of the following services (the figure 7 give an illustration of the problem through a decomposition under a hierarchical structure):

- $StartPoint_1$ constituted with *Source* S_1 and *Storage* C_{input1}.
- $StartPoint_2$ constituted with *Source* S_2 and *Storage* C_{input2}.
- $Path_1$, constituted with *ConstantModulator* cm_1, being connected in series with *Variable Modulator* b_1 and with *ConstantModulator* cm_2.
- $Path_2$, constituted with *ConstantModulator* cm_3 being connected in series with *Variable Modulator* b_2 and with *ConstantModulator* cm_4.
- $Path_3$, constituted with *ConstantModulator* cm_5 being connected in series with *VariableModulator* b_3 and with *ConstantModulator* cm_6.
- StopPoint constituted with *Storage* C_{ouput1}.

The associated missions to the system behavior are then the result of the composition of these elementary services. These services are based on the availability of the hardware or physical resources described by elements cm_1, b_1, cm_2, cm_3, b_2, cm_4, cm_5, b_3 and cm6. In this application, the services of higher level are:

- Go from $StartPoint_1$ to StopPoint ($path_1$: C_{input1}, b_1, C_{output})
- Go from $StartPoint_1$ to StopPoint ($path_2$: C_{input1}, b_2, C_{output})
- Go from $StartPoint_2$ to StopPoint ($path_3$: C_{input2}, b_3, C_{output})

The higher-level service of the robot is thus to go to StopPoint.

5.3 Fuzzy reconfigurability analysis

The system ability to adapt or to cancel his current functioning is evaluated truth the *availability* attribute.

A binary evaluation is too drastic and does not take account the fact that a component is not just available or unavailable. To modeling this reality, and to provide more effectiveness to this evaluation, a fuzzy estimation of the system availability could be achieved.

The space of evaluation described by using a triangular membership function with the following availability degrees:

- Low {(0, 0)(20,1)(40, 0)}.
- Medium {(30, 0)(50,1)(70, 0)}.
- High {(60, 0)(80,1)(100, 0)}.

These three groups allow to distribute in a fair way the variation range of *availability*, this for every *Element* of the system. The universe of discourse is defined from 0 to 100 % of availability. The output named *Action* is defined by these four fuzzy sets:

- To do nothing (*Action* is *Nothing*).
- To accommodate the faulty system (*Action* is *FaultAccomodation*): If one beacon (b_1 or b_2) is faulty, then the condition allowing an

Accommodation of this fault is that the second beacon remains available.

- To revise the objectives of the system (*Action* is *ObjectiveRevison*): If beacons involved in a mission are unavailable, it is necessary to modify the system objectives. If b_1 and b_2 are unusable, the mission "go from D_1 to A" is unachievable. It is then necessary to try to achieve the second mission: "go from D_2 to A".
- To reconfigure the system (*Action* is *SystemReconfiguration*): If all beacons come to be useless, a reconfiguration is required because no achievable mission remains.

According to this logic, the fuzzy inference system rules are:

- If b_1 is low then Action is FaultAccomodation
- If b_2 is low then Action is FaultAccomodation
- If b_1 is low and b_2 is low then Action is ObjectiveRevision
- If b_1 is low and b_2 is low and b_3 is low then Action is SystemReconfiguration

The figure 8 presents a part of the fuzzy inference engine.

This illustration is voluntarily based on the only consideration of the availability of the intermediate points defining the trajectory. The availability of all elements should be taken into account to define a complete analysis of the system reconfigurability.

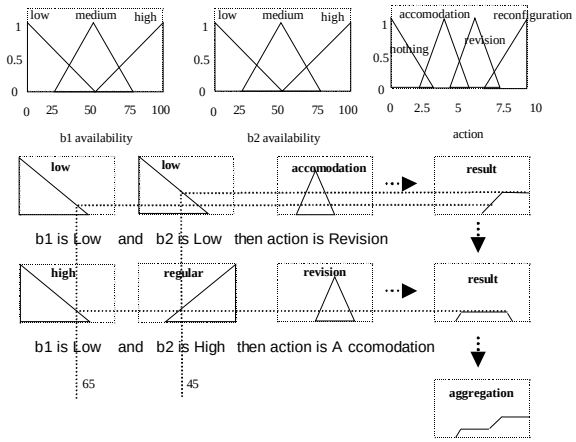

Fig. 8. Part of the fuzzy inference engine

6. CONCLUSIONS

This article presents an approach for the design of fault tolerant systems, using a generic description. It is based on the notion of service and it allows a dynamic reconfiguration of the faulty system.

To support it, software architecture based on software design patterns is proposed. This software architecture allows the dynamic composition of components. The analysis of the system reconfigurability is achieved by the use of fuzzy logic algorithms. A trajectory tracking illustrates the presented ideas.

7. REFERENCES

Åström et al., "C*ontrol of Complex Systems"*, Springer-Verlag, 2000, ISBN 1-85233-324-3.

Blanke M. et al., "*What Is Fault Tolerant Control*", IFAC SafeProcess'2000, Budapest, p. 312-323, June 14-16, 2000

Booch G., "*Conception Orientée Objets et Applications"*, Addison-Wesley, 1992, ISBN 2-87908-004-5.

Gamma, E.Gamma et al, "*Design patterns, elements of reusable O.O. software"*, Addison-Wesley, 1995, ISBN 2-84180-054-7.

Gehin A.L., Staroswiecki M., "*Modèle de description d'un système automatisé tolerant aux fautes*", MSR'99, Paris, Mars1999.

Gehin A.L., Staroswiecki M., Assas M.L., "*A Bottom-Up Approach to Analyse Reconfiguration Possibilities*", 10th Int. WorkShop on Principles of Diagnosis DX'99, Loch Awe, UK, 1999

Gehin A.L., Staroswiecki M., "*A Formal Approach to Reconfigurability Analysis, Application to the Three Tanks Benchmark*", European Control Conference ECC'99, Karlsruhe, August 31-September 3, 1999.

Hoblos G., Staroswiecki M., Aïtouche A., "*Tolérance aux Fautes de Capteurs et d'Actionneurs*", Journal Européen des Systèmes Automatisés, n°3/2001, p. 331-352.

Muller P.A., "*Modélisation Objet avec UML*", Eyrolles, 1997, IBSN 2-212-08966-X

Patton R.J., "*Fault Tolerant Control, the 1997 situation*", IFAC SafeProcess'97, Hull, UK, vol. 2, p. 1033-1055, August 26-28, 1997.

Perronne J.M., Amann P., Thiry L., Hassenforder M., "*A Framework for the building of Fuzzy Logic inference systems*", technical report COCA.

Scholten H., *"Logique floue et régulation PID"*, PubliTronic,1994, ISBN 2-86661-0490.

Staroswiecki M., Attouche S., Assas M. L., "*A Graphic Approach to Reconfigurability Analysis*", 10th Int. WorkShop on Principles of Diagnosis DX'99, Loch Awe, UK, 1999