Self-Reconfigurable Control - Part I: Modeling Based on Automated Planning

He-xuan Hu [1,2]
Agricultural and Animal Husbandry College of Tibet University [1]
Lin-zhi, Tibet, P.R. China
College of Energy and Electrical Engineering
Hohai University [2]
Nanjing, Jiangsu Province, P.R. China
e-mail: hexuan_hu@hhu.edu.cn

Abstract—This paper presents a new formal modeling framework for reconfigurable control. The main idea is to use the STRIPS (STanford Research Institute Problem Solver), a formal language for automated planning which can integrate the cause-effect knowledge and the automated reasoning mechanism into one model. A qualitative model based on a graph of states is chosen. Transitions between states are determined from the possible actions (as defined as controls to the actuators). Each action is modeled in terms of preconditions and effects. The STRIPS can qualitatively define the fault models without requiring detail and precise knowledge of faulty components. Automated planning is an interesting tool for reconfigurable control. Since the process may end up in any one of a very large number of states after a break-down, it is not realistic to have recovery plans designed in advance for all such states. It is, thus, desirable to invoke a planner in the case of break-downs, which finds a plan that brings the process back into normal operation again, and that is the nature of reconfigurable control.

Keywords-STRIPS; Automated Planning; Reconfigurable Control; Cause-effect; Qualitative Model

I. INTRODUCTION

The notion of "reconfiguration" has been widely used among the various domains, for example, Autonomous UAV (unmanned aerial vehicle) [1], Multi-robot System [2], deep space missions [3], [4], and flexible and reconfigurable manufacturing systems [5], [6], and [7]. A reconfigurable system can be defined as a system with an architecture consisting of the components of the system and a configuration that specifies how these entities are used to meet a goal, and it has a mechanism to choose a new configuration and set it up in the process by itself. As indicated in [8], "Autonomous systems" have appeared largely in space exploration, elderly care and domestic service; they are particularly attractive for such applications because their advanced decisional mechanisms allow them to execute complex missions in uncertain environments. In [9], Williams and Nayak have proposed an idea that is "Immobile Robots" similar to the notion of total autonomous system. Immobile Robots need the sophisticated regulatory and immune systems that accurately and robustly control their complex internal functions. They will use these models to dramatically reconfigure themselves in order to survive decades of autonomous operations. Achieving these large scale modelling and configuration tasks will require a tight coupling between the higher level coordination function provided by symbolic reasoning, and the lower level autonomic processes of adaptive estimation and control. Self-modelling and self-configuration, coordinating autonomic functions through symbolic reasoning, and compositional, model-based programming are the three key elements of model-based autonomous systems architecture. That is the reason why, we propose in this paper, a qualitative self-modelling algorithm which reflects the real system possibilities (updated itself in presence of faults) and which allows the search of an adequate control strategy for reconfiguration.

The faults considered in this paper are faults which totally change the system's physical structure, such as the complete loss of an actuator or the complete loss of a system component. The system model is no longer valid in this case. This is the reason why, we propose a flexible system modelling mechanism to automatically update the system model when faults occur. The mechanism rests on two parts: a database containing basic knowledge about the system, and an algorithm for constructing a system model based on that knowledge.

There are different modeling requirements in different domains. In the context of fault diagnosis and fault tolerant control, the modeling is expected to satisfy the following requirements as mentioned in chapter 1, [23]:

(1) gaining experience and understanding of the physical process to be controlled, including the constraints set by nature and circumstances surrounding the job;

(2) setting goals that are attainable, or objectives along with tradeoffs, that the computer can "understand" sufficiently well to give proper advice or make control decisions;

(3) formulating a strategy for going from the initial state to the goal state;

(4) translating goals and strategy into detailed instructions to the computer such that it can perform the task automatically;

(5) monitoring the behavior of process;

(6) detecting the presence and location of faults, or conflicts between actions and goals, and the anticipation that either of these is about to occur;

(7) making the correct decisions after the presence of faults.

To achieve the challenge in reconfigurable control, we propose a qualitative model based on the automated planning (STRIPS) and a self-verification mechanism based on model checking. The automated planning offers the reasoning ability to formulate the physical process, goals and strategy for going from the initial state to the goal state. The model checking offers the verification ability to make control decisions which will be presented in the sister paper: Part II.

This paper is organized as follows. In section II, we propose to use a state transition representation STRIPS (Stanford Research Institute Problem Solver) to formalize the system's knowledge. In section III, we present the system modeling algorithm. In section IV, we summarize the work done and discusses directions for future research.

II. SYSTEM KNOWLEDGE FOR SYSTEM MODELING

The basic system knowledge of the system corresponds to the observations, diagnoses, and description. These elements refer to three different kinds of knowledge [10]: variable knowledge, semi-variable knowledge, and fixed knowledge. The first includes such information as observations, which vary according to the execution of control commands and environmental changes; the second includes such information as diagnoses, which only change if the system moves from the normal to faulty operation mode and vice versa, and the last includes such information as system descriptions and domain priorities, the non-changing information that expresses the intrinsic system properties as intended by the designers. Domain priorities allow expressing the possibility to achieve "degraded" or non-nominal objectives in faulty modes. For example, the rotation speed of a motor can be reduced of 10% if the overload of motor is inferior to 20%.

To formalize the system's knowledge, we propose to use a state transition representation STRIPS (Stanford Research Institute Problem Solver) [11] where the system knowledge is represented as logical atoms derived from first-order logic. Each state corresponds then to a set of logical atoms (e.g., observations, diagnoses and system descriptions) that are either true or false within a certain degree of interpretation, the transitions are the control actions that change the truth values of these atoms.

In artificial intelligence, STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Richard Fikes and Nils Nilsson in 1971. The same name was later used to refer to the formal language of the inputs to this planner. This language is the base for most of the languages for expressing automated planning problem instances in use today. It is important to distinguish between the original STRIPS planner and the STRIPS representation language. Here, the STRIPS means the formal language. The STRIPS suggests a representation of planning problems, such as states, actions, and goals. The key is that STRIPS is expressive enough to describe a wide variety of problems, but restrictive enough to allow efficient algorithms to operate over it. In this selection, we first outline the basic representation language of classical planners, known as the STRIPS language.

For the convenience of the reader, we briefly review the relevant terminologies of propositional logic and first-order logic. Propositional Logic is the most basic branch of Mathematical Logic. The propositional logic is based on "propositions" which one can argue as being true or false. The propositions can be formed by combining atomic propositions using logical connectives. First-order logic is its extension by adding the notion of "predicate" which is a verb phrase template that describes a property of objects, or a relationship among objects represented by the variables.

A literal is a symbol used to represent a Boolean statement in logic that can take the value either true or false. The positive literals just take the value true. Ground and function-free literal means the simplest literal which is indivisible and not composite.

Representation of states: Planners decompose the world into logical conditions and represent a state as a conjunction of positive literals. The propositional literals and first-order literals can be used; for example, *at(Plane1, A)* and *at(Plane2, B)* might represent plane 1 is at the airport *A* and another plane 2 is at airport *B*. Literals in first-order state descriptions must be ground and function-free. Literals such as *at(f(x), y)* are not allowed. The closed-world assumption is used, meaning that any conditions that are not mentioned in a state are assumed false.

Representation of goals: A goal is a partially specified state, represented as a conjunction of positive ground literals. A propositional state "*s*" satisfies a goal "*g*" if "*s*" contains all the atoms in "*g* " (and possibly others).

Representation of actions: An action is specified in terms of the preconditions that must hold before it can be executed and the effects that ensue when it is executed. For example, an action for flying a plane *p* from one location *A* to another *B* is:

Action (*Fly (p, A, B*)

Precond: $at(p, A) \wedge Plane(p) \wedge Airport(A) \wedge Airport(B)$

Effect: $\neg at(p, A) \wedge at(p, B)$

at(p, A) means that the plane *p* is at the location *A*; *Plane(p)* means that the plane exists; *Airport(A)* and *Airport(B)* mean that the airport *A* and *B* are available.

This action schema consists of three parts:

(1). The action name and parameter list - for example, *Fly(p, A, B)* - serves to identify the action.

(2). The precondition is a conjunction of function-free positive literals stating what must be true in a state before the action can be executed.

(3). The effect is a conjunction of function-free literals describing how the state changes when the action is executed. A positive literal *P* in the effect is asserted to be true in the state resulting from the action, whereas a negative literal $\neg P$ is asserted to be false.

Definition 1 (The system model): Let *L = {P1, P2, ... Pn}* be a finite set of logical atoms. With this set, a system can be represented as a state-transition system. Formally, it is a 3-tuple $\Sigma = (S, A, \gamma)$, where:

(1) $S = \{s1, s2\ldots\} \subseteq 2^L$ is a finite or recursively enumerable set of states. Each state *s* is a subset of *L*. Intuitively, *s* tells us which logical atoms currently hold true. If $p \in s$, then *p* holds true in the state represented by *s*, and if $p \notin s$, then *p* does not hold true in the state represented by *s*.

(2) $A = \{a1, a2\ldots\}$ is a finite or recursively enumerable set of control actions. Each control action $a \in A$ is a

multiple of subsets of L, which can be expressed as a = (preconditions, determin-effects, nondetermin-effects). As the term suggests, the set of preconditions is called the preconditions of a, and the set of determin-effects (nondetermin-effects) represents the deterministic (resp. non-deterministic) effects of a. Non-deterministic effects represent an unknown, since it is not known which of them will actually take place. We assume that for any a, any nondeterministic effect is consistent with the deterministic effects *(i.e.*, no single atom and its negation exist simultaneously in the effect sets of a).

(3) γ: $S \times A \rightarrow 2^S$ is a state-transition function. $\gamma(s, a)$ = *determin-effects*(a) Υ *nondetermin-effects*(a) if $a \in$ A is applicable to $s \in S$, otherwise, $\gamma(s, a)$ is undefined. In other words, whenever an action is applied to a state, it produces another state. This is useful information because once A is known; S can be specified by giving just a few of its states.

It should be noted that γ is defined for one action but the action is defined non-deterministically as above. In other words, when an action is applied to a state, it produces one or several follow-up states. That is the reason why $\gamma(s, a)$ is introduced as a set in the definition of the model. In the procedure of reconfiguration, the actions defined by these logical atoms are used to generate the system model and then this model is checked state by state by model checking. The phase of generating system model is based on actions and the phase of model checking is based on states. A same set of logical atoms is used in these two different phases.

It also should be noted that an action a is always associated to an elementary component of the system. Elementary components are components directly connected to the process such as sensors and actuators. The action of sensor, "reading", is a special kind of action which is activated automatically and runs continuously. Actions are then combined to achieve system's functions or system's goals [12].

This model allows describing actions in terms of their preconditions and effects and describes the states as conjunctions of positive literals. The precondition states what must be true in a state before an action can be executed. The effect describes how the state changes when the action is executed. An action is 'applicable' in any state that satisfies the precondition; otherwise, the action has no effect.

The list of the possible states can be generated from the Cartesian product of the diagnosis atoms, the control-command atoms and the observation atoms. The initial state is one of these possible states and is captured by the diagnoser and the observer at the moment that the fault report is received. The interesting succeeding states are generated automatically by the algorithm presented in the next section.

III. The Algorithm Of Modeling

The system modeling algorithm is given in Algorithm 2.1. It performs a procedure close to iterative deepening, at each time of iteration discovering a new part of the state space. The algorithm returns an automaton (S, A, T, s_0) as the updated model, where "S" is the set of states, "A" is the set of available control actions, "T" is the automata's transition set, and "s_0" is the initial state directly identified from observations and diagnoses. The operator '-' (resp. '+') expresses that an element is subtracted from (resp. added to) a set of elements.

In the algorithm, "t" is an element of "T"; "*Old*" is the set of states that are checked at the current iteration, while "*New*" is the state set for next iteration; "*Com*" is the set of applicable control actions whose pre-conditions belong to the current checked state "s_{old}" (line 8).The notation "$\| S \|$" means the cardinality of set S, *i.e.*, the number of elements of the set S. "s_{old}" means one state of set "*Old*" (the set of states that are checked at the current iteration); "s_{new}" means a new state created by the transition $\gamma(s_{old}, a_{run})$. "a_{test}" means one action of set "A" (the set of available control actions). "a_{run}" means one action of set "*Com*" (the set of applicable control actions). "s_{reach}" means one state of set "*Reach*" (the set of states that can be reached from the state "s_{old}").

A set of states that can be reached from the state "s_{old}" is found (line 12) by applying the control actions "*Com*". Then, each of these reachable states is checked to see whether or not it is new to this automaton (line 15). If it is, then it will be added to the set "*New*" for next iteration (line 16-18). If there is no new state (line 2, 23), then the algorithm terminates.

```
Algorithm 1: Expand (S, A, T, s0)
1  S ← s0; Old ← s0; T ← Ø; j = 0;
2  While Old ≠ Ø do
3    { New = Ø;
4      For x=1 to ‖Old‖ do
5          s_old = Old[x];
                    /* Select a state of set Old in sequence */
6          For k=1 to ‖A‖ do
7              a_test = A[k];
                    /* Select an action of set A in sequence */
8              If (preconditon (a_test) ⊆ s_old) then Com=Com+ a_test;
9               End For
10              For i=1 to ‖Com‖ do
11             a_run = Com[i];
                    /* Select an action of set Com in sequence */
12             If s_new ∈ γ(s_old, a_run) then Reach = Reach+s_new;
13                  For r=1 to ‖Reach‖ do
14                s_reach = Reach[r];
                    /* Select a state of set Reach in sequence */
15                          If s_reach ∉ S then
16                              { s_j+1 = s_reach; S = S+ s_j+1;
17                                t_j+1 = (s_old, a_run, s_j+1); T = T+ t_j+1;
18                                New = New + s_j+1; j= j+1;}
19                    End for
20                    Reach = Ø;
21              End for
22     End for
23     Old = New; Com = Ø;
24 End While
```

This algorithm of generating system model is based on actions and its objective is to provide a flexible method of generating system model. Firstly, the basic system knowledge is divided into three different kinds of knowledge which complement each other. Secondly, all of the system knowledge can be represented by the first order logic which can be operated and handled easily and efficiently. Thirdly, the cause-effect information is integrated into the action schema by the representations of pre-conditions and effects. Therefore, the model possesses

the ability of automated reasoning which is the core of reconfiguration and automated generation of model.

As the limitations of the space, we ignore the example of application and some details.

IV. CONCLUSION

In this paper, there are two main advantages by using STRIPS to construct the system model. The first one is its powerful description. Although the first order logic is used in the classic STRIPS, using other powerful logics can make the STRIPS expressive enough to describe a wide variety of dynamic systems, such as the ones for maintaining a property, achieving a goal periodically or within a number of steps after the request was made, and achieving several goals in sequence. The second one is its powerful automated reasoning. The STRIPS can integrate the cause-effect knowledge and the automated reasoning mechanism into one model.

With these two advantages, the STRIPS model can achieve self-updating when the system condition has changed. For example, the STRIPS find an operator configuration that will transform a given initial model into one that satisfies a specific goal condition. Moreover, it allows building the model gradually with the information received instead of fully in one time.

Furthermore, a new area of study called process planning has emerged from the application of automated planning techniques to machining procedures ([13], [14], [15]). There are many representations for a control program for a manufacturing system like, for example, GRAFCET, Ladder or Petri nets.

Using automated planners to generate the plan for normal operations may not be so interesting, since this plan is rarely changed and a lot of time and effort may thus be spent on designing and optimizing this plan by human experts. However, automated planning is an interesting tool for error recovery. Since the process may end up in any one of a very large number of states after a break-down, it is not realistic to have recovery plans designed in advance for all such states. It is, thus, desirable to invoke a planner in the case of break-downs, which finds a plan that brings the process back into normal operation again. Such a plan must be correct, in order not to jam or even destroy the plant, and it must also be found within a reasonable time, since large-scale processes typically have very high stand still costs [16].

Finally, as the reconfiguration for autonomous systems fuses together researches from such diverse areas of AI as model-based reasoning, qualitative reasoning, planning and scheduling, execution, propositional satisfiability, concurrent reactive languages, Markov processes, model-based learning, and adaptive systems, the representation of the system model, a transition system, can be very easily extended by adding several variables to integrate these diverse researches above, as like [17], [18], [19], [20]，[21] and [22] did.

ACKNOWLEDGMENT

This work is supported by “The Nature Science Foundation of Tibet：13-38”, “A Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions (Coastal Development Conservancy)”, “Technology Foundation for Selected Overseas Chinese Scholar, Ministry of Personnel of China”, “the Fundamental Research Funds for the Central Universities”, and “he Scientific Research Foundation for the Returned Overseas Chinese Scholars, State Education Ministry”.

REFERENCES

[1] C. Ippolito, "Polymorphic Control Reconfiguration in an Autonomous UAV with UGV Collaboration," IEEE Aerospace. Conference, Big Sky, Montana USA, March 2008.

[2] R. O’Grady, A.L. Christensen, and M. Dorigo, "Autonomous Reconfiguration in a Self-assembling Multi-robot System," Lecture Notes in Computer Science, Ant Colony Optimization and Swarm Intelligence, Volume 5217, Springer, 2008.

[3] B. C. Williams, and P. Pandurang Nayak, "A Model-based Approach to Reactive Self-Configuring Systems," In Proceedings of the National Conference on Artificial Intelligence, 1996.

[4] D. Watson,"Model-based autonomy in deep space missions, " IEEE Intelligent Systems, Vol.18(3), 2003, pp.8-11.

[5] M.-G. Mehrabi, A.-G. Ulsoy, Y. Koren, and P. Heytler, "Trends and perspectives in flexible and reconfigurable manufacturing systems," Journal of Intelligent Manufacturing, Vol.13, 2002, pp.135–146.

[6] S. Saad, "The reconfiguration issues in manufacturing systems," Journal of Materials Processing Technology, Vol. 138(1,3), 2003, pp.277–283.

[7] F. Lamotte, P. Berruet, and J.-L. Philippe, "A model for the reconfiguration of manufacturing systems," In Proceedings of the 16th IFAC World Congress, Prague, 2005.

[8] B. Lussier, A. Lampe, R. Chatila, J. Guiochet, F. Ingrand, M.-O. Killijian, and D. Powell, "Fault Tolerance in Autonomous Systems How and How Much?," in 4th IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments, (Nagoya, Japan), 2005.

[9] B. C. Williams and P. Pandurang Nayak, "Immobile Robots: Artificial Intelligence in the New Millenium," Cover article of AI Magazine, Vol.17(3), 1996, pp.16-35.

[10] H.-X. Hu, A.-L. Gehin, M. Bayart, "A Formal Framework of Reconfigurable Control Based on Model Checking," in American Control Conference, Seattle, Washington, USA, 2008.

[11] Richard E. Fikes, and Nils J. Nilsson, “STRIPS: A new approach to the application of theorem proving to problem solving,” Artificial Intelligence, Vol. 2, Issues 3-4, Winter, 1971, pp. 189-208.

[12] H.-X. Hu, A.-L. Gehin, M. Bayart, "Model Aggregation for Reconfigurable Control Based on Generic Component Model," in ICSSSM’06, Troyes, France, 2006.

[13] Ayletta, R.S., Soutter, J., Petley, G.J., Chungc, P.W.H., and Edwards, D., "Planning plant operating procedures for chemical plant," Engineering Applications of Artificial Intelligence, Vol. 14, 2001, pp. 341–356.

[14] G. Stephanopoulos and C. Han, "Intelligent Systems in Process Engineering: A Review," Computers chem. Engng., Vol. 20, No. 6/7, 1996, pp. 143-191.

[15] A. Benveniste and Karl J. Astrom, "Meeting the Challenge of Computer Science in the Industrial Applications of Control: An Introductory Discussion to the Special Issue," IEEE Transactions On Automatic Control, Vol. 38, No. 7, JULY, 1993.

[16] I. Klein, P. Jonsson and C. Backstrom, "Efficient planning for a miniature assembly line," Artificial Intelligence in Engineering, Vol. 13, 1999, pp. 69-81.

[17] B. C. Williams, and V. Gupta, "Unifying Model-based and Reactive Programming within a Model-based Executive," In Proceedings of the International Workshop on Principles of Diagnosis (DX99) , Loch Awe, Scotland, 1999.

[18] B. C. Williams, S. Chung, V. Gupta, "Mode Estimation of Model-based Programs: Monitoring Systems with Complex Behavior," Proceedings of the International Joint Conference on Artificial Intelligence, Seattle, USA, 2001.

[19] P. Kim, B. C. Williams and M. Abramson, "Executing Reactive, Model-based Programs through Graph-based Temporal Planning," Proceedings of the International Joint Conference on Artificial Intelligence, Seattle, USA, 2001.

[20] M.Ingham, B.C. Williams, T. Lockhart, A. Oyake, M. Clarke and A. Aljabri, "Autonomous Sequencing and Model-based Fault Protection for Space Interferometry," International Symposium on Artificial Intelligence, Robotics and Automation in Space, Montreal, Canada, June, 2001.

[21] M. Hofbaur, and B.C. Williams, "Mode estimation of probabilistic hybrid systems," Hybrid Systems: Computation and Control, Lecture Notes in Computer Science (HSCC 2002), 2289, 2002, pp.253–266.

[22] A.-L. Gehin, H.-X. Hu, and M. Bayart, "A self-updating model for analysing system reconfigurability," Engineering Applications of Artificial Intelligence, Vol.25, Issue 1, 2012, pp.20-30.

[23] T. B. Sheridan, "Telerobotics, automation, and human supervisory control," MIT Press, 1992.