International Journal of Control and Automation
Vol.7, No.8 (2014), pp. 103-116
http://dx.doi.org/10.14257/ijca.2014.7.8.10

Plan Library and Conventions of Space Robot System

Cen Yu and WeiJia Zhou

Shenyang Institute of Automation Chinese Academy of Sciences, State key Laboratory of Robotics Nanta Street 114#, Shenyang, Liaoning Province, China, 110016E, 086-24-23970937
University of Chinese Academy of Sciences, Beijing 100049, CHN
yucen@sia.cn, zwj@sia.cn

Abstract

In space exploration, robots are important to do some hazardous jobs instead of human. The robots usually work in an unstructured environment to operate non-cooperative targets. There are unpredictable events which can reduce the capacity of robots. And unfortunately the communication channel between robots and ground command center has limited band-width and high latency. Therefore, space robot system needs plans that can autonomously deal with unpredictable events. Executable plans must satisfy some well defined conventions. This paper proposes a formal representation of plan library based on Petri net as well as the way to define conventions based on LTL. The formal representation formalizes concepts such as event, action and behavior. And plan is a behavior instance. This approach could be used to generate rational plans automatically or with a litter help of human. We illustrate them with a space robot application example.

Keywords: *Space robot system, discrete event system (DES), Plan generation, linear temporal logic (LTL), Reasoning service*

1. Introduction

Now the space robot researches have focused on in-space operation robots and planetary surface exploration robots [1]. The in-space operation robots assist shuttle docking and scientific experiment in satellites or space stations, such as ERA (European Robotic Arm) [2], Canadarm [3], and JEMRMS (Japan Experiment Module Remote Manipulator System) [4]. And the planetary surface exploration robots rove and sample on planetary surface, such as Sojourner [5].

These space robots have the following characteristics: (a) robots work in unstructured environments; (b) robots operate non-cooperative targets; (c) there are unpredictable events which can reduce robots capacity; (d) the communication channel between robots and ground command center has limited band-width and high latency. Since these characteristics, we need to generate plans by which space robot system deals with unpredictable events autonomously. Executable plans must satisfy some well defined conventions.

To formally define plans, a knowledge model is need to represent things about space robot system in a general case which means that it doesn't make too many assumptions on the specific robot system. Different knowledge models have been studied. The most popular on remains the rule-based model, for which different implementations have been proposed [6].The traditional means has inherent weakness in the process of knowledge representation and knowledge sharing. To solve this problem, ontology technology is used in the field of

ISSN: 2005-4297 IJCA

knowledge engineering. It is an ideal approach to construct knowledge case based on ontology in order to store and represent domain knowledge [7].

When a plan is mentioned, it must be defined on a control structure that models concurrency control strategy. Kripke structure [8] is widely used since its autonomous capability and excellent real-time responsiveness. Robot system has a complete control strategy when the Kripke structure has been fulfilled by a plan.

Owing to uncertainties in the execution, robot system should be modeled by DES (Discrete Event System) [9]. DES modeling method has the following kinds: Petri nets [10], Alphabet-based approaches [11], Perturbation methods [12], Control theoretic techniques [13], and Expert systems design [14]. A plan of DES defines when and which action should be enabled or disabled according to the happening events.

Earlier researches on plan representation tended to encapsulate functionality into behaviors as basic units to form plans. Since that, these control systems are called behavior-based control systems [15], such as ALLIANCE [16], CAMPOUT [17]. Behavior-based control systems have been used on mobile robot platforms to deal with loosely coordinated tasks. Self-learning mechanisms for behavior-based system optimize the parameters of behavior selection algorithm [18-20].

Current researches have achieved something such as modeling plan executing process [21-24] and modeling uncertainty in sensing [25, 26]. Existing works focus on planning strategy and process modeling, but the lack of formal representation of concepts in plan generation restricts their usage in space exploration.

There are some logic systems, such as Floyd-Hoare logic [27] and the temporal logic of actions [28], reason about algorithms of computer program. They assume that if the preconditions of an action are true and the action is enabled, then postconditions must be true when the action has been finished. This assumption is almost certainly false in the field of robotics. Robots would get different results according to unpredictable event in an action.

This paper is organized as follows: a general knowledge model for space robot system is introduced in Section 2; Section 3 proposes a formal representation of plan library based on Petri net; Section 4 designs a logical system to define conventions and decide whether a plan satisfies a set of convention; finally, we illustrate them with a space robot application example in Section 5 and give conclusion in Section 6.

2. General Knowledge Model for Space Robot System

To represent things in the real-world, knowledge is modeled by a conceptualization [29] defined as an ordered triple $C =< D, W, \Re >$, where D is a domain of discourse, W is a set of maximal states of affairs of such domain (also called possible worlds), and $\Re$ is a set of conceptual relations on the domain space $< D, W >$. A conceptual relation ρ^n of arity n on $< D, W >$ is a total function $\rho^n : W \to 2^{D^n}$ from W into the set of all n-ary (ordinary) relations on D. Authors have published a paper [30] for more details of the general knowledge model.

There are sophisticated logical tools for representing and reasoning the conceptualization of world knowledge, such as propositional logic and predicate logic. But unfortunately, the propositional logic has very limited power of expression and the predicate logic is undecidable. Due to the good qualities on expression power, decidability, and superiority in object-oriented classification, we employ the Description Logic [31], which is extended from first-order predicate logic.

Description Logic uses feature function f that maps things in domain D to individuals in symbolic domain Δ. A concept is a description which gathers the common properties among a collection of individuals. So that, a concept C is a subset of Δ, and a relation is an n-tuple of individuals in Δ.

With the Description Logic, knowledge model $\Sigma =< T, A >$ consists of TBox T and ABox A. The TBox contains intensional knowledge in the form of a terminology and is built through declarations that describe general properties of concepts. TBox is a finite set of subsumption assertion. The ABox contains extensional knowledge that is specific to the individuals in Δ. ABox is a finite set of instance assertion.

$ABox = ABox\_Common \cup ABox\_State$ can be divided into the set of common assertion and the set of state assertion, where $ABox\_Common \cap ABox\_State = \varnothing$. The values of assertions in $ABox\_Common$ are independent of possible world ω. Conversely, the values of assertions in $ABox\_State$ are dependent of possible world ω and $ABox\_State(\omega)$ is the set of assertion which is true in possible world ω. Before we have a set W of possible world, $ABox\_State = \varnothing$. The set W and $ABox\_State$ can be constructed through the method introduced by Section 4 when we have a plan.

A plan specifies the stages of execution (refer to possible worlds), the assertions which are true in a stage (refer to elements of $ABox\_State(\omega)$), and the translations from one stage to another. $Plan\_Lib$ is constructed for plan generating and $Convention\ s$ are properties that must be satisfied by a rational plan. As shown in Figure 1.

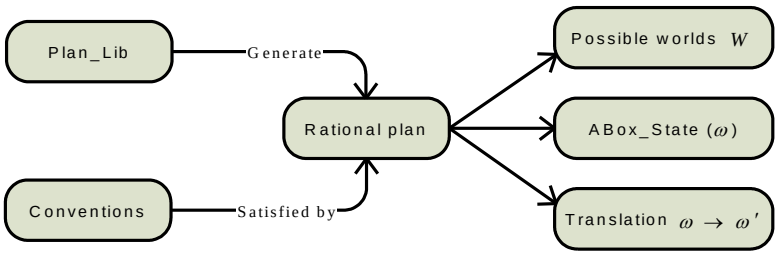

Figure 1. Rational Plan is Generated by Plan Library and Satisfies Conventions

Plans must be defined on a control structure which models concurrency control strategy. This paper employs Kripke structure [8] since its autonomous capability and excellent real-time responsiveness. The Kripke structure is a tuple $M = (S, R, L, S_0)$, where S is a non-empty finite set of state, R is a set of translation, $L: S \mapsto 2^{BA}$ assigns every state a set of basic assertion, and $S_0 \subseteq S$ is a set of initial states. Robot system has a complete control strategy when the control structure $M = (S, R, L, S_0)$ has been fulfilled by a plan.

The set of state which is active at a point of time is called S_{mark}, and $S_{mark} \subseteq S$. A trace of a plan is a sequence $\pi = S_{mark,0}, S_{mark,1} \cdots$, where $S_{mark,0} \subseteq S_0$. All possible traces of a rational plan must satisfy the properties defined in *Conventions*.

3. Plan Library

$Plan\_Lib =< Event\_Set, Action\_Set, Behavior\_Set >$ consists of a set of event, a set of action, and a set of behavior. Event is a set of assertions and, when we say the event happens, it means that all assertions in the set are true now. Action is an atomic functional interface.

Behavior is a control strategy that receives and sends events, enables and disables actions. Plan is a behavior instance generated from *Behavior_Set*. And the plan is rational if it satisfies *Conventions*. Relationships in plan library are shown in Figure 2.

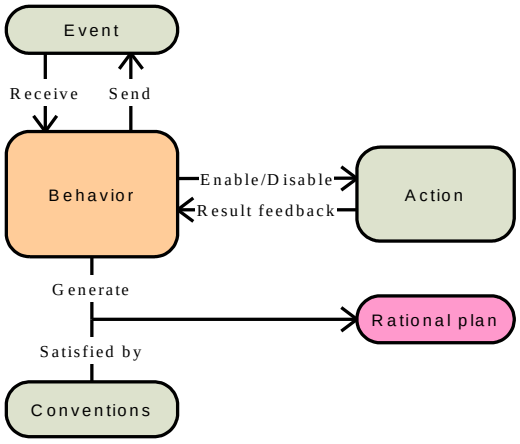

Figure 2. Relationships in Plan Library

3.1. Definitions of Plan Library

Definition 1: Event

An event $E(x_1,\ldots x_n)=\{Assertion_1,\cdots Aseertion_m\}$ is a set of assertion, where E the event name and variable individuals is $x_1,\ldots x_n$ are called event parameters. Its corresponding event parameter specification $Parameter_E=\{C_{E,1}(x_1),\ldots C_{E,n}(x_n)\}$ specifies the value scope of event parameters. An event instance $E(a_1,\ldots a_n)$ arises from event $E(x_1,\ldots x_n)$, where assertions $C_{E,1}(a_1),\ldots C_{E,n}(a_n)$ must be true.

$Event\_Set$ is a set of ordered tuple $\langle E(x_1,\ldots x_n), Parameter_E\rangle$.

Definition 2: Action

An action $A(x_1,\ldots x_n)=(Enable_A, Ready_A, Disable_A, Result_A)$ is an atomic functional interface, where A is the action name, variable individuals $x_1,\ldots x_n$ are called action parameters, $Enable_A$ is the event that action is enabled, $Ready_A$ is the event that action get ready, $Disable_A$ is the event that action is disabled, $Result_A$ is the set of result event. Its corresponding action parameter specification $Parameter_A=\{C_{A,1}(x_1),\ldots C_{A,n}(x_n)\}$ specifies the value scope of action parameters. An action instance $A(a_1,\ldots a_n)$ arises from $A(x_1,\ldots x_n)$, where assertions $C_{A,1}(a_1),\ldots C_{A,n}(a_n)$ must be true.

If the event $Enable_A$ happens and the action A has been initialized, then A will be enabled to wait $Ready_A$. The enabled A starts executing if $Ready_A$ happens. The executing A must be disabled immediately if $Disable_A$ happens. When A finishes, it sends an result event $R\in Result_A$.

In essence an action instance is a mapping from one initial state to several possible final states. It could be a calculation or a functional interface of actuator.

$Action\_Set$ is a set of ordered tuple $\langle A(x_1,\ldots x_n), Parameter_A\rangle$.

Definition 3: Behavior

A behavior $B(x_1,\ldots x_n)=(P_B, Terminate_B, Input_B, Output_B, E_B)$ is an action control strategy based on events, where B is the behavior name, variable individuals $x_1,\ldots x_n$ are called behavior parameters, P_B is the behavior precondition event, $Terminate_B$ is the behavior terminating event, $Input_B$ is the set of input event, $Output_B$ is the set of output event, and E_B is the set of postcondition event. Its corresponding behavior parameter specification $Parameter_B=\{C_{B,1}(x_1),\ldots C_{B,n}(x_n)\}$ specifies the value scope of parameters. A behavior instance $B(a_1,\ldots a_n)$ arises from $B(x_1,\ldots x_n)$, where $C_{B,1}(a_1),\ldots C_{B,n}(a_n)$ must be true.

If the event P_B happens and the behavior B has been initialized, then B starts executing. During the execution, B receives input events in $Input_B$ and sends output events in $Output_B$. The executing B must be terminated immediately if $Terminate_B$ happens. When B finishes, it sends a postcondition event $E \in E_B$.

In essence a behavior instance is a mapping from one initial state and several input event instances to several possible final states and output event instances. It involves branch selections, events generation, events response, and the clusters of behavior results.

$Behavior\_Set$ is a set of ordered tuple $\langle B(x_1,\ldots x_n), Parameter_B\rangle$. It is divided into two parts, including primitives and combinations:

$$Behavior\_Set = Primitives \cup Combinations$$

A primitive corresponds to an action and a combination is constructed by primitives. The internal structure of combination is called behavior structure.

3.2. Petri Net Model of Plan Library

Behavior structure is modeled by Pr/T Petri net with enabling arcs and inhibitory arcs. Each $Place_i$ of the Petri net is assigned a set $Assertion\_Set(Place_i)$ of assertions. These assertions are true when $Place_i$ has a token and vice versa.

As shown in Figure 3, a behavior structure is formed by several building blocks and a control block, where the building blocks have interfaces that could be connected through the control block. To be concise, we use combinations as building blocks to construct more complex behavior not just only using primitives.

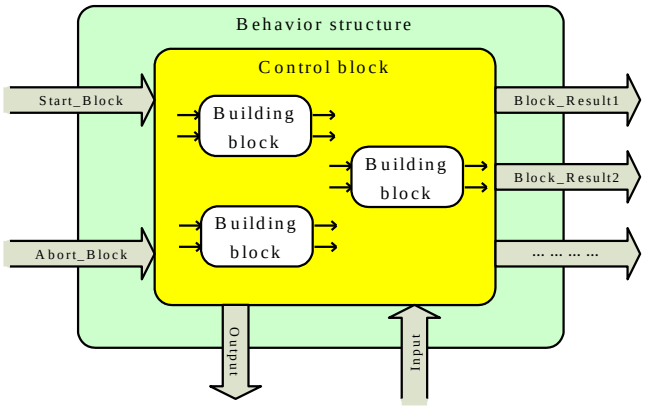

Figure 3. Behavior Structure

Primitive building block is defined in Section 3.2.1. Combination building block is defined in Section 3.2.2. And control block is defined in Section 3.2.3.

3.2.1 Primitive Building Block: A primitive building block corresponds to an action and its Petri net model is shown in Figure 4.

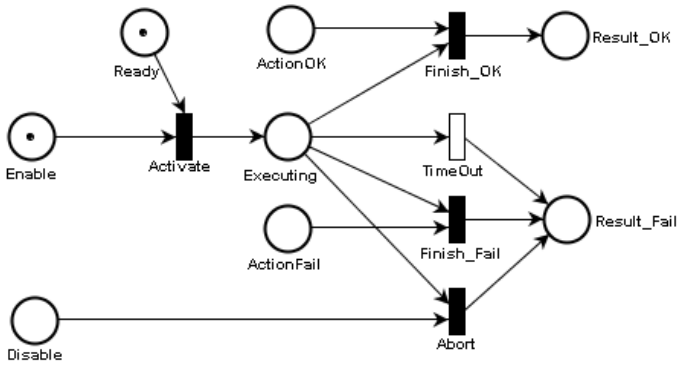

Figure 4. Primitive Building Block

Refer to the definition of action, correspondence between action and this Petri net model is shown as follows:

$$Action = (Enable_A, Ready_A, Disable_A, Result_A)\begin{cases} Enable_A = Enable \\ Ready_A = Ready \\ Disable_A = Disable \\ Result_A = \{Result\_OK, Result\_Fail\} \end{cases}$$

Refer to the definition of behavior, correspondence between primitive and this Petri net model is shown as follows:

$$Primitive = (P_B, Terminate_B, Input_B, Output_B, E_B)\begin{cases} P_B = \{Enable, Ready\} \\ Terminate_B = \{Disable\} \\ Input_B = \{Action\_OK, Action\_Fail\} \\ Output_B = \varnothing \\ E_B = \{Result\_OK, Result\_Fail\} \end{cases}$$

The primitive building block in Figure 4 has two result places, but it could have more result places in a general case.

3.2.2. Combination Building Block: The Petri net model of a combination building block is shown in Figure 5. The denotations $\{Input\}$ and $\{Output\}$ mean that the number of events could be 0 or a natural number.

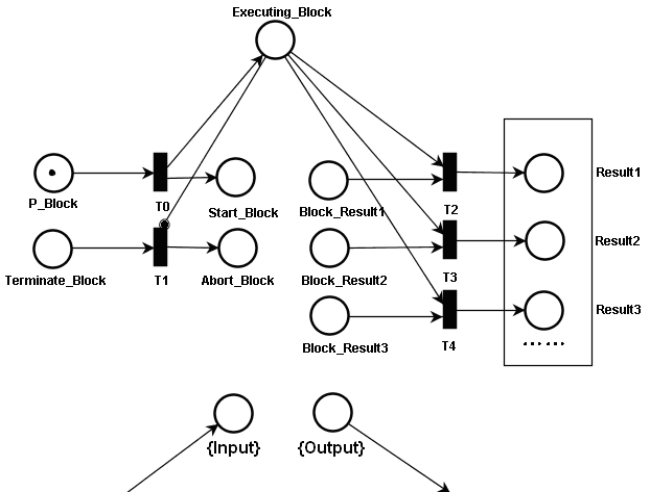

Figure 5. Combination Building Block

Refer to the definition of behavior, correspondence between combination and this Petri net model is shown as follows:

$$Combinatio\quad n = \left(P_{B}, Terminate\quad_{B}, Input\quad_{B}, Output\quad_{B}, E_{B}\right)\begin{cases} P = \{P\_Block\quad\} \\ Terminate\quad = \{Terminate\quad\_Block\quad\} \\ Input\quad = \{Input\quad\} \\ Output\quad = \{Output\quad\} \\ E = \{Result\quad 1, Result\quad 2, \cdots\} \end{cases}$$

The denotations $Start\quad\_Block\quad$, $Abort\quad\_Block\quad$, $Block\quad\_Result\quad 1, Block\quad\_Result\quad 2, \cdots$, $\{Input\quad\}$, and $\{Output\quad\}$ are interfaces that need to be connected to the interfaces of a control block.

3.2.3. Control Block: In a behavior structure, control block manages the procedure of building blocks through their interfaces. Here is a simple example of sequence control block, and there is another example in Section 5. Control block *Sequence (A, B)* is connected with two building blocks: *A* and *B*. If *A* fails then *B* would be skipped. Its Petri net model is shown in Figure 6.

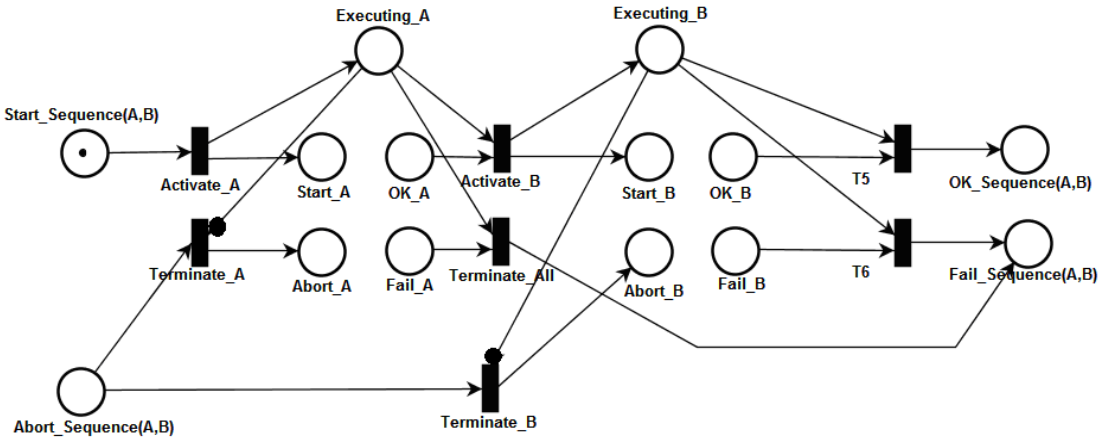

Figure 6. Control Block *Sequence (A, B)*

We can design other control blocks according to the involved building blocks and external events.

3.3. Initialization Process and Update Process of Plan Library

Before the initialization process of plan library, *Event_Set*, *Action_Set*, and *Behavior_Set* are all empty. It begins when we have the definitions of functional interfaces in the form of:

$A(x_1,\ldots x_n) = (Enable_A, Ready_A, Disable_A, Result_A)$, $Parameter_A = \{C_{A,1}(x_1),\ldots C_{A,n}(x_n)\}$.

Firstly, add the actions in the form of $\langle A(x_1,\ldots x_n), Parameter_A \rangle$ into $Action\_Set$.

Secondly, add the events in the form of $\langle Enable_A, Parameter_A \rangle$, $\langle Ready_A, Parameter_A \rangle$, $\langle Disable_A, Parameter_A \rangle$, $\langle Result_A, Parameter_A \rangle$ into $Event\_Set$.

Finally, define the Petri net models of primitives corresponding to the actions as shown in Section 3.2.1, and then add the primitives in the form of $\langle Primitive(x_1,\ldots x_n), Parameter_A \rangle$ into the set *Primitives*.

The initialization process of plan library is shown in Figure 7.

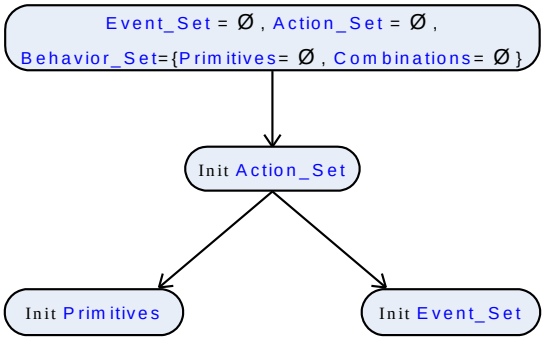

Figure 7. Initialization Process of Plan Library

Plan library need to be updated in two cases as follows:

(1) A functional interface has been added/deleted/changed. Then $Action\_Set$, $Primitives$, $Event\_Set$, and $Combinations$ must be updated as shown in Figure 8.

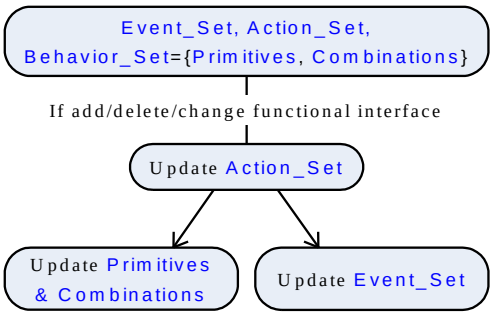

Figure 8. Update Process Caused by Functional Interface

(2) A combination has been added/deleted/changed. Then $Combinations$ and $Event\_Set$ must be updated as shown in Figure 9.

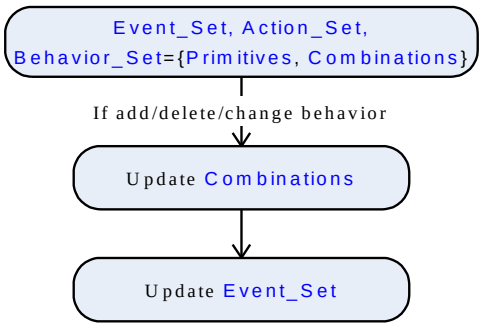

Figure 9. Update Process Caused by Combination

4. Conventions and Reasoning Services about Plan

A plan is a behavior instance. If robot system chooses to execute it, the set W of possible world and the set $ABox\_State$ of state assertion could be defined according to the Petri net model. Suppose that the set of marking place in possible world ω is $\{Place_1, Place_2, \cdots Place_m\}$, then the subset of $ABox\_State$ in which all assertions are true in possible world ω is defined as follows:

$$ABox\_State(\omega) = \bigcup_{i=1}^{m} Assertion\_Set(Place_i)$$

Each time the plan executes, it generates a sequence $\pi = \omega_0, \omega_1, \cdots$ that is called a trace of the plan. Reasoning services about plan decide whether all possible traces of a plan satisfy the properties defined in $Convention\ s$ or not. If the answer is true, we say the plan is rational. Otherwise, the plan is irrational.

Since the traces are represented by a discrete-time structure, their properties in $Convention\ s$ are obviously defined by the temporal logic system for discrete-time model. Fortunately, there are well developed temporal logic systems for discrete-time model, such as CTL* [32] or one of its sublogics, CTL (computation tree logic, Clarke *et al.*, 1986) [33] or LTL (linear temporal logic, Pnuelli 1981) [34]. The advantages of the LTL have been analyzed in [35]. Because of its expressiveness, compositionality, uniformity, and bounded model checking, we choose LTL as the temporal logic system about plan.

In the LTL, formulas are composed from the set of atomic propositions using the usual Description Logic connectives as well as the temporal connective G ("always"), F ("eventually"), X ("next"), and U ("until"):

- for all $p \in ABox\_State$, p is a state formula
- $p \in ABox\_State(\omega_i)$ is recorded as $\pi, i \models p$
- if φ and ψ are state formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, and $\varphi \vee \psi$
- any state formula φ is also a path formula
- if φ and ψ are path formulas, then so are $\neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $X\ \varphi$, and $\varphi\ U\ \psi$
- $\pi \models \varphi$, for any state formula φ , if and only if $\pi, 0 \models \varphi$
- $\pi \models \neg\varphi$ if and only if $\pi \not\models \varphi$
- $\pi \models \varphi \wedge \psi$ if and only if $\pi \models \varphi$ and $\varphi \models \psi$
- $\pi \models \varphi \vee \psi$ if and only if $\pi \models \varphi$ or $\pi \models \psi$

- $\pi \models \varphi \cup \psi$ if and only if, for some $i \geq 0$, $\pi, i \models \psi$ and $\pi, j \models \varphi$ for all $0 \leq j < i$
- $\pi \models \times \varphi$ if and only if $\pi, 1 \models \varphi$

Conventions are LTL path formulas that must be satisfied by each possible trace of a rational plan. Such as safety property $\pi \models \neg (true \cup error)$ that means $error$ will never happen, and liveness property $\pi \models true \cup necessary$ that means $necessary$ must happen.

Reasoning services about plan decide whether a given plan P satisfies a LTL path formula φ. One of the most efficient algorithms for LTL properties checking is the automata theoretic approach (Lichtenstein and Pnueli 1985; Vardi and Wolper 1986) [36, 37].

In the automata theoretic approach, we build automata $B_{\neg\varphi}$ which accepts all traces for which $\neg\varphi$ holds. The asynchronous product of P and $B_{\neg\varphi}$, $Asyn$, is constructed. If there are no accepting runs of $Asyn$, then P satisfies the LTL path formula φ.

For an irrational plan, the automata theoretic approach will give the trace which violates conventions. This trace will be used as a reference for refining.

5. Application Example

It has been used in the prototype of scientific experiment assistant robot system on space station, which is developed by Shenyang Institute of Automation Chinese Academy of Sciences, State key Laboratory of Robotics, Department of Space Automation Technologies and Systems.

The robotic arm is a part of the prototype. It consists of six joints and an end-effector. There is a human operator interface for the robotic arm to input position information and feed visual images back. Each one of joints has its own embedded controller, and these embedded controllers are connected with a master controller through communication buses. As shown in Figure 10.

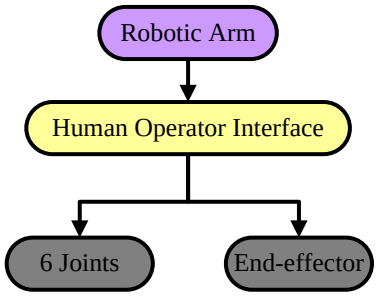

Figure 10. The Example Robot System

Each robot controller provides several functional interfaces with variable parameters. From the perspective of plan library, these interfaces are actions. For example, one of the functional interfaces of a joint is called *rotate* with four parameters: *joint_id*, *maximal speed*, *maximal torque*, and *target angle*. It's corresponding action of robotic arm is *Rotate*(*joint_id, max_speed, max_torque, target_angle*) with a parameter specification. Then we add the action to the plan library of robotic arm. The mapping from the set of primitive behavior to the set of action is bijective as shown in Section 3.2.1.

The set of primitive behavior is used as building blocks to construct complex behavior structures. Here is a representative example. Assume that there are two primitive behaviors of robotic arm: one is the previously introduced *Rotate* which rotates a special joint to target angle, and another is *Calc_inv* which calculates the inverse kinematics to

translate end position of robotic arm into angles of all joints.

To generate a plan for the servo control of robotic arm, we design a complex behavior *Sync_motion* with seven time-dependent parameters $(x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t, \Delta_t)$ which makes that the end position of robotic arm follows an existing position-time curve or the real-time input from human operator interface, where $< x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t >$ is the target position in the next time interval Δ_t. If the behavior doesn't care about an end position parameter, for instance, α_t, then we writes $\alpha_t = nil$.

When *Calc_inv* detects that the parameters have been changed, it calculates the target angles of joints, and then the control block of *Sync_motion* enables *Rotate* to make that joints rotate to target angles in parallel. If *Calc_inv* quits with the result *End_calc*, then the behavior structure of *Sync_motion* will quit with the result OK. Petri net model of *Sync_motion* is shown in Figure 11. It's worth noting that a behavior enables and disables actions rather than calling them.

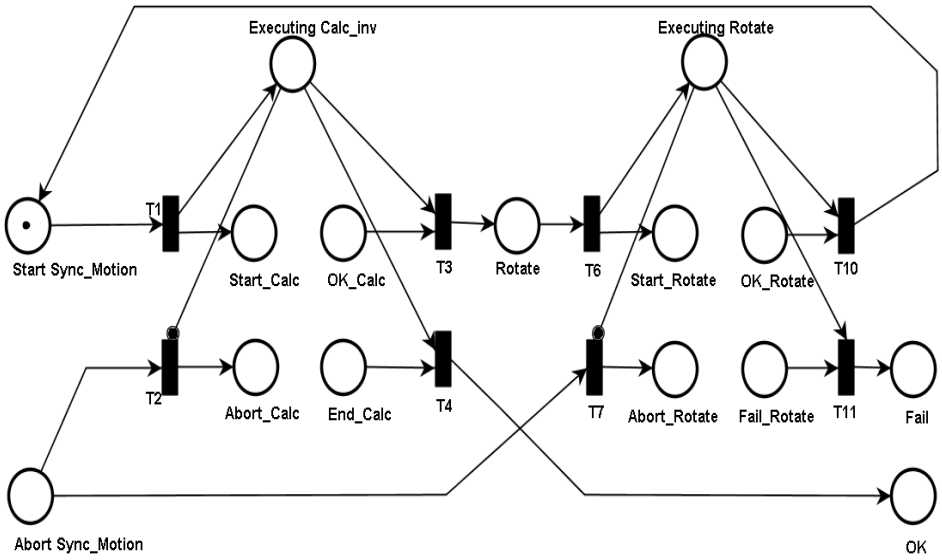

Figure 11. Petri Net Model of Behavior *Sync_motion*

In the application, joints would delay several time intervals to rotate. So they can do some motional optimization to run smoothly. Since these time intervals are short (for instance, less than 10ms), this approach is acceptable and very effective.

6. Conclusion

This paper proposes a formal representation of plan library based on Petri net. Additionally, we design a logical system based on LTL to define conventions and decide whether a plan satisfies a set of convention. This approach doesn't make too many assumptions on the specific robot system, so it can be widely used. With a space robot application example, we illustrate that the plan library is flexible and extensible. On the downside, we need experts to generate plan from a plan library for a given task. More future works are necessary to design a mechanism for assisting an abecedarian to generate plans or automatically generating plans.

References

[1] L. Pedersen, D. Kortenkamp, D. Wettergreen, I. Nourbakhsh and T. Smith, "Space robotics technology assessment report", National Aeronautics and Space Administration Ames Research Center Moffett Field, California, **(2002)** May, pp. 94035-1000.
[2] R. Boumans and C. Heemskerk, "The European Robotic Arm for the International Space Station", Robotics and Autonomous Systems. 1, vol. 23, **(1998)**.
[3] B. A. Aikenhead, R. G. Daniell and F. M. Davis. "Canadarm and the Space Shuttle", Journal of Vacuum Science & Technology A: Vacuum, Surfaces, and Films. 1, vol. 2, **(1983)**.
[4] D. King, "Space Servicing: Past, Present and Future", Proceeding of the 6th International Symposium on Artificial Intelligence and Robotics & Automation in Space, St-Hubert, Quebec, Canada, **(2001)**.
[5] C. R. Weisbin, D. Lavery and G. Rodriguez, "Robots in space into the 21st century", Industrial Robot. 2, vol. 24, **(1997)**.
[6] F. Rechenmann, "Declarative and Procedural Object-based Knowledge Modeling", Proceedings of IEEE Systems Man and Cybernetics Conference, **(1993)**.
[7] C. J. Hao, "Research on Knowledge Model for Ontology-Based Knowledge Base", Business Computing and Global Informatization (BCGIN), **(2011)**.
[8] E. Clarke, O. Grumberg and D. A. Peled, "Model Checking", The MIT Press, **(2000)**.
[9] C. G. Cassandras. "Discrete Event Systems: Modeling and Performance Analysis", Aksen Assoc. Inc, **(1993)**.
[10] A. A. Desrochers, "Modeling and Control of Automated Manufacturing Systems", IEEE Computer Society Press, **(1990)**.
[11] P. J. Ramadgae and W. M. Wonham, "The Control of Discrete Event Systems", Proceedings of the IEEE, **(1989)**.
[12] X. Cao and Y. C. Ho. "Models of Discrete Event Dynamic Systems", Control Systems Magazine, vol. 4, no. 10, **(1990)**.
[13] P. R. Kumar and S. P. Meyn, "Stability of Queuing Networks and Scheduling Policies", Proceedings of IEEE Decision and Control, **(1993)**.
[14] C. L. P. Chen and C. Wichman, "A CLIPS Rule-Based Planning System for Mechanical Assembly", Proceedings of NSF DMS Conf, Atlanta, **(1992)**.
[15] M. J. Mataric, "Behavior-based Control: Main Properties and Implications", Proceedings of IEEE International Conference on Robotics and Automation, Workshop on Architectures for Intelligent Control Systems, **(1992)**.
[16] L. E. Parker, "ALLIANCE: An Architecture for Fault Tolerant, Cooperative Control of Heterogeneous Mobile Robots", Proceedings of IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS), **(1994)**.
[17] P. Pirjanian, T. L. Huntsberger and A. Barrett, "Representation and Execution of Plan Sequences for Multi-Agent Systems", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, **(2001)**.
[18] L. E. Parker, "L-ALLIANCE: Task-Oriented Multi-Robot Learning In Behavior-Based Systems", Advanced Robotics, Special Issue on Selected Papers from IROS'96, **(1997)**.
[19] M. D. Bugajska, A. C. Schultz, J. G. Trafton, M. Taylor and F. E. Mintz, "A Hybrid Cognitive-Reactive Multi-Agent Controller", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, **(2002)**.
[20] E. Daglarli and H. Temeltas, "Artificial Behavioral System by Sensor-Motor Mapping Strategy for Multi-Objective Robot Tasks", Proceedings of Third International Conference on Natural Computation, **(2007)**.
[21] G. Yasuda, "A Distributed Autonomous Control Architecture for Synchronization and Coordination of Multiple Robot Systems", Proceedings of SICE Annual Conference, **(2012)**.
[22] G. Yasuda, "Hierarchical and Distributed Implementation of Synchronization and Coordination for Discrete Event Multiple Robot Systems", Proceedings of IEEE/ASME International Conference on Mechatronics and Embedded Systems and Applications (MESA), **(2012)**.
[23] Y. Zhang and L. E. Parker, "Task Allocation with Executable Coalitions in Multirobot Tasks", Proceedings of IEEE International Conference on Robotics and Automation (ICRA), **(2012)**.
[24] N. Palomeras, P. Ridao, M. Carreras and C. Silvestre, "Using Petri Nets to Specify and Execute Missions for Autonomous Underwater Vehicles", Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, **(2009)**.

[25] Y. Zhang and L. E. Parker, "IQ-ASyMTRe: Synthesizing Coalition Formation and Execution for Tightly-Coupled Multirobot Tasks" Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), **(2010)**.
[26] A. Jayasiri, G. K. I. Mann and R. G. Cosine, "A General Architecture for Decentralized Supervisory Control of Fuzzy Discrete Event Systems", Proceedings of American Control Conference (ACC), **(2012)**.
[27] S. L. Bloom and Z. Ésik, "Floyd-Hoare Logic", Iteration Theories. Springer Berlin Heidelberg, **(1993)**.
[28] L. Lamportm, "The Temporal Logic of Actions", ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 3, no. 16, **(1994)**.
[29] N. Guarino, "Formal Ontology and Information Systems", Proceedings of the 1st Int'l Conf on Formal Ontology in Information Systems, Trento, Italy, **(1998)**.
[30] C. Yu and W. J. Zhou, "Knowledge Interchange in Task-Oriented Architecture", For Space Robot Application. TELKOMNIKA Indonesian Journal of Electrical Engineering, vol. 4, no. 11, **(2013)**.
[31] F. Baader, "The Description Logic Handbook", Theory, Implementation, and Applications, Cambridge University Press, **(2003)**.
[32] T. Hafer and W. Thomas, "Computation Tree Logic CTL* and Path Quantifiers in the Monadic Theory of the Binary Tree", Automata, Languages and Programming, Springer Berlin Heidelberg, **(1987)**.
[33] E. M. Clarke, E. A. Emerson and A. P. Sistla. "Automatic Verification of Finite-state Concurrent Systems Using Temporal Logic Specifications", ACM Transactions on Programming Languages and Systems (TOPLAS), vol. 2, no. 8, **(1986)**.
[34] A. Pnueli, "The Temporal Semantics of Concurrent Programs", Theoretical Computer Science, vol. 1, no. 13, **(1981)**.
[35] A. Miller, A. Donaldson and M. Calder, "Symmetry in Temporal Logic Model Checking", ACM Computing Surveys (CSUR), vol. 3, no. 38, **(2006)**.
[36] O. Lichtenstein and A. Pnueli, "Checking that Finite State Concurrent Programs Satisfy Their Linear Specification", Proceedings of the 12th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, **(1985)**.
[37] M. Y. Vardi and P. Wolper, "An Automata-theoretic Approach to Automatic Program Verification", Proceedings of the First Symposium on Logic in Computer Science, IEEE Computer Society, **(1986)**.