Formal Modeling and Verification of Dynamic Reconfiguration of Autonomous Robotics Systems

Yujian Fu[1], **Mebougna Drabo**[2]
[1]Department of Electrical Engineering & Computer Science, Alabama A&M University, Normal AL, U.S.A.
[2]Department of Mechanical Engineering, Alabama A&M University, Normal AL, U.S.A.

Abstract— *Dynamic reconfiguration refers to the ability of a system to dynamically change its structure and interface according to different situations. It provides feasible and flexible modeling and simulation environments with powerful modeling capability and the extra flexibility to design and analyze robotics systems. The aim of this work is the modeling and verification of autonomous robotics systems (ARMs) subject to dynamic changes using extensions of Petri nets. It's been one of the research axes of using Petri nets to model reconfigurable systems, where structure changes during runtime, especially in the high level Petri Net domain. Numerous formalisms with different particularities have been proposed. These formalisms try to deal with some aspects of these systems. In this paper, we presented a new, generic and expressive approach to the dynamic reconfiguration on the extension of Predicate Transition Nets, called Predicate Transition Reconfigurable Nets (PrTR Nets). This approach allows the dynamic changes of net structure and provides the flexibility of semantic analysis on the reconfiguration analysis of the net. The formal definition of the PrTR nets formalism will be discussed, and a case study on a humanoid robotic system will be studied. Analysis and verification of the motion scenario will be discussed and related issues will be discussed.*

Keywords: Reconfiguration, Humanoid robotics, Predicate transition nets

1. Introduction

Dynamic reconfiguration refers to the ability of a system to dynamically change its structure and interface according to different situations. It provides feasible and flexible modeling and simulation environments with powerful modeling capability and the extra flexibility to design and analyze complex systems. In addition, the goals of dynamic behaviors of autonomous robots makes them ideally suited for numerous environments with challenging terrain or unknown surroundings. Applications are apparent in exploration, search and rescue, and even medical settings where several specialized tools are required for a single task. The ability to efficiently reconfigure between the many structures and behaviors of which an autonomous robot is capable is critical to fulfilling this potential. Thus, it is important for making autonomous robotics systems (ARSs) adaptable to changes is one of the main challenge in the autonomous robotics systems.

Moreover, considering the correctness of change and ensuring the appropriate behavior during reconfiguration, it will be more important that the mechanism for change be explicitly represented into the model so that at each stage of product development, designers can experiment the effect of structural changes, by prototypes or verification tools. This means that the structural and behavioral changes are taken into account from the very beginning of the design process rather than handling by an external and global system, e.g. some exception handling mechanism, designed and added to the model describing the system normal behavior. Thus we favour and internal and incremental over an external and uniform description of changes, and a local over a global handling of changes. This approach is compatible with the bottom-up modular synthesis of Petri Nets where a complex system is derived from successive refinements of places or transitions by sub-systems.

The remainder of the paper is organized as follows: Section 2 gives related works of reconfiguration using Petri nets and some other formal methods. Section 3 introduces the formal representation of PrTRN model for autonomous systems. Section 4 describes our approach for constructing and analyzing models of a humanoid robot in the pressing button scenario. Section 5 discusses the result and concludes the paper.

2. Related Works

In this section, we will focus on the related works of reconfiguration of the autonomous systems using Petri Nets. Other than that, the reconfiguration specification using other formal methods will be discussed also.

Several research works have been in the specification of reconfiguration using Petri Nets. Valk's self-modifying nets [18], [19] is considered as an early attempt for an extension of Petri net model with a built-in mechanism for handling changes. The changes are captured by two fundamental functions precondition and post condition. The notion of systems of replacement of matrices takes the place of the notion of systems of replacement/addition of vectors that characterize Petri nets.

In the work of [4], [3], [16], a class of high level Petri nets, called reconfigurable nets, is defined. Reconfigurable

nets can dynamically modify their own structure by rewriting some of their net structure that described by rewriting rules associated with the transition. A reconfigurable net is a Petri net with local structural modifying rules performing the replacement of one of its subnets by another subnet. The tokens in a deleted place are transferred to a created one. These nets were used for modeling dynamic changes within workflow systems as in [8]. It was shown that boundedness of a reconfigurable net can be decided by constructing a simplified form of Karp and Miller's coverability tree, however this construction did not allow to decide whether a given place of the net is bounded. However, the reconfiguration only take place in consideration of the structural changes. The rewriting rules that associated with the transition cannot be executed simultaneously with regular transition firing – each transition has two exclusive firing conditions.

On top of the work of [14], [15], to overcome the drawback of reconfigure transition, a Flexible nets was proposed in [13]. Reconfiguration of the structure of the net is interpreted as an operation that manipulates this structure by manipulating its components which are signed objects. The presence of a positive object (resp. negative object) in some place can be a reason to add (resp. delete) this object to (resp. from) the structure of this net. The formalism proposed is called Flexible Nets and reflects the idea that the model has a dynamic structure. This structure can be expanded, shrunken, or destroyed.

In [23], the authors proposed PrN (Predicate/Transition nets) to model mobility. The main concepts of of mobility is introduced as an agent. The agent space is composed of a mobility environment and a set of connector nets that bind mobile agents to mobility environment. Agents are modeled through tokens. So these agents are transferred by transition firing from a mobility environment to another. The structure of the net is not changed and mobility is modeled implicitly through the dynamic of the net.

In [17], authors proposed MSPN (Mobile synchronous Petri net) as formalism to model mobile systems and security aspects. They introduced notions of nets (an entity) and disjoint locations to explicit mobility. A system is composed of set of localities that can contain nets. To explicit mobility, specific transitions (called autonomous) are introduced. Two kinds of autonomous transition were proposed: new and go. Firing a go transition move the net form its locality towards another locality. The destination locality is given through a token in an input place of the go transition. Mobile Petri nets (MPN) [2], [1] extended colored Petri nets to model mobility. MPN is based on Ãř-calculus and join calculus. Mobility is modeled implicitly, by considering names of places as tokens. A transition can consumes some names (places) and produce other names. The idea is inherited from Ãř-calculus where names (gates) are exchanged between communicating process. MPN are extended to Dynamic Petri Net (DPN) [2]. In DPN, mobility is modeled explicitly, by adding subnets when transitions are fired. In their presentation [2], no explicit graphic representation has been exposed.

In [5], authors studied equivalence between the join calculus [9] (a simple version of π-calculus) and different kinds of high level nets. They used "reconfigurable netâĂİ concept with a different semantic from the formalism presented in this work. In reconfigurable nets, the structure of the net is not explicitly changed. No places or transitions are added in runtime. The key difference with colored Petri nets is that firing transition can change names of output places. Names of places can figure as weight of output arcs. This formalism is proposed to model nets with fixed components but where connectivity can be changed over time. Label associated is used in the reconfigure transition in "Labeled Reconfigurable NetsâĂİ [14], witch gives information about mobility.

In nest nets [21], tokens can be Petri nets. This model allows some transition when they are fired to create new nets in the output places. Nest nets can be viewed as hierarchic nets where we have different levels of details. Places can contain nets that their places can also contain other nets. So all nets created when a transition is fired are contained in a place. So the created nets are not in the same level with the first net. This formalism is proposed to adaptive workflow systems.

The power of Petri nets resides in its verification methods. To ensure verification of high level Petri nets, some works were proposed. In [5], author proved the equivalence between Reconfigurable nets and the join calculus. Reconfigurable nets can be interpreted in join calculus and so can be verified. In [6], P/ω nets are translated into linear logic programming. Author of [17], encoded Synchronous mobile nets in rewriting logic; they can use Maude to verify specifications. In this paper, we have first way through simulating the net or drawing automatically its reachability tree. The second way that requires more development in future papers, consists of the unfolding of the flexible nets into the Dynamic Nets. These last one can be transformed into CPN.

In this work, we attempt to provide a formal and graphical model for dynamic reconfiguration of robotic systems. We extended Predicate Transition nets with reconfigure guard function that is associated with (reconfigurable) transitions that are enabled when reconfiguration. Change of behavior is modeled explicitly by the possibility of adding or deleting nets which can be single node (transitions or places). Modification can happen when the reconfiguration function is satisfied and tokens available in the reconfigurable place. This allows different type of mobilities of the system behavior changes. The system is described by component based architecture, therefore, the mobility and bindings to resources can be modeled by output from the different components. We have introduce new sorts and operations on the new sorts to specify the mobility and compatibility of system changes. The calculus of system changes and

decision making will be modeled in separate components. The concept of component based model of autonomous behavior is inspired by SAM model [20], [22] that must compute this type of information.

The proposed formal specification approach should include all the regular motions, ensure the correctness of these normal behavior, describe the reconfiguration and ensure the correctness of the reconfiguration by guaranteeing the motion will be continued and finished successfully.

3. Modeling Dynamic Reconfiguration of Autonomous Robotics Systems

In this section, we will present the Predicate Transition Reconfigurable Nets (PrTR Nets) and Analysis of the PrTRN nets. PrTRN nets are an extension on the Predicate Transition Nets (PrT nets) with reconfiguration functions.

3.1 Predicate Transition Reconfigurable Nets (PrTRN Nets)

In the PrTR nets, we consider following three aspects of extension – net structure, signature, and net inscription (mainly guard). The dynamic semantics and state system will be discussed later based on the above extension. To allow the changes of place and transition, two special sorts Pl and Tr are defined, representing types of place and transition respectively. An instance of the sort Pl can be a place in a PrTR net, or can be a closed PrTR subnet that starts and ends with places. Similarly, An instance of the sort Tr can be a transition in a PrTR net, or can be an open PrTR subnet that starts and ends with transitions. A super sort PNS is defined as compatible with these two sorts Pl and Tr to facilitate the net refinement and substitution.

In the traditional PrT nets, the operations on the regular sorts can be regular operations including arithmetic operations, relational and logic operations, and set operations. Considering the new sorts are the set of regulars sorts and new sorts, we define operations of the sort of place and transition as following:

1) The arithmetic operations on the new sorts are not allowed.
2) The set operations can be the same on the new sorts provided that treats the new sorts as set.
3) The substitution is defined on the new sorts Pl, Tr and PNS.

Definition 1 (Predicate Transition Reconfigurable Nets): Precisely, a Predicate Transition Reconfigurable Nets (PrTR Nets) is an extension on the traditional Predicate Transition Nets (PrT Nets) [12], [11] and is defined by a tuple$(P^R, T^R, F, \Sigma^R, Eq^R, \varphi^R, L, G, G^R, M_0)$, where:

1) P^R is a finite set of places that includes reconfigurable place, T^R is a finite set of transitions ($P^R \cap T^R = \Phi, P^R \cup T^R \neq \Phi$), and F is a set of arcs or flow relations between each pair of P and T, e.g. $F \subseteq (P^R \times T^R) \cup (T^R \times P^R)$. The tuple (P, T, F) forms a basic Petri net structure.
   - A reconfigurable place is nothing more than a regular place but is used to hold the specific tokens (such as those with sorts Pl, Tr and/or PNS). Graphically, a reconfigurable place is denoted by a double line circle.
   - A reconfigurable transition is similar as regular transition but the is enabled and fired by reconfigurable guard G^R. A reconfigurable transition is graphically defined by a double lined box.
2) $\Sigma =< St, Op >$ consists of some sorts (St) of constants together with set of operations (Op) and relations on the sorts. We define three special sorts PNS, Pl and Tr represent PrT net system, place and transition. The operation on the regular sorts are still same. In addition, we define two new operations as follows:
   - subsort (denoted by $\preccurlyeq$) relation is defined as: $Pl \preccurlyeq PNS$, $Tr \preccurlyeq PNS$, and $\neg Pl \preccurlyeq Tr$ and $\neg Tr \preccurlyeq Pl$.
   - Θ will return the sort of any variables, terms and expressions defined on the place, arc and transition, i.e., $Op = Op \cup \Theta$ or $\Theta \in Op$.
   - $\curlywedge$ is a merge operation defined on the sorts St such that
     - $\forall v_1, v_2 \in Var$ (where Var is the set of variables). $\curlywedge$ can be performed iff $\Theta(v_1) = \Theta(v_2)$.
     - $\curlywedge$: Var_1, ..., $Var_n \rightarrow \Theta(Var)$ where $i \in [1..n]$ and $Var = Var_1 \cup ... \cup Var_n$.
   - A dual operation $\curlyvee$ is a split operation defined on the sorts St such that
     - $\forall v \in Var$ (where Var is the set of variables). $\curlyvee$ can be performed iff $\Theta(v_1)$ returns a multiple set, i.e., the capacity of $\Theta(v_1)$, $|\Theta(v_1)| > 1$.
     - $\curlyvee$: $Var \rightarrow \Theta$ (Var_1, ..., Var_n) where $i \in [1..n]$ and $Var = Var_1 \cup ... \cup Var_n$.
     - $\curlyvee$ is a reverse operation of $\curlywedge$, and vice versa.
3) Eq defines the meanings and properties of operations in OP.
4) $\varphi^R : P \rightarrow St$ is a relation associated each place p in P with a subset of sorts.
5) L is a labelling function on places, transitions, arcs, and variables. Given a place $p \in P$ or a transition $t \in T$, $L(p)$ returns the name of place p, $L(t)$ returns the name of transition t. Given an arc $f \in F$, the labelling function of f, L(f), is a set of labels associated with the arc f, which are tuples of constants ($CONs$) and variables (X), which is best described by L($f, Terms_{\Sigma,X}$). We use $Terms_{\Sigma,X}$ represents the expressions on the label of arc f. We use L($Terms_{\Sigma,X}$) to represent L($f, Terms_{\Sigma,X}$) when

there is no confusion in context. If $f \notin F$, $L(f) = \Phi$.

6) G is a mapping from transitions to a set of inscription formulae. The inscription on transition $t \in T$, $R(t)$, is a logical formula built from variables and the constants, operations, and relations in structure Σ; variables occurring free in a formula have to occur at an adjacent input arc of the transition.
7) G^R is a reconfigurable mapping function defined on the reconfigurable transitions and returns a set of inscription formulae. The reconfigurable function G^R is boolean function that defined on the evaluation of the variables in the reconfigure place. After evaluating the variable of reconfiguration field, the reconfigurable function G^R will output the corresponding token which is a subnet that the system is supposed to take.
8) M_0 is the initial or current marking with respect to sort, which assigns a multi-set of tokens to each place p in P with the same sort, $M_0 : P \rightarrow MCONs$.

In the above definition, a reconfigure place is nothing more than a regular place with sorts and holds tokens. The sorts of reconfigure place is defined as a tuple of $< reconfig, PNS >$, where $reconfig$ is a status that is set when the reconfiguration is needed, PNS is the PrTR subnet systems that described by the above definition. The subsort relation ($\preccurlyeq$) defines an implicit compatible sorts – any token with the sort of PNS can be used to substitute the token with the sort of Pl or Tr. Any subnet of PrTR net N can have sort, i.e., $\varphi(N) = \cup_i(\varphi(p_I) \cup \varphi(p_O))$, where p_I is the set of all input places and p_O is the set of all outgoing places. On top of the above, we define following compatible relation:

- Given any two subnets N_1 and N_2, we can say the sorts of two subnets N_1 and N_2 are compatible iff the sorts of input places p_i^1 (where $i \in [1..k]$) and $p_i^1 \in N_1$, $\cup_i St(p_i^1$ and p_j^2 (where $j \in [1..m]$) and $p_j^2 \in N_2$, $\cup_j St(p_j^2)$ are same.
- Given any two subnets N_1 and N_2, we can say the sorts of two subnets N_1 and N_2 are compatible iff the sorts of expressions defined in the guard G of all interface transitions are same, i.e., $G^1(t_i)$ (where $i \in [1..k]$) and $t_i \in N_1$, $G^2(t_j)$ (where $j \in [1..m]$) and $t_j \in N_2$, are same.

3.2 Dynamic Semantics

The dynamic semantics of PrTR nets are defined by two conditions – enabled and firing of a transition. In order to define the enabled and firing of a transition, we first give the precondition and postcondition of a transition in a PrTR net.

Let $L(f, Terms_{\Sigma,X})$ be label expression that associates with arc $f = (p,t) \in F \vee f = (t,p) \in F$. For any place $p \in P$, we can define the two functions – precond for the token consuming ($precond(L((p,t), Terms_{\Sigma,X}))$) and and postcond for token producing ($postcond(L(f, Terms_{\Sigma,X}))$) for a transition t as follows.

Definition 2 (Precondition of t): Precondition of a transition t, denoted by $precond(t)$ or $\cdot t$, is defined as the set of all input places (including the reconfigure place) of the transition that hold tokens, the sorts of the tokens are compatible with the expressions in the guard of the transition $G(t)$ (or $G^R(t)$) so that $G(t)$ (or $G^R(t)$) is true.

$\forall p \in P.precond(L((p,t), Terms_{\Sigma,X})) : p \rightarrow L((p,t), Terms_{\Sigma,X}) : \alpha$ where $L(f, Terms_{\Sigma,X}) : \alpha \in \varphi(p)$, $(p,t) \in F$, and $M'(p) = M(p) - precond(L((p,t), Terms_{\Sigma,X}))$,

Definition 3 (Postcondition of t): Postcondition of a transition t is defined as producing the tokens to the output places of the transition with the compatible sorts of the output places, e.g.,

$\forall q \in P.postcond(L(f, Terms_{\Sigma,X})) : p \rightarrow L((p,t), Terms_{\Sigma,X}) : \alpha$ where $L(f, Terms_{\Sigma,X}) : \alpha \in \varphi(p)$, $(p,t) \in F$, and $M'(p) = M(p) \cup postcond(L((p,t), Terms_{\Sigma,X})$.

From above two functions of precondition and postcondition, we can see that for any transition t, the tokens consumed in the incoming places of the transition t can be described by a substitution of label expression in the function $precond(L(f, Terms_{\Sigma,X}))$, if the substitution of label expression in the token set of preset of transition t; while the tokens produced in the postset of the transition t can be described by a substitution of label expression in the function $postcond(L(f))$, if the substitution of label expression satisfy the sort of postset of the transition t. The token set of preset and postset of transition t can be described by the substitution of sorts of preset and postset, i.e., $pre(t)(\varphi(p) : \alpha)$ and $post(t)(\varphi(p) : \alpha)$. Therefore, we have enabling and firing conditions as follows:

Definition 4 (Enabled t): A transition t, denoted by $enabled(t)$, is enabled if either the guard $G(t)$ that is true or the reconfigure function $G^R(t)$ is true.

$precond(L(f, Terms_{\Sigma,X})) \in pre(t)(\varphi(p) : \alpha)$

In Fig. 1, there are two transitions and three places with the initial markings (M_0) in places p_1 and p_3. The net structure is shown in the left box, while the net inscription is shown in the right box. The guard of transition t $G(t)$ is not true since the two transitions t and t_1 are mutual exclusively enabled. Thus the example of Fig. 1 is non-deterministic reconfiguration. When the transition t_1 is enabled if the guard $G(t)$ and reconfiguration function $G^R(t)$ are true under the substitution of tokens in the places p_1 and p_3.

If transition t_1 is enabled, required tokens specified by the label expression of input arcs of the transition must be available in the preset of the transition. If the transition t_1 is fired, those required tokens are consumed and produce some tokens that satisfy the label expression of output arcs of the transition. Both consumed tokens and produced tokens must have the same sort of incoming places of the transition and

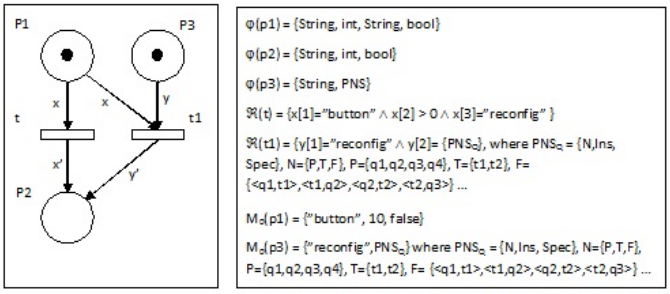

Fig. 1: Example of Reconfiguration (before reconfiguration t is enabled)

outgoing places of the transition respectively. In Fig. 1, by the initial markings in the places p_1 and p_3, we can say that the transition t_1 is enabled. Similarly the transition t is also enabled. It is worth to note that they cannot be fired at the same time.

Definition 5 (Firing of t): A transition t can be fired if

- the transition t is enabled $enabled(t)$, i.e., $G(t)$ or $G^R(t)$ is true and
- the tokens in the input places are consumed and there are tokens produced to the output places. $postcond(L(f, Terms_{\Sigma,X})) \in post(t)(\varphi(p) : \alpha)$
- in the case of reconfiguration, in addition to the tokens consumed in the regular input places, the token in the reconfigure place will be consumed and a subnet with the sort of PNS will be output. In this case, $G(t)$ will be false and $G^R(t)$ is true even $enbled(t)$ and $enbled(t^R)$ can be true. In other words t and t^R cannot be fired simultaneously.

Allowing both t and t^R to be enabled can cause the non-deterministic firing of the two transitions. In reality, the truth is t won't be able to enabled due to the disfunction of normal behavior. This can be controlled by the motion planner or supervisory controller.

In Fig. 2, it shows after the firing of transition t_1, the net is reconfigured and the markings is updated. The updated net is specified in the reconfigure place p_3. When the reconfiguration happen, the status should indicate that the system needs to reconfigure which is specified in the last field of reconfigure place. All the three fields make the reconfiguration has higher priority, the reconfigure function will ensure the condition is true so that the system can update to the new topology.

4. A PrTR Net Representation of Pressing Button Humanoid Robot

The component based architecture of a humanoid robot system with hands and legs is specified in Fig. 3. In Fig. 3, four components are specified as $MotionPlanner$, $HandMotionComponent$, $LegMotionComponent$, and $TransEnvironment$. Each component block is defined by following the above component definition. Each component captures a set of operations or functions of subunit or subsystems with required ports definition. For instance, component of *MotionPlanner* will accept all sensoring data, coordinate the tasks and sends out the commands to other components. The component *HandMotion* will conduct the hand motion including moving forward or backward in specified angle and direction. Similarly for the component of *LegMotion*. Once the command is sent out from $MotionPlanner$, the command will take all the parameters that are needed by $HandMotion$ and $LegMotion$ component. This will ensure these components can finish the task continuously without frequently often communication to $MotionPlanner$. The component $TransEnvironment$ is an alia component for the environment that sent out from $MotionPlanner$. This is just for representation purpose. By using the component $TransEnvironment$ some environment behavior is simply captured and the main behavior of the hand motion of the humanoid robot can be clearly represented. The component $HandMotion$ can be split into two subcomponents $LeftHandMotion$ and $RightHandMotion$. Similarly for the component of $LegMotion$. In the current version, we only focus on the reconfiguration of hand motion.

It is clearly to note that the behavior of each component is represented by a PrTR net. The double circled places are reconfiguration places. The double lined box is reconfiguration transition. Due to space limitation, we only show the presentation of net structure. For the net inscription, you may refer to the report [10].

4.1 Verification & Discussion

In our work we used Maude [7] model checker to verify the property specification in PrTRN model. To automatically implement the model checking using Maude, we design an

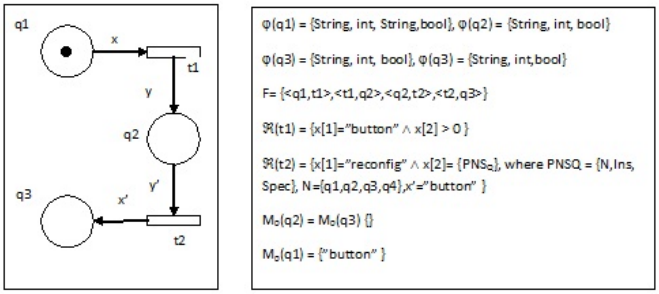

Fig. 2: Example of Reconfiguration (after reconfiguration)

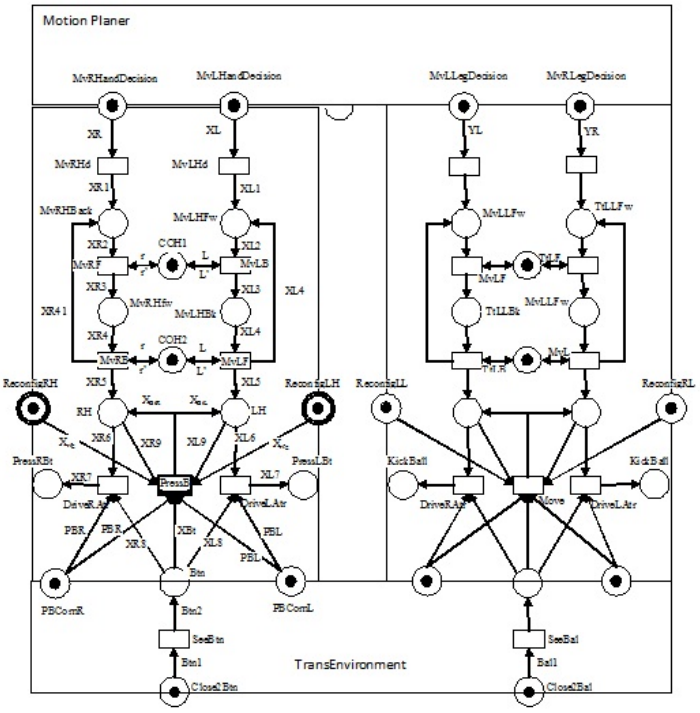

Fig. 3: PrTR Model of Reconfigurable Humanoid Robot

algorithm to translate the PrTRN to the Maude programming language.

All the properties are verified against the model and several problems are identified and fixed. The most important findings using verification is to ensure the system properties and detect errors in the design model. We have specified several properties regarding to the above categories. By running the Maude model checker, several design errors are identified. Thus the model has been fixed and updated for each detected error. Now, it is error free for the current model. For instance, in the balance category, the properties was false in any initial conditions. The problem is the places ($COH1$ and $COH2$) do not hold the behavior of another arm. To fix it, the token will be sent back to these places. However, the properties are still false. We found that this is caused by the one of the initial condition missed in these places. In addition to it, the guard condition of the transition $MvLB$ and $MvRF$ are updated with one predicate to check the initial case of moving arms.

Reconfiguration category is hard to verified. There are

several problems identified. First, the properties are false because the tokens that were consumed from the place Btn in the normal behavior cannot be available during reconfiguration. In other words, the button is not available during reconfiguration, while in reality, the button is available all the time. This is easy to fix by sending the token back to the correct place. Similar problems in the places of $PBComR$ and $PBComL$, which indicates the pressing command is still valid.

5. Conclusion

Petri nets are an elegant model for concurrency. Combining with graphical representation and its mathematical background, Petri Nets have been widely used to specify and verify concurrent multi-processes systems. In this paper, we have proposed "Predicate Transition Reconfigurable Nets" (PrTRN), a formalism to specify reconfiguration in the autonomous robotics systems with dynamic structure. We have shown the expressiveness of this formalism through the modeling of pressing button scenario of humanoid robot. The use of PrTRN facilitates the tasks of the developer that want to realize formal specification of robotics systems.

The future works include two aspects – transparency and automation. We will develop some transformation rules that help the translation from PrTRN toward regular PrT nets or even Place Transition nets. So that automatic code generation from PrTRN can be realized. To improve the verification at the analyzing level, we need to work on the automatic verification tool that can be used to verify the PrTRN net automatically without the interaction of human beings. Therefore the PrTRN models and the required proprieties will be automatically related and checked.

Acknowledgment

The authors would like to thank all reviewers for the kindly comments and suggestions on this work.

References

[1] A. Asperti and N. Busi. Mobile petri nets. *Mathematical. Structures in Comp. Sci.*, 19(6):1265–1278.

[2] A. Asperti, N. Busi, P. Porta, and S. Donato. Mobile petri nets. Technical report, 1996.

[3] E. Badouel, Éc. Nat, S. Polytechnique, and Y. Cameroun. Modeling concurrent systems: Reconfigurable nets. In *In Proc. Int. Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTAâĂŹ03*, pages 1568–1574. CSREA Press, 2003.

[4] E. Badouel and J. Oliver. Reconfigurable Nets, a Class of High Level Petri Nets Supporting Dynamic Changes within Workflow Systems. Research Report RR-3339, INRIA, 1998.

[5] M. G. Buscemi and V. Sassone. High-level petri nets as type theories in the join calculus. In *Proceedings of the 4th International Conference on Foundations of Software Science and Computation Structures*, FoSSaCS '01, pages 104–120, London, UK, UK, 2001. Springer-Verlag.

[6] I. Cervesato. Petri nets and linear logic: a case study for logic programming. In *In Proc. of GULP-PRODE'95*, pages 313–318. Palladio Press, 1995.

[7] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and J. F. Quesada. Maude: specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2):187–243, 2002.

[8] C. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of conference on Organizational computing systems*, COCS '95, pages 10–21, New York, NY, USA, 1995. ACM.

[9] C. Fournet and G. Gonthier. The Join Calculus: A Language for Distributed Mobile Programming. *Applied Semantics*, pages 268–332, 2002.

[10] Y. Fu and S. Drager. Reconfiguration Anlysis of Automatic Robotics Systems. Technical report, Air Force Research Lab, August 2012.

[11] H. J. Genrich. Predicate/Transition Nets. *Lecture Notes in Computer Science*, 254, 1987.

[12] X. He. A formal definition of hierarchical predicate transition nets. In *Proceedings of the 17th International Conference on Application and Theory of Petri Nets*, pages 212–229, London, UK, 1996. Springer-Verlag.

[13] L. Kahloul, A. Chaoui, and K. Djouani. Modeling and analysis of reconfigurable systems using flexible petri nets. In *Proceedings of the 2010 4th IEEE International Symposium on Theoretical Aspects of Software Engineering*, TASE '10, pages 107–116, Washington, DC, USA, 2010. IEEE Computer Society.

[14] K. Laid and C. Allaoua. Code mobility modeling: a temporal labeled reconfigurable nets. In *Proceedings of the 1st international conference on MOBILe Wireless MiddleWARE, Operating Systems, and Applications*, MOBILWARE '08, pages 34:1–34:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[15] K. Laid and C. Allaoua. Coloured reconfigurable nets for code mobility modeling. *Int. J. of Computers, Communications & Control*, 2008:358–363, 2008.

[16] M. Llorens and J. Oliver. Structural and dynamic changes in concurrent systems: Reconfigurable petri nets. *IEEE Trans. Comput.*, 53(9):1147–1158, Sept. 2004.

[17] F. Rosa-Velardo, O. Marroquín-Alonso, and D. de Frutos-Escrig. Mobile synchronizing petri nets: A choreographic approach for coordination in ubiquitous systems. *Electron. Notes Theor. Comput. Sci.*, 150(1):103–126, Mar. 2006.

[18] R. Valk. Self-modifying nets, a natural extension of petri nets. In G. Ausiello and C. BÃűhm, editors, *Automata, Languages and Programming*, volume 62 of *Lecture Notes in Computer Science*, pages 464–476. Springer Berlin / Heidelberg, 1978.

[19] R. Valk. Generalizations of petri nets. In J. Gruska and M. Chytil, editors, *Mathematical Foundations of Computer Science 1981*, volume 118 of *Lecture Notes in Computer Science*, pages 140–155. Springer Berlin / Heidelberg, 1981.

[20] W. M. P. van der Aalst, K. M. van Hee, and R. A. van der Toorn. Component-based software architectures: A framework based on inheritance of behavior. *Science of Computer Programming*, 42(2-3):129–171, 2002.

[21] K. M. van Hee, I. A. Lomazova, O. Oanea, A. Serebrenik, N. Sidorova, and M. Voorhoeve. Nested nets for adaptive systems. In *Proceedings of the 27th international conference on Applications and Theory of Petri Nets and Other Models of Concurrency*, ICATPN'06, pages 241–260, Berlin, Heidelberg, 2006. Springer-Verlag.

[22] J. Wang, X. He, and Y. Deng. Introducing Software Architecture Specification and Analysis in SAM through an Example. *Information and Software Technology*, 41(7):451–467, 1999.

[23] D. Xu and Y. Deng. Modeling mobile agent systems with high level petri nets. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 5, pages 3177 –3182 vol.5, 2000.