Requirements Planning with Event Calculus for Self-adaptive Multi-agent System

Wei Liu[1,2], Feng Yao[1] and Ming Li[1]
[1]School of Computer Science and Engineering, Wuhan Institute of Technology, Wuhan, China
[2]Hubei Province Key Laboratory of Intelligent Robot, Wuhan Institute of Technology, Wuhan, China

Keywords: Requirement Planning Graph, Event Calculus, Cooperative Robot System.

Abstract: Self-adaptation of Multi-agent cooperative systems requires dynamic decision making and planning at runtime. Modeling the contextual and executable requirements of such systems as planning actions and states, this paper proposes a requirements-driven planning approach to self-adaptation. The planning model includes the states of the system context and the actions describing the behaviors of its multiple agents; the interactions between these agents and their environment are computed through an expansion of the requirements-driven planning graph, which is then used to verify whether the agents can collaborate in order to reach the desired goal states from their current states. In addition, the requirements are represented for Event Calculus to facilitate monitoring and reasoning about the actions of agents, achieving requirements driven planning at runtime.

1 INTRODUCTION

Self-adaptive systems must be capable of synthesising adaptation strategies at runtime to deal with the dynamically changing and uncertain environment in which they evolve. Engineering such self-adaptive systems, it is argued that requirements, architectures, and middlewares are all principal techniques (Salehie and Tahvildari,2009).

Multi-Agent Systems(MAS) are cooperative models and distributed optimization techniques, which can be useful in self-adaptive systems. Engineering such cooperative self-adaptive systems requires dealing with requirements, and software architectures. Because of possible deviations between the systems agents runtime behavior and the requirements, self-adaptive systems shall be requirements-aware (Sawyer et al., 2010).

Representation and modelling requirements, approaches for self-adaptive software include REAssuRE (Welsh et al., 2011), RELAX (Whittle et al., 2010) already use goal-based modelling notations. However, these approaches do not consider requirements as being contextual and executable.

Contextual requirements mean that changes in the software context can trigger the changes of the predefined software requirements. Early research on contextual requirements (Ali et al., 2010), (Seyed and Minseok, 2012) only used contexts as the preconditions for goal decomposition or the triggering event in business processes modeling.

Executable requirements mean that requirements should be reasoned about at runtime and interpreted as implemented behaviors. Bencomo et al., (2010) treated requirements as the firstclass runtime entities for software systems to reason about them and to relax their interpretations at runtime. However, they do not represent the dynamic information of adaptive requirements.

To meet these challenges of contextual and executable requirements, this paper proposes a requirements-driven planning approach for the runtime self-adaptation. The approach consists of the following major elements:

1) We support changes in the software context and use requirements to drive adequate planning at runtime;

2) We define a requirements-driven planning graph model that includes states representing the software contexts and actions representing the system behaviors;

3) We propose an algorithm for expanding the requirements-driven planning graph to model the Interactions between agents and their

Liu, W., Yao, F. and Li, M.
Requirements Planning with Event Calculus for Self-adaptive Multi-agent System.
DOI: 10.5220/0005660001100117
In *Proceedings of the 8th International Conference on Agents and Artificial Intelligence (ICAART 2016) - Volume 1*, pages 110-117
ISBN: 978-989-758-172-4

environment as Event Calculus specifications;

4) We use the specification to monitor whether the agents can collaborate to reach the desired goal states from their current states.

Our major contribution here is in translating the requirements model into Event Calculus to facilitate necessary reasoning for requirements-driven planning at runtime.

To illustrate the concepts and algorithms, we use a simple but representative scenario of two robots to adapt their actions through requirements-driven planning. One robot Nao has more capabilities in monitoring the environment (including the status of the second robot iCreate), whilst the iCreate is a "dumb" cleaning robot who has excellent capability in cleaning (its main designed purpose), speedy and stable movement, and fall-detection. The states for them are initially different, the goals (individual) states may mutually exclude, interfere, or support each other, under varying situations. Their interpretation of the environment observable behaviour can also be inconsistent, e.g., Nao has internal notion of whether the door is open or close through its image-detection capability, whilst iCreate don't. Nonetheless, a cohesive plan that benefit both robots and the composed multi-agent system with respect to the collective requirements goals is achievable with the proposed framework.

The remainder of the paper is organized as follows. Section II defines the basic concepts of requirements-driven planning and event calculus planning. Section III presents the approach and details the requirements-driven planning graph expansion and requirements extraction algorithms. Section IV reports the results of our experiments with a robotics application. Section V presents related work. Finally, Section VI makes some concluding remarks.

2 PRELIMINARIES

2.1 Self-adaptation Through Planning

Plans are automatically generated by finding a path from the current states of system and its environment to goal states, which amounts to choose and order a sequence of actions in order to achieve the goal.

In our previous work, we have applied a probabilistic planning algorithm, PGraphPlan, to support modelling uncertain requirements (Esfahani and Malek, 2013). Compared to a probabilistic planning performed offline (Little and Thibaux, 2007), our online planning and replanning approach is considered more suitable for runtime requirements driven adaptation for two main reasons: (i) it represents the current states, goal states and system behavior as a planning model, and (ii) it induces a contingency plan once when the states of system environment or goals dynamically change.

A task for goal-oriented requirements planning can be defined as follows.

Definition 1 (***Goal-oriented Requirement Planning Task***): A goal-oriented requirement planning task is a triple $T = <O, I, G>$, where

- O is a set of actions,
- I is a set of the initial states,
- G is a set of the goal states.

In logic planning, a state is defined as an atomic boolean literal without any nested propositional expression structure. At runtime, when the goal is in any one of the initial states, a new planning will be triggered to execute some actions. The planning will not be terminated until any goal state is reached.

2.2 Event Calculus Planning

Since we are interested in runtime self-adaptive multi-agent systems by which some values of states will change during the execution process, two issues must be addressed. The first is to identify the states to monitor which could trigger plan actions in the next layer of actions (Tun et al., 2009). The second is to identify feature interactions (Tun et al., 2013), in other words, which planned actions shall not execute at the same timestamp. Two actions associated with the same agent cannot be executed synchronously, e.g., in the case of "landmark detection" and "face detection" associated with the NAO robot.

To address these issues as we did for development-time diagnosing and detection of feature interaction problems, for runtime reasoning about actions and change at runtime we still use Event Calculus because it supports temporal descriptions of the events and actions. Here we give the basic predicates of discrete event calculus used in this paper (Mueller, 2004). *Initiates*(e, f, t) means that Fluent f becomes true after event e occurs at time t. *Terminates*(e, f, t) means that Fluent f becomes false after event e occurs at time t. *Happens*(e, t) means that Event e occurs at time t. *HoldsAt*(f, t) means that Fluent f is true at time t. In our paper, Fluent f indicates the states of agent or environment, time point t could be a real time point or some time stamp, and Event e indicates an action executed by the agents of the system.

3 FRAMEWORK OF REQUIREMENT-DRIVEN PLANNING

Requirements-driven planning defines a runtime model for requirements in order to adapt them to contextual and dynamic changes of the system. The main elements of our requirements-driven planning framework are the behaviour and executable models (see Figure 1).

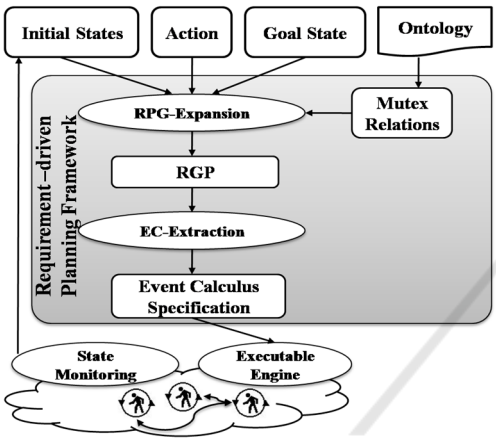

Figure1: Requirements-driven planning framework.

The input to our frameworks includes the three elements for the goal-oriented requirements planning task in Definition 1, i.e., the initial and goal states together with the plan actions. The current states will be monitored to update the initial states for re-plan. Additionally, the ontology represents the vocabulary of domains and enables us to reason about its concepts using subsumption, semantic equivalence and disjoint. The result of this reasoning is represented as mutex relations. Using the goal-oriented requirement tasks and mutex relations, a requirements-driven planning graph (RPG) is obtained by an expansion algorithm. Another algorithm for extracting Event Calculus specifications follows, which is the input of executable engine to control the software components or software agents. In the subsequent section, we detail the elements of the framework.

4 REQUIREMENTS-DRIVEN PLANNING MODELING

4.1 Requirements-driven Planning Graph

The plan actions are pre-defined to represent all actions that could be executed by different agents in the self-adaptive system.

Definition 2 (***Plan Action***): A plan action a is a triple $<pre(a); add(a); del(a)>$, where

- $pre(a)$ is a set of the preconditions of a,
- $add(a)$ is a add list of a, and
- $del(a)$ is a delete list of a.

The add and delete lists are sets of next-states. For a state $s \in add(a)$ and $s' \in del(a)$, we say that execution of action a will make s true and make s' false. If all the preconditions of a plan action could be satisfied, it will be triggered to the effect adding some next-states and deleting some other next-states.

This modeling results in a Requirements-driven Planning Graph (RPG), defined as follows.

Definition 3 (***Requirements-driven Planning Graph (RPG)***): A requirements-driven planning graph RPG is a couple $<N, E>$ where

- N is a set of nodes, organized by alternating action and state in a layered sequence $N = S_0 \oplus A_1 \oplus S_1 \oplus A_2 \oplus S_2 \cdots S_n$ *in which*
  - S_0 are the initial states, that is $S_0 = I$,
  - A_n action layer A_i includes all actions a whose preconditions are met in the previous state layer, i.e, $pre(a) \subseteq S_{i-1}$,
  - A state layer S_i includes all actions $a_j^i \in A_i$ such that $S_i = S_{i-1} \cup \bigcup add(a_j^i) \setminus \bigcup del(a_j^i)$;
- E is a set of directed edges $E = E_{pre} \cup E_{add} \setminus E_{del}$ where:
  - $Epre(s, a)$ is the set of edges from state $s \in S_{i-1}$ to action $a \in A_i$ such that $s \in pre(a)$,
  - $Eadd(a, s)$ is the set of edges from action $a \in A_i$ to state $s \in S_i$ such that $s \in add(a)$,
  - $Edel(a, s)$ is the set of edges from action $a \in A_i$ to state $s \in S_i$ such that $s \in del(a)$.

At runtime, an RPG will keep being expanded with actions until all goal states S_n are reached.

4.2 Reasoning Mutex Relations using Ontology

An ontology is "a specification of a representational vocabulary for a shared domain of discourse" (Gruber, 1993). The purpose of ontology is to model

and reason about domain knowledge. In OWL description logic, semantic equivalence amounts to a double subsumption, i.e., two concepts C and D are semantically equivalent, denoted C ≡ D, iff all instances of C belong to D and vice versa. When two concepts C and D do not share any instance, they are said to be semantically disjoint, written C≢ D.

In logical planning, states are associated with Boolean propositions or predicates, which is however insufficient to represent the full semantics of the states for requirements driven planning. Hence, we extend the semantics of a state as follows.

Definition 4 (***State Semantics***): A state semantics is a triple $s = <prop(s), pred(s), val(s)>$, where:

- $prop(s)$ is the property of s;
- $pred(s)$ is the predicate of s;
- $val(s)$ is the value of property.

In this way, the ontological relationships can now be expressed on the states as following mutex relations. There are two state s_i and s_j, if $prop(s_i) \equiv prop(s_j)$, $val(s_i) \equiv val(s_j)$ and $pred(s_i) \not\equiv pred(s_j)$, then s_i and s_j have mutex relation, denoted as $s_i \diamondsuit s_j$.

Extending state semantics onto the plan actions associated with the states, there can be three types of mutex relations between plan actions: inconsistent effects, interference and competing needs. They are categorized using the mutex relation between the state semantics belonging to the set of preconditions and effects of the plan actions involved. Inconsistent effects relation indicates that two plan actions a_p and a_q do not have inconsistent effects, which is denoted as $a_p \diamondsuit^{ie} a_q$. Interference relation indicates that two plan actions a_p and a_q do not have interference, which is denoted as $a_p \diamondsuit^{in} a_q$. Competing needs relation indicates that two plan actions a_p and a_q do not have competing needs, which is denoted as $a_p \diamondsuit^{cn} a_q$.

5 REQUIREMENTS-DRIVEN PLANNING ALGORITHM

5.1 RPG-Expansion Algorithm

We propose an algorithm RPG-Expansion that takes as input an RPG with i layers and the plan action set O_i and produces an RPG at the $i + 1$ layer if possible. RPG-Expansion is executed inductively until the goal states are included in the state layer S_i.

RPG-Expansion will be executed until the goal is included in the state layer S_j. The output is a requirement planning graph RPG_{i+1}.

Algorithm 1: RPG-Expansion(RPG_i, O_i).

Require: RPG_i and the plan action set O_i
Ensure: RPG_{i+1} or fail
$S_j = S_{j-1}$;
for all $a_j = <pre(a_j), add(a_j), del(a_j)> \in A_j$ **do**
  $S_j \leftarrow S_j \cup add(a_j)$
  $S_j \leftarrow S_j \setminus del(a_j)$
  if $G_i \subseteq S_j$ **then**
    return RPG_{i+1}
  end if
end for
if $S_j == S_{j-1}$ **then**
  return fail
else
  for all $a_p = <pre(a_p), add(a_p), del(a_p)> \in O_i$ **do**
    for all $a_q = <pre(a_q), add(a_q), del(a_q)> \in A_j$ do
      if $a_p \diamondsuit^{cn} a_q \vee a_p \diamondsuit^{ie} a_q \vee a_p \diamondsuit^{cn} a_q == 0$ **then**
        if $pre(a_k) \subseteq S_j$ then
          $A_{j+1} \leftarrow A_{j+1} \cup a_k$
        end if
      end if
    end for
  end for
end if
return RPG_{i+1}

5.2 EC-Extraction Algorithm

Two main functions of runtime executable extraction shall be realized by the EC-Extraction algorithm. First, the algorithm can analyze the substitute relation between plan actions in the same action layer in RPG and generate different solution for achieving the goal. Second, the algorithm can decide which of the related states should be monitored at every timestamp, which triggers the modification of the behaviour of the system to achieve the goal states in a changing environment.

Event calculus (EC) is suitable to represent self-adaptive policies in our approach to realize the second function for two reasons. First, EC can describe all the artifacts in requirement plan graph without semantic lost. A state s (trigger state or result state) of plan action is represented as fluent in EC, which is denoted as $f(s)$. A plan action a is represented as event in EC, which is denoted as $EC(a)$. Different predicates in EC can represent the edges in graph. Second, EC can describe the sequence of two plan actions which could not execute at the same timestamp. The asynchronous relation between two plan actions a_i and a_j is denoted as $AC(a_i, a_j)$.

Table 1shows the rules for translating an RPG into an executable model.

Table 1: Rules of generating self-adaptive policies.

Rules	Description
Rule1	For every $s \in S_0$, InitiallyP[$f(s)$] is generated;
Rule2	For every $a \in A_k$, $\forall s \in add(a)$, Happens[a, t_i] and Initiates[a, f(s), t_j] ($t_j = t_i+1$) are generated;
Rule3	If a_i, $a_j \in A_k$ and $AC(a_i, a_j)$, then Happens[a_i, t_i] and Happens[a_j, t_j] ($t_j = t_i+2$ or $t_i = t_j+2$) are generated;
Rule4	For every $a \in A_k$ (k>1), $\forall s \in pre$ (a), if $\forall a' \in A_{k-1}$, s_i ! $\in add(a')$ and Happens[a, t_i], then HoldsAt[$f(s)$, t_j] ($t_j = t_i$-1)is generated;
Rule5	For every $a \in A_k$, $\forall s \in del(a)$, if there is Happens[a, t], Terminate[a, $f(s)$, t_j] ($t_j = t_j + 1$) is generated;
Rule6	For every s$\in S_l$ (S_l is the last state layer) and $\forall a \in A_l$, there is Happens[a, t_l](t_l is the latest one of all a in last action layer) , if $s \in G$ and s ! $\in add(a)$, then HoldsAt[$f(s)$, t_j]($t_j = t_i + 1$).

In these rules, we use time stamp as the time in predicates. If these self-adaptation policies are used in real time, then " $t_j = t_i + 1$" is replaced by " $t_j > t_i$" and " $t_j = t_i - 1$" is replaced by" $t_j < t_i$".

We propose an algorithm EC-Extraction for generating a runtime executable model.

Algorithm 2: EC-Extraction(*RPG*).

Require: *RPG*
Ensure:
1: Set a time stamp t ← 0
2: Get the initial states *ini*
3: Generate Initially(*ini*)
4: *t* + +
5: **for all** $A_k \in RPG$ **do**
6: **for all** $a_i \in A_k$ **do**
7: **if** there is an action $a_j \in list$ and $AC(a_i, a_j)$ **then**
8: add a_i into *nextlist*
9: **else** add a_i into *list*
10: **end if**
11: **end for**
12: **for all** a_k=<*pre*(a_k), *add*(a_k), del(a_k)> $\in list$ **do**
13: generate Happens(a_k, *t*+1) and Initiates(a_k, add(a_k), *t*+1)
14: **for all** $a_t \in A_{k-1}$ **do**
15: **if** $s \in pre(a_k)$ and s_i ! $\in add(a_t)$, **then**
16: generate HoldsAt(*s*, *t*)
17: **end if**
18: **end for**
19: **end for**
20: **if** *nextlist* is not empty then
21: *t* + +
22: list ← nextlist
23: **go to** 12
24: **end if**
25: **end for**

The algorithm EC-Extraction represents how to translate the requirements-driven planning graph RPG into Event Calculus format.

6 EXPERIENCE AND ANALYSIS

6.1 Case Study: Multi-robots Cooperation

We have been experimenting with a requirements planning through an early prototype demonstrator of cleaning scenarios with two heterogeneous robots. Both iRobot Create and NAO rely on discovery protocols to advertise their presence in the environment, the former uses Bluetooth discovery while the latter uses Bonjour. The two robots need to collaborate in order to secure a particular area in our laboratory.

Table2 shows the 10 states mentioned in cleaning scenarios are described with PDDL.

Table2: States description with PDDL.

State	Description
s_1	(light ?lab ?notdark)
s_2	(statusNao ?nao ?available)
s_3	(know ?door ?closed)
s_4	(at ?door ?post)
s_5	(at ?create ?post)
s_6	(statusCreate ?create ?wait)
s_7	(safearea ?ar)
s_8	(cleaned ?create, ?lab)
s_{10}	(statusDoor ?door ?closed)
s_{12}	not(at ?create ?post)

Consider for example in robots cleaning scenario, the requirements-driven planning task T_{rob} =<O_{rob}, I_{rob}, G_{rob}>. The input includes I_{rob} = {s_1, s_2, s_6, s_{10}} and G_{rob} = {s_8}. There are six actions defined in Orob which includes a_1:detectDoor, a_2:getCreatePost, a_3:caculateSaftDistance, a_4:cleanInSafe, a_5:availableCreate, and a_6:detectCreate. A requirements-driven planning graph generator is realized with RPG-Expansion algorithm. There is a valid requirement planning graph RPG_3 that translates I_{rob} to G_{rob}, as shown in Figure 2.

Example1: there are two plan actions a_1: naoDetectdoor and a_6 : naoDetectCreate in a_1 have asynchronous relation $AC(a_1, a_6)$, according to rule 3 Happens[a_1, t_1] and Happens[a_6, t_3] are generated.

Example2: there is a plan action a_4 : cleanInSafe; $s_{12} \in del(a_4)$ and Happens[a_4, t_5], according to rule 5 Terminate[a_4, $f(s_{12})$, t_6] is generated.

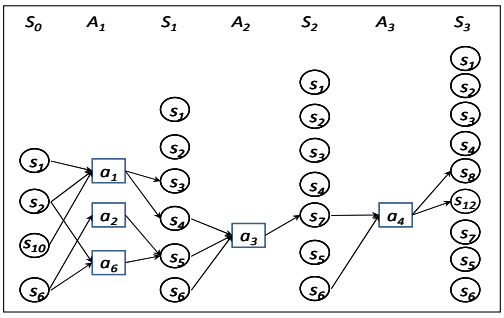

Figure 2: *RPG*$_3$ for multi-robots cooperation scenario.

The executable adaptive model could decide that which of the related states should be monitored at every timestamp. For example, HoldsAt[$f(s_6)$,t_2] means that the value of fluent $f(s_6)$ should be true at timestamp t_2. If the value of fluent $f(s_6)$ is changed at t_2, then a new requirement planning task $T_{rob'}$ = <O_{rob}, $I_{rob'}$, G_{rob}>will be triggered. In the condition of $I_{rob'}$ ={s_2, s_3, s_4, s_{13}}, the plan action a_5: *avaiableCreate* will be induced into the planning to achieve the goal. A new executable adaptive model will be generated as shown in Figure 3.

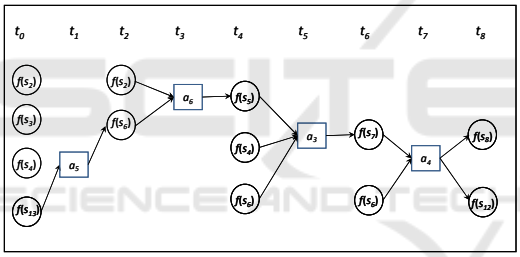

Figure 3: A solution for new planning.

6.2 Result Analysis

A working prototype that follows the executable model has been implemented, as is shown in Figure 4.

The generation of behaviour model is realised by a RPG-Expansion module. The effectiveness of the RPG-Expansion algorithm is mainly measured by the mutex rate, which is the proportion of number of mutex relations to number of all relations among actions.

Provided that there are k step of matching and the number of action in agent capability model is n. The number of mutex relations is m and the number of all relation among actions is $n(n-1)/2$. With this assumption, mutex rate is $2m/n(n-1)$.

Without the mutex relations between actions, traditional decision making methods for self-adaptation need to search for a match for all actions

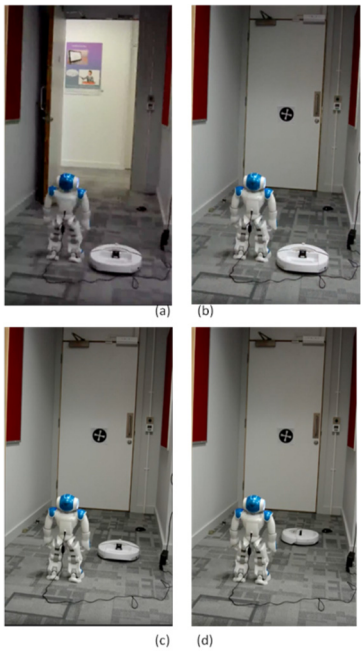

Figure 4: Collaborating Robots Rlanning with Changing Contexts: (a) Door is open, (b) Door is closed (detected by Nao), (c) iCreate moves (instructed by Nao), (d) iCreate turns (obstacle detected).

in planning graph. The worst case is when the matching succeeds at the last round, in which the number of matching is nk. Our approach analyzed the mutex relations before the decision making process. The average result of matching is $2mk/(n-1)$ for $0 \le m \le n(n-1)/2$, $2mk/(n-1) \le nk$. The smaller m is, the more average number of matching can be reduced in our approach.

7 RELATED WORK

A number of proposals offer goal based requirement models for requirements-driven self-adaptive. Baresi et al., (2010) propose FLAGS requirements models which are based on the KAOS framework and are targeted at adaptive systems. In FLAGS, fuzzy goals are mostly associated with non-functional requirements. Souza et al., (2011) note that the (partial) un-fulfilment of requirements triggers adaptation. They introduce awareness requirements to refer to success, failure, performance and other properties of software requirements and propose to monitor changes in these properties and decide when adaptation should take place. These approaches alter the goal model at runtime and enforce adaptation

directives on the running system. Our framework monitors the goal state at runtime and alters the plan actions for self-adaptation.

Sabatucci et al., (2013) proposed a GoalSPEC language for supporting evolution and self-adaptation. In GoalSPEC every goal describes three elements: initial state, final state and actors. The actors operate a state transition from an initial state to n final state. In the next work, the overview of a framework for adaptive workflow was presented to find a distributed plan to address the injected goals. Our framework has the potential to enrich these works by the consideration of parallel and mutex relations between actions which considers all the reasonable solutions for current states.

Planning-based approach shall plan future behaviour of the system continuously. Sykes proposed an implementation of Krammer (Sykes et al., 2008) and Magee's three-layer architecture that distinguishes between component-based control, architectural (re)configuration, and high-level task (re)planning (Kramer and Magee, 2007). Plans generated from the highest layer (i.e., goals) are configured by the middle layer (i.e., configurations) to be executed by the lowest layer (i.e., components). Some solutions have begun to study the task replanning at runtime: PLASMA (Tajalli et al., 2006) supports replanning and adapting the middle layer in a similar, layered architecture, which is to provide a framework that automates the generation and enactment of plans while the employed feedback loops. Requirements-Driven Feedback Loops (Chen et al., 2014) exerts adjusting controls to optimize away limiting uncertainty factors. However, current approaches are unable to intelligently compute new adaptation plans by taking into account mutex relations using the semantic knowledge of the application domains.

8 CONCLUSIONS

The main contribution of the paper is a semantic rule-based transformation from requirements model to event calculus specifications that can support runtime interaction with environment and replanning the multi-agent system at runtime. Comparing with the predefined policy-based approaches, planning-based approach can overcome the conflicts between policies that are otherwise impossible for system to resolve for achieving the goal states collectively. Our replanning approach leaves out the translation of control actions into execution operations and structural adaptations, which we believe is reasonable for executing the known plan actions at runtime.

Our future work will integrate this proposal with other multi-agent interaction modeling techniques based on the agent commitments and we will conduct case studies on the automated guided vehicle (AGV) domain that include human agents.

ACKNOWLEDGEMENT

The authors would like to thank B. Nuseibeh, Y. J. Yu, A. Bennaceur in Open University. Project supported by the National Natural Science Foundation of China under Grant (No. 61502355, and No.61272115), the Natural Science Foundation of Hubei Province (No.2014CFB779), the Doctor foundation for Science Study Program of Wuhan institute of technology (No.K201475).

REFERENCES

Ali, R., Dalpiaz, F., and Giorgini, P., 2010. A goal-based framework for contextual requirements modeling and analysis. *Requir. Eng., vol. 15, no. 4, pp. 439-458.*

Baresi, L., Pasquale, L. and Spoletini, P., 2010. Fuzzy goals for requirements-driven adaptation. *In Requirements Engineering Conference (RE), 18th IEEE International. IEEE, 2010, pp. 125–134.*

Bencomo, N., Whittle, J., Sawyer, P., Finkelstein, A. and Letier, E., 2010.Requirements reflection: requirements as runtime entities. *In Proc. of the 32nd ACM/IEEE International Conference on Software Engineering, ICSE, pp. 199-202.*

Chen, B., Peng, X., Yu Y., Nuseibeh, B. and Zhao W., 2014.Self-adaptation through incremental generative model transformations at runtime. *In the 36th International Conference on Software Engineering.*

Esfahani, N., Malek, S., 2013. Uncertainty in Self-Adaptive Software Systems. *Software Engineering for Self-Adaptive Systems II Lecture Notes in Computer Science Volume 7475, pp214-238.*

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition, vol. 5, no. 2, pp. 199C220, [Online]. Available:http://dx.doi.org/ 10.1006/knac.1993.1008.*

Kramer, J., and Magee, J., 2007. Self-managed systems: an architectural challenge. *In Proc. of Workshop on the Future of Software Engineering, FOSE, pp. 259-268.*

Little, I. and Thibaux, S., 2007. Probabilistic planning vs replanning. *In Proceedings of the ICAPS'07 Workshop on the International Planning Competition: Past, Present and Future.*

Mueller E. T., 2004. Event calculus reasoning through satisfiability. *J. Log.Comput., vol. 14, no. 5, pp. 703-*

730.

Salehie, M. and Tahvildari, L., 2009. Self-adaptive software: Landscape and research challenges, TAAS, vol. 4, no. 2.

Sawyer, P., Bencomo, N., Whittle, J., Letier, E. and Finkelstein, A., 2010. Requirements-aware systems: A research agenda for re for self-adaptive systems. *In Proc. of the 18th IEEE International Requirements Engineering Conference, RE, pp. 95-103.*

Seyed, H.S. and Minseok, S., 2012. Understanding Requirement Engineering for Context-Aware. *Journal of Software Engineering and Applications,Vol.5 No.8.*

Sykes, D., et al, 2008. From Goals to Components: A Combined Approach to Self-management. *In Proceedings of Workshop on Software Engineering for Adaptive and Self-managing Systems.*

Sabatucci, L., Ribino, P., Lodato, C., Lopes, S., Cossentino, M. , 2013. GoalSPEC: a Goal Specification Language supporting Adaptivity and Evolution. *In: EMAS 2013, LNAI 8245. p.237–256.*

Tajalli, H., Garcia, J., Edwards, G. and Medvidovic, N., 2010. PLASMA: a plan-based layered architecture for software modeldriven adaptation. *In Proceedings of IEEE/ACM International Conference on Automated Software Engineering, pp.467-476.*

Tun, T. T., Jackson, M., Laney, R., Nuseibeh, B. and Yu Y., 2009. Are your lights off? using problem frames to diagnose system failures. *In 21st IEEE International Requirements Engineering Conference (RE), vol. 0, pp. 343-348.*

Tun, T. T., Laney, R., Yu, Y. and Nuseibeh, B., 2013. Specifying software features for composition: A tool-supported approach. *Computer Networks. vol. 57, no. 12, pp. 2454-2464, feature Interaction in Communications and Software Systems.*

V. E. Silva Souza., Lapouchnian, A., Robinson, W.N., Mylopoulos, John., 2011. Awareness Requirements for Adaptive Systems. *In ICSE Symposium on Software Engineering for Adaptive and Self-Managing Systems, SEAMS 2011, Waikiki, Honolulu , HI, USA, May 23-24.*

Welsh, K., Sawyer, P. and Bencomo, N., 2011.Towards requirements aware systems: Run-time resolution of design-time assumptions. *In Proc. of the 26th IEEE/ACM International Conference on Automated Software Engineering, ASE, pp. 560-563.*

Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C., and Bruel, J.-M., 2010. Relax: a language to address uncertainty in selfadaptive systems requirement. *Requir. Eng., vol. 15, no. 2, pp. 177-196.*