Goal-Oriented Requirements Modelling for Running Systems

YunSong Jian, Tong Li, Lin Liu
Tsinghua National Laboratory for
Information Science and Technology (TNList)
School of Software, Tsinghua University
Beijing, China
jianyunsong@gmail.com
{litong08@mails., linliu@}tsinghua.edu.cn

Eric Yu
Faculty of Information
University of Toronto
Toronto, Canada
yu@ischool.utoronto.ca

***Abstract*— Today, software systems are moving towards online deployment as collaborative composite services, the operating environments and users' needs for which are continuously changing. Thus, it is important to understand how to cope with run-time requirements by dynamic adaption at different levels. This paper aims to summarize our general position and understanding of this problem. In particular, we introduce a typology – a theoretical classification framework of different types of systems, with increasing levels of capability, namely, the static, reactive, adaptive and collaborative systems. The typology is built on a common architecture using rule base to store knowledge for run-time use. The accompanying development method includes a series of steps to allow run-time adaptation, and a goal-oriented modeling method and notation to analyze possible requirements changes at run-time.**

Keywords- goal-oriented requirements modelling; run-time adaptation

I. INTRODUCTION

Conventional Requirements Engineering research focuses more on the requirements elicitation and analysis activities in advance of design. Setting out from an initial problem statement, requirements engineers understand users' needs, environmental constraints, model system requirements, compare and evaluate alternative designs, and make rational design decisions. Once the system under design is deployed and executed, requirements related activities step into a secondary position – handling change requests. Usually, these change requests are handled offline, based on a formal or informal decision making process of the development team. Today's systems are required to execute under a more open and dynamic environment, new requirements and services emerge constantly, existing systems need to evolve and reorganize to cope with these run-time needs.

There are many existing research work in the RE and SE literature. In [5], Fickas and Feather have clearly stated the significance of monitoring requirements studied based on the analysis to two commercial software cases. In contrast, Jureta et al. [11] have proposed formalism in response to the emergent of services computing and its requirements for a transformation from static requirements to dynamic requirements. A more comprehensive probe into the research issues, challenges in this area is from the research roadmap by Cheng et. al. on Software engineering for self-adaptive systems [3]. Other related work is in the area of software adaptation [1] and evolution [16]. In [15], Kokar, et. al proposed a control theory-based framework for specifying and designing software that controls itself during operation. All these efforts agree on the importance of the research problem of requirements modeling, monitoring, evolution in an on-going basis for a running, live and ever-changing system. Most of the existing research aims at building a formalism to support the aforementioned tasks better, in the sense of, self-adaptive, automated, minimal disturbance to the running system.

This paper aims to achieve a common understanding to the basic concepts of requirements analysis for running systems. In other words, the paper proposes a conceptual framework to illustrate the shift of paradigms from conventional design-time requirements analysis, to run-time reaction according to the environmental changes, to today's on-demand services requirements and capability match-making and composition. The fundamental goal of the framework is to explicitly point out what has to be defined prior to running, and what is changeable during run time, how to make the system adaptable to run-time demands without taking the system offline. Since we are taking a top-down analysis process, the conceptual framework sets out from very simple concepts, and uses a well-known example of execution scenarios in traffic light control software. We then gradually refine it and incorporate more design and implementation considerations.

II. THE GOAL-ORIENTED MODEL OF RUN-TIME REQUIREMENTS

In this paper, we model system requirements with a 4-tuple: <G, E, A, R>. The definitions of the four elements are as follows:

Definition 1. E = $\{e_0,...,e_m\}$ is a set of environmental variables. If $e_i \in$ E, that means e is an environmental variable that is observable to the system. This can be abstract variables, such as time and location, or specific variables such as, object state, or system run time performance variables, such as the CPU utilization, the memory usage, the bandwidth availability and the user waiting time. E is used to describe the system run-time environment.

Definition 2. G= $\{g_0,...,g_m\}$ is a set of goals stating expected state of the environment under the control of system, which could be refined iteratively, so the goal set is a

dynamically changing set. The notation inherits the concepts from goal-oriented requirements engineering language such as, *i** [19] and *Tropos* [2]. The model elements in G are further classified into functional goals and quality of service goals referred as softgoal as in *i**.

Definition 3. $A = \{a_0, \ldots, a_n\}$ is a set of activities the system can perform to change the state of environment variables. The activities are defined in response to the changes of the operating environment and the stakeholders' goals. An activity a_i is used to represent the specific procedures to be carried out by the system.

A special kind of activities are adaptation actions, including, adding/removing/updating a goal, adding/removing/changing an environment variable, adding/removing a new activity, and adding/removing/updating a new production rule. Model elements in red color are evaluated at run-time.

Definition 4. $R = G \times E \times A$ is a set of production rules. A rule is used to describe the mapping relationship among the goal to be satisfied, the environment conditions, and the activities to be carried out. The system uses a rule engine to compare the expected goal state and the run-time state of the environment E. The comparison result will be matched with the conditions on the left-hand side of the adaptation rule. And the other part of the rule is a sequence of adaptation actions to be carried out. For example,

- A decomposition rule *dc:* $g_{1, \ldots, } g_n \rightarrow g_0$ defines that g_i, is a sub-element of g_0, which is used to transform g_0 into a set of sub-goals to be fulfilled.
- A means-ends rule *me:* $a_j \rightarrow g_i$, defines that an activity a_j is a means to achieve g_i.
- A contribution rule *cr:* $a_j \xrightarrow{help} sg_i$ .defines that an activity a_j contributes to the satisfaction of sg_i.

After the four major types of modeling elements in the framework are defined, the next step is to consider how to generate the four kinds of elements for building the run-time requirement model. A Goal-Oriented Requirements Adaptation Method (GORAM) is proposed to build the model of run-time requirements. The method consists of four steps as follows:

- Identify the initial set of goals in G. Starting from the top level goals, and apply decomposition rules to refine into lower level sub-goals or sub-softgoals until they are all refined to certain environmental states which can be monitored when the system is running or they are operationalized as activities.
- Determine the set E of environment variables of the system under design based on the reduction of the goal model. Assume that all these parameters could be monitored and obtained continuously.
- Determine the activities set A. The initial sets of activities of the system are predefined by domain experts and the system architect. The domain experts then envisage the changes of the operating environment and the users' needs, and then define a list of possible adaptation actions.
- The mechanisms for generating the rules set R are different according to the adaptation level of the running system. The following segment illustrates the mechanisms for generating the rules set R.

III. A TYPOLOGY TO SYSTEMS

This section introduces a typology – a theoretical classification framework of different types of systems, with 3 different levels of capability, namely, the static, reactive, adaptive, and a special subclass of adaptive systems - collaborated adaptive systems.

(a) Static requirements scenario:

In the static requirements scenario, the system requirements do not change at run-time. It means that the system responds to the operating environment and users' requests in a fixed and predefined way. In the requirements model <*G, E, A, R*> for static systems, the goal set G is fixed, goal refinement and operationalization is determined at design time. Environment variables associated to the goal model all take constant value. The sets of adaptation actions and rules are both empty. The Figure 1 show the Goal refinement model for static systems as follow:

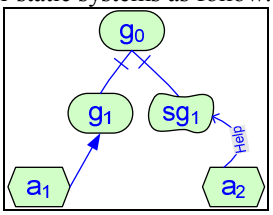

Figure 1. Goal refinement model for static systems

(b) The sensitive/reactive systems scenario:

In the reactive systems scenario, the system does change its behavior at run-time and the system could respond to certain predefined operating environmental changes and users' requests. A goal model G and the environmental parameters set E can be generated according to the GORAM method. Figure 2 shows the goal refinement for reactive systems. The adaptation actions set A and the rules set R are predefined by the domain experts. This kind of systems has some adaptation ability but the ability is limited by a set of predefined rules. And the adaptation is the result of non-deterministic goal refinements at design time. In other words, the achievement of goals depends on the states of the run-time environmental variables. This is an example of gradually adding uncertainty into requirements analysis and system development.

In this case, a new kind of rules is included in the rule base.

- A run-time reaction rule rc: $a_j \xrightarrow{e_i = c_j} g_i$, defines that an activity a_j is a means to achieve g_i, and it is triggered when environment variable e_i takes a certain value c_j.

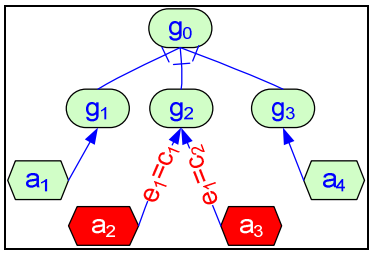

Figure 2. Goal refinement for reactive systems

(c) The run-time adaptation scenario:

In the adaptation scenario, the system requirements changes and the system adaptation actions and the adaptation rules are not completely predefined, there is greater uncertainty at design time. When the model of run-time requirement <*G, E, A, R*> is built, G and E are generated in the same way as in reactive systems. The adaptation actions set A is generated according to the GORAM method. At first the adaptation rules set R only contain some predefined rules. When the system is running, in order to handle the changes of the operational environment and the users' needs, as shown in Figure 3, two new goals, a new environment condition, two new activities and 5 new rules are added to the sets G, E, A and R at run-time respectively.

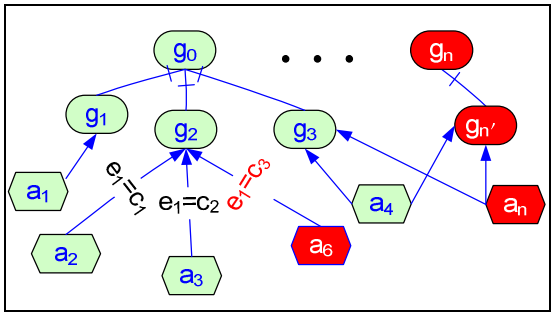

Figure 3. Adaptive systems Goal refinement Model

To allow these changes, the system needs to provide a portal for users to add new goals. At the same time, the system architecture has to allow the incorporation of new operational modules supporting the required new activities. The system data structure has to allow the adding of new environment variables and their monitoring. The system inference engine has to allow the acquisition of new rules.

When the system is running, new adaptation rules can be generated by observing the effect of taking certain actions. After the goal model is built, some new environmental factors are identified and the expected values of the parameters are elicited. The environmental parameters set E contain all these environmental parameters elicited from the goal model. By monitoring the environment, the system could obtain run-time values for these environmental parameters. The system contains a differentiator to compare the run-time states and the expected states of these quality parameters. Based on the comparison result the system can generate an initial sequence of adaptation actions related to the environmental parameters. As the system carries out this initial sequence of adaptation actions, the values of the related environmental parameters would change. Now the system can use the differentiator again to compare the new run-time values and the expected values of these quality parameters. Based on the comparison result the system could evaluate the initial sequence of adaptation actions and continue to evolve until the comparison result is good enough to enable the changes. And then a new rule is generated by the system. The state of the environment parameters is the trigger condition of the new generated rules. While the final evolved sequence of adaptation actions is the adaptation actions. In this way, existing rules in the rules set could also be revised. Figure 4 shows the process of generating new rules. Through the feedback loop the system can refine or revise the rules.

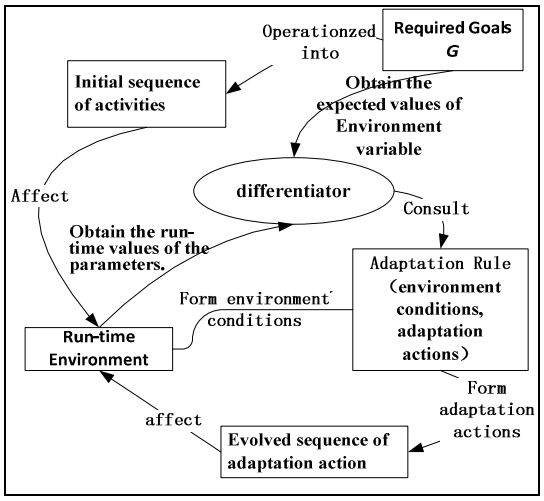

Figure 4. The adaptation architecture of the system

(d) Collaborative services adaptation scenario

An important specific scenario to consider under the category of adaptive system is the collaborative services scenario, where each actor is considered an adaptive system. These actors initiate service goals and provide service actions to each other. At the same time, other actors' service capabilities and needs also become part of the observable environmental variables. The complexity of the model increases, but the theoretical framework remains the same. The run-time changes and adaptation in such system include: the joining or leaving of service actors, the emergence or removal of service goals, capabilities obtained or outsourced through collaboration, etc.

In the collaborative systems' scenario, the goal and the behavior of the actors allow a greater level of uncertainty at design time. When the system is running, new goals and services may emerge. When we build the model of run-time requirements <*G,E,A,R*>, the major differences between the collaborative systems and the general adaptive systems is that the activities in set A could be a service. The system

which depends on other systems will provide service to fulfill their system goals. The processes for generating G, E, R are the same as the run-time adaptation scenario in (c). The minor difference is that now the goals, environmental variables, actions and rules are all associated with an owner actor. And there are delegation and informing activities between actors. The states of actors are partially observable by other actors, and added as environment variables. Figure 5 shows an abstract model for a collaborative system.

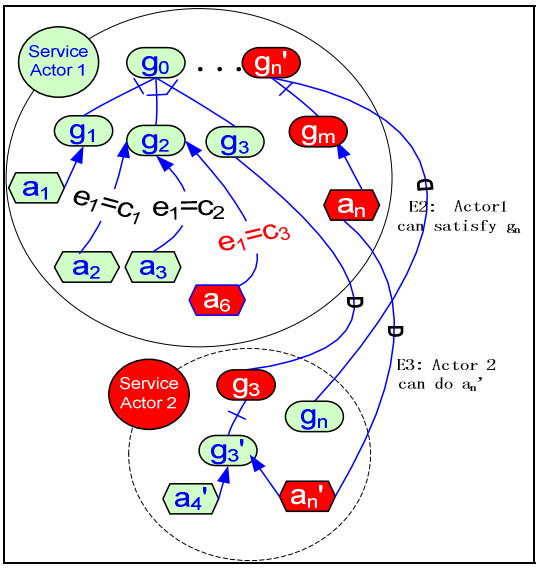

Figure 5. Collabrative Agent-oriented Adaptation Model

IV. TRAFFIC LIGHT CONTROL EXAMPLE

The traffic light control system in [8, 9] is adopted in this paper as an example to illustrate the run-time requirements model and the mechanisms.

"When a section of road is being repaired, it's often necessary to enforce one-way traffic. Half of the road width is used for traffic and half for the repair work. The traffic is controlled by a pair of simple portable traffic light units contain a Stop light and a Go light. Each end of the one-way section equipped with a light unit connected to a small computer that controls the sequence of the light. The computer controls the lights by emitting RPulses and GPulses, to which the units respond by turning the lights on and off."

The problem diagram for this problem is as Figure 6:

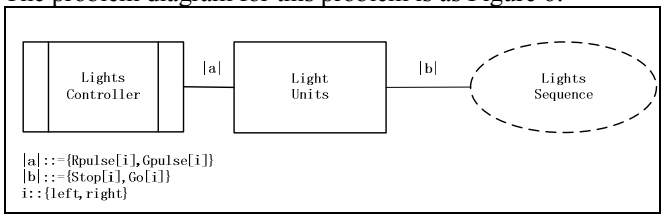

Figure 6. PF problem diagram for static traffic light [8,9]

A. *Use the GORAM to model the static traffic light system*

First, we build the goal model. Through analysis we could summarize the top level goal of the system as "one-way working". And then, the top level goal needs to be decomposed into lower levels to form the goal model G-static in Figure 7 as follows:

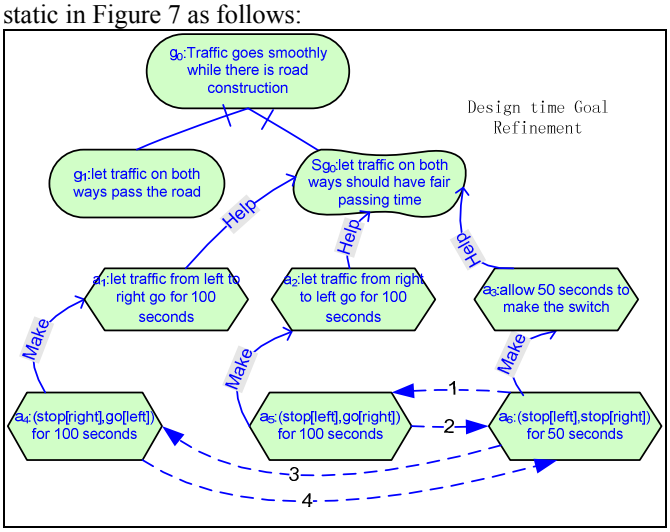

Figure 7. Goal refinement model for static systems

And the simple one-way traffic lights system regime repeats a fixed cycle of four phases. First, for 50 seconds, both units show "Stop"; then, for 100 seconds, one unit shows "Stop" and the other "Go"; then for 50 seconds both show "Stop" again; then for 100 seconds the unit that previously showed "Go" shows "Stop" and the other shows "Go". The cycle could be the sequence as follows: {(Stop[left], Stop[right]) for 50 seconds; (Stop[left], Go [right]) for 100 seconds; (Stop[left], Stop [right]) for 50 seconds; (Go[left], Stop[right]) for 100 seconds} described in the Figure 8. Then the cycle is repeated.

```
(Stop[left],Stop[right]) for 50 seconds;
(Stop[left],Go[right]) for 100 seconds;
(Stop[left],Stop[right]) for 50 seconds;
(Go[left],Stop[right]) for 100 seconds
```

Figure 8. Example actions sequences [8]

B. *Model the static and reconfigured traffic light system*

The regime of the system may be different when we move the system to a road repair on a hill. Assume that the right side is higher than the left side. The repetition of the sequence is {(Stop[left], Stop[right]) for 50 seconds; (Stop[left], Go[right]) for 100 seconds; (Stop[left], Stop[right]) for 75 seconds, (Go[left], Stop[right]) for 150 seconds}.

The system is reconfigured during the deployment by considering environmental changes. The system environment is not monitored during running, but the deployment team needs to. In this case, the time duration of the traffic light could reset. Thus, the E set in the requirement model remains to be empty. The action set of the system should be that each pair of the traffic lights could have the Go or Stop light on.

The system does not respond to the requirements changes at run-time, so the adaptation rules remains empty. The four elements of the run-time requirement model <G, E, A, R> are as follows:

TABLE I. RUN-TIME REQUIREMENT MODEL FOR THE SIMPLE ONE-WAY TRAFFIC LIGHTS SYSTEM

Run-time requirement model element	Value of the element
G: goal model	G-static model
E:environmental parameters	{time}
A: activities	{Stop[left];Stop[right]; Go[left];Go[right]};
R: adaptation rules	{}

C. *Use GORAM to model the reactive traffic light system*

For example, "consider a more ambitious version of the one-way traffic lights system - a sensitive one-way traffic lights system. The light units are chosen for a new product that will be both safer and more efficient. The light units equipped with road sensors to monitor the passage of vehicles at the ends of the controlled stretch of the road where the lights units are placed. So it may be possible for the system to tune the time of the lights according to the traffic conditions."[8] The problem diagram for this problem looks like Figure 9 as follows:

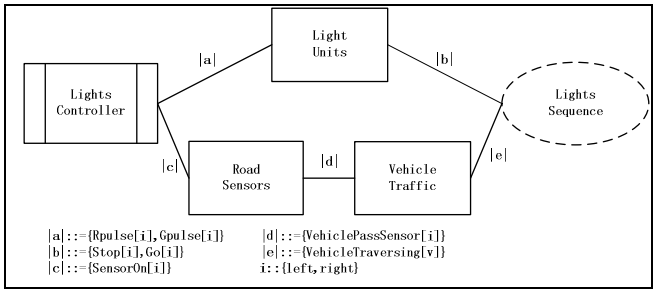

Figure 9. Sensitive One-Way Traffic Lights Problem Diagram [8]

We use the GORAM method to model this kind of system. The top level goal should also be the "one-way working." However, when we decompose the top level goal into lower levels, there are some differences from the G-static goal model. In the sensitive one-way traffic light system we focus more on the high efficiency of the system. The system aims to shorten the average traffic waiting time. The goal model G-reactive for the sensitive one-way traffic lights system is described in Figure 10.

With the goal model G-reactive we could elicit some environment parameters to monitor such as waiting-time [left]: the waiting time of the vehicles from left to right and waiting-time [right]: the waiting time of the vehicles from the right to the left. The value of the waiting-time is in the range [0 sec, 300 sec]. The waiting-time is zero, which means that the vehicles are running and the maximum value of the waiting-time is 300 seconds. There are two other environmental states need to be monitored. The two states are waiting-vehicles [left]: whether there are vehicles waiting in the left side or not, and waiting-vehicles [right] for the right side. The waiting-vehicles parameter is a Boolean variable. The true value means that there are vehicles waiting. The action set of the system remains the same {Stop[left] ; Stop[right]; Go[left]; Go[right]}.

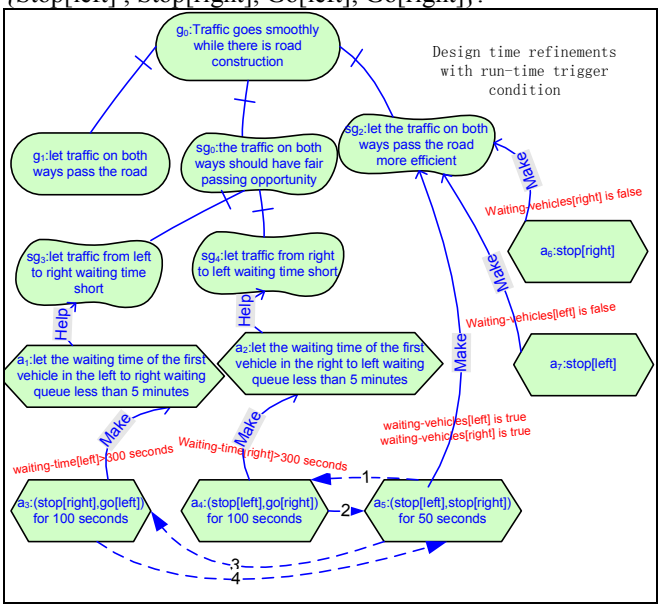

Figure 10. Sensitive One-Way Traffic lights Goal Model

Table II shows the four elements of the run-time requirement model <G, E, A, R> as follows:

TABLE II. RUN-TIME REQUIREMENT MODEL FOR THE REACTIVE ONE-WAY TRAFFIC LIGHTS SYSTEM

Run-time requirement model element	Value of the element
G:Goal model	{G-reactive model}
E:environmental parameter set	*{waiting-time[left];waiting-time[right]; waiting-vehicles[left];waiting-vehicles[right];*
A:activities	*{Stop[left] ;Stop[right]; Go[left];Go[right]}*
R:adaptation rule set	{R1(env: waiting-vehicles[left] is false, activity: Stop[left]=on, Go[left]=off;) R2(env: waiting-vehicles[right] is false, activity: Stop[right]=on, Go[right]=off;) R3(env: waiting-time[left]>300, activity: Stop[left]=off, Go[left]=on, Stop[right]=on, Go[right]=off;) R4(env: waiting-time[right]>300, activity: Stop[left]=on, Go[left]=off, Stop[right]=off, Go[right]=on;) R5(env: waiting-vehicles[left] is true, waiting-vehicles[right] is true, activity: Stop[left],Stop[right] for 50 seconds. Stop[left],Go[right] for 150 seconds. Stop[left],Stop[right] for 50 seconds. Go[left],Stop[right] for 150 seconds.)}

As the traffic light control system develops, the traffic light control system could monitor the traffic situation and respond to the users' requests. When we use the GORAM to build the run-time requirement model *<G, E, A, R>*, define a new high level softgoal, such as "the traffic light control system should be effective". And then we decompose the high level softgoal into lower level sub-softgoals such as

"the waiting time of the passerby from south to north should be less than 5 minutes", "the weather condition should be considered" and so on. During the decomposition process, we elicit some quality parameters such as the "waiting time of the passerby from south to north" and the expected value of less than 5 minutes and "the weather condition". These parameters could be monitored by the system, so the quality parameters such as "waiting time of the passerby from south to north and the weather condition" are added in the quality parameters set E. The adaptation actions set A and the adaptation rules set R are predefined by the domain experts. After building the run-time requirement model<*G, E, A, R*>, the system will use the differentiator to compare the expected value and the run-time value of this quality parameter, and then according to the adaption rules to choose a sequence of adaptation actions to carry out.

While the one-way traffic light system becomes more intelligent, the traffic light system has more ability to learn new adaptation rules to adjust to the environment changes. Use GORAM method to build the run-time requirements model for the intelligent traffic light system. The difference between the goal model G-runtime and the G-reactive is goal decomposition. The most important target of the intelligent system is to make the system more effective. Figure 11 below describes the goal model G-runtime for the intelligent one-way traffic lights system.

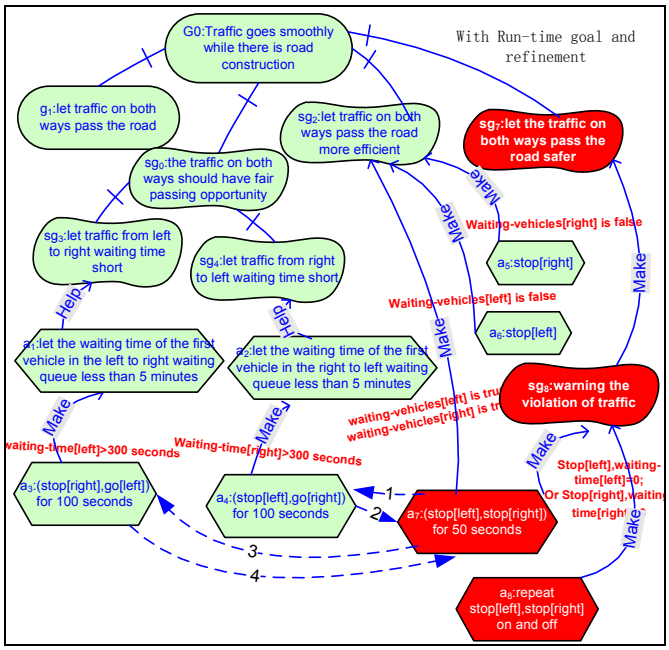

Figure 11. Run-time adaptation One-Way Traffic lights Goal Model

The environment parameters E and the action set A are the same as the sensitive one-way traffic light system. Waiting time should be considered as a new environment parameter added in the E set. However in the intelligent one-way traffic light system, the rules set is self-adaptive at run time. New rules can be added into the rule set R. At first the rules set in the sensitive one-way traffic light system is contained in the intelligent one-way traffic light system. Then based on run-time feedback, new rules could be added into the rules set such as the rule R6 below.

Consider a scenario such as the following: on the left road entry, Stop sign is on, while on the right hand side, Go sign is on. There are vehicles entering the road from right hand side. However, there is a vehicle ignoring the stop sign and entering the road. If we add a safety goal in the set G in the system requirements model, it then can be further refined into a sub-goal named "give warning sign when there are accidents or abnormities".

In set E, there is an environmental constraint "waiting time for the first car which has right should be 0".

In set A, when the sensor detects the above situation, the warning action can be:

1. By turning on and off the red light on both sides, it can be shown that there is an emergency in the section of the road, and no vehicle can enter the road any more. A new action is added, satisfies a new goal.
2. Use the past action strategy to solve this new problem, in other words, to let the stop sign on on both sides. In this case, a new rule is added, which connects an existing action to a new goal.

R: run through the steps defined in figure 4,

a. Execute the initial actions;
b. Compare the execution results with expected value.
   The expected goal model state is:
   Stop light is on, and no vehicle is passing.
   The observed environmental state is:
   Stop sign is on, but there is a vehicle entering the road.
c. Based on the learning procedure and the evaluation mechanism, the action sequence can be evolved to generate new action series.
d. By executing the action sequence, the differentiator will conduct another round of comparison and evolve iteratively until there is a good enough solution satisfies the goal. Thus, a new rule is generated.

R6 :(condition: Stop[left],waiting-time[left]=0;
Or Stop[right],waiting-time[right]=0
adaptation: repeat stop[left],stop[right] on and off)

Table 3 shows the four elements of the run-time requirement model <G,E,A,R> as follows:

TABLE III. RUN-TIME REQUIREMENT MODEL FOR THE INTELLIGENT ONE-WAY TRAFFIC LIGHTS SYSTEM

Run-time requirement model element	Value of the element
G:Goal model	{G-runtime model}
E:environmental parameter set	{waiting-time[left];waiting-time[right]; waiting-vehicles[left]; waiting-vehicles[right] ; }
A:activities set	{Stop[left];Stop[right]; Go[left];Go[right]};
R:adaptation rule set	{R1,R2,R3,R4,R5, in the sensitive one-way traffic light system;R6 mentioned above and new rules generated .}

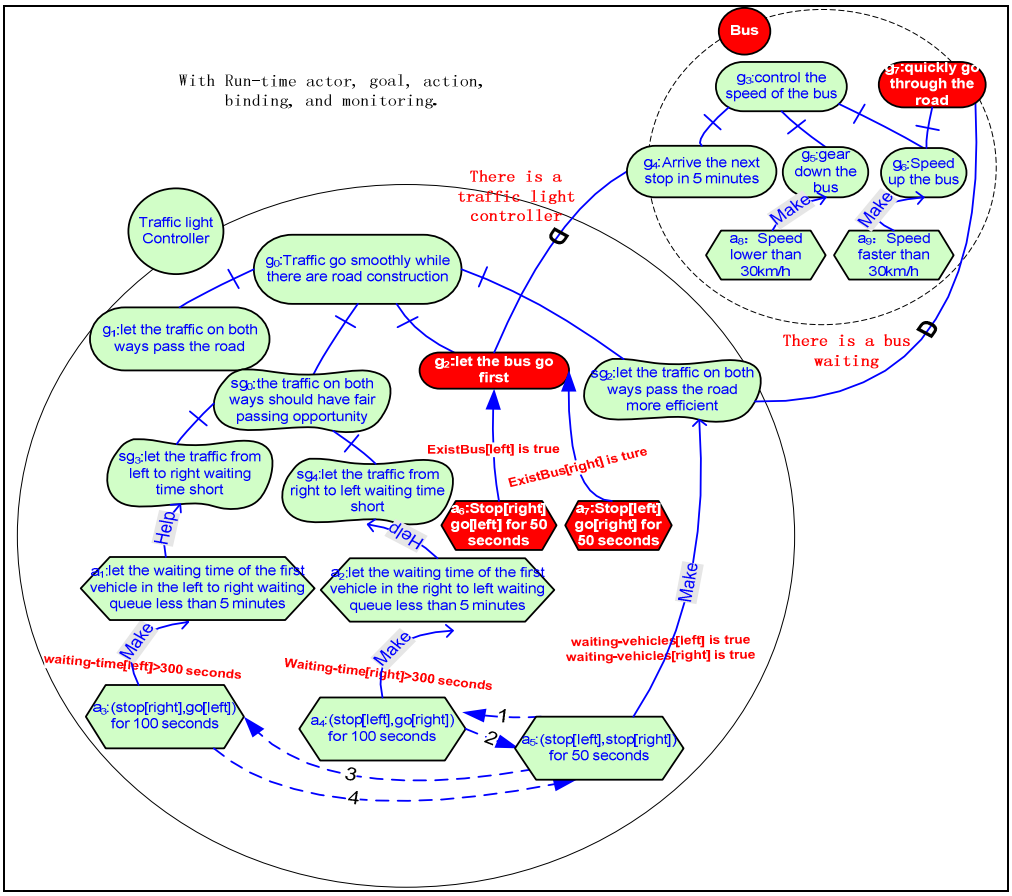

Figure 12. Run-time adaptation for multi-agent systems

D. *Use GORAM to model collaborative traffic light system*

Consider a more intelligent version of the one-way traffic lights system. At first, the goal model of the one-way traffic lights system is the same as the run-time adaptation scenario model. Consider that now the traffic light controller is aware of the fact that a public transit bus is about to enter the road and they need certain interactions with each other. The top level goal of the bus is to "arrive in the next Stop in 5 minutes" and use GORAM to model the run-time situation. Figure 12 shows the relationship between the on-board controller of the bus and the one-way traffic lights system.

Use GORAM method to build the run-time requirements model for the collaborative traffic light system. The difference between the goal model G-collaborative and the G-runtime is the insertion of new actor Bus and its goals. The environment parameters E and the action set A are updated to reflect these changes. New environment parameters are added in the E set including whether there are buses waiting. In the collaborative traffic light controller, the rule set is self-adaptable at run time in accordance with the other actors in the surrounding environment. In the collaborative agent's scenario, there would be further differentiations within this category – whether agents have knowledge about each other, what kinds of knowledge, the awareness of the existence of each other, or the model of each other. This will lead us into the well-developed research discipline of multi-agent models in AI.

V. DISCUSSIONS

The Goal-oriented run-time requirements model proposed in this paper aims to provide a goal-oriented interpretation to run-time requirements changes and system adaptation. To allow for run-time requirements changes and system adaptation, system needs to maintain a goal refinement model during execution, in which the set of user goals, the set of environmental variables, the set of possible system actions and production rules connecting the previous three sets can evolve dynamically. There are four typical scenarios defined in this paper.

The first scenario refers to systems that do not change at run-time. In this case, the goal refinement model is fixed, the action set and production rules manipulating the goal refinement are also predetermined.

The second scenario refers to systems that "seem to" react to environmental changes. In this case, the goal refinement is also predefined, but there are certain actions taking environmental states as an input or trigger.

The third scenario refers to adaptive systems, that may designed to satisfy certain goals, but later, when a new goal

emerges, the system will be able to find new actions or action series to operationalize it, or find new production rules to map the goals of existing actions, or add new environmental variables to be monitored, which triggers the new identified refinement.

The fourth scenario refers to a specific case of adaptive system that is composed of collaborative agents. Each of these agent has an associated goal-refinement structure, these goal refinement structure are interconnected through run-time service request, publication, search, binding, delegation and revoking relationships.

Related work on requirements for running system tackles the problem from several different angles [7, 10, 14]. In [13], Baresi et al have proposed the concept of live goal, and uses extended KAOS models to support service composition. In [17, 18], Qureshi et al have proposed using context ontology to annotate goal models, and capturing adaptive requirements that entail monitoring and adaptation behaviors. Most closely related are the ones also taking a goal-oriented approach to support run-time adaptation [6]. Lapouchnian et al. [12] propose using goal models as the core knowledge to support autonomic software design, autonomic behaviors are interpreted as goal-model applications, goal monitoring, reasoning and goal-to-design translations. Our approach differs in that variability at the system run-time behavior level is considered, but not limited to predefined options. Qureshi and Perini in [18] proposed engineering adaptive service requirements with *Tropos*. In their approach, goal-refinements are still handled at design time, and run-time adaptation are interpreted as handling service selection, lookup, request acquisition and update of requirements specification. The goal refinement process is intertwined with service modeling and request handling, so there are certain steps of goal-refinements decisions can be deferred till run-time, when the context information and operational environment are better understood. This agrees with what has been mentioned in [3], this paper is taking an online goal refinement approach. Furthermore, such online goal refinement processes are conducted by all participating agents in parallel, which lead to continuous changes and adaptations in the social and computing environment.

In the future, the system architecture, the different types of rules in the rule base storing knowledge for run-time use, and the monitoring mechanism for collecting the environmental parameters and the possible adaptation strategies in response to the run-time emerging needs will be implemented and examined. The management of graphical goal models, in particularly, the scalability and modularity improvement such as in [4] will also be incorporated.

ACKNOWLEDGMENT

Financial support from the National Natural Science Foundation of China (Grant No.60873064), the National Basic Research and Development 973 Program (Grant No.2009CB320700), and the Key Project of National Natural Science Foundation of China (Grant no. 90818026), National HeGaoJi Key Project No. 2009ZX01045-001-001-02 are gratefully acknowledged.

REFERENCES

[1] Bihari, T.E. and K. Schwan, Dynamic adaptation of real-time software. ACM Trans. Comput. Syst., Vol.9, No.2, 1991. p. 143-174.

[2] Castro, J., Kolp, M. and Mylopoulos., J. Towards Requirements-Driven Information Systems Engineering: The Tropos Project. Information Systems, Volume 27, Number 6, September 2002.

[3] Cheng, B. H. C., H. Giese, P. Inverardi, J. Magee, and R. de Lemos. Software Engineering for self-adaptive systems: A research road map. In B. H. C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, eds. Software Engineering for Self-Adaptive Systems, Vol. 08031 of Dagstuhl Seminar Proceedings. Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2008.

[4] Fernanda Alencar, Márcia Lucena, Carla Silva, Emanuel Santos, and Jaelson Castro. Improving the Modularity of i* Models. iStar 2010 − Proceedings of the 4th International i* Workshop. June, 2010.

[5] Fickas, S. and M.S. Feather. Requirements monitoring in dynamic environments. in Proceedings of the Second IEEE International Symposium on Requirements Engineering, IEEE Computer Society. p. 140.

[6] Goldsby, H. and B.H.C. Cheng. Goal-Oriented Modeling of Requirements Engineering for Dynamically Adaptive System. in Proceedings of the 14th IEEE International Requirements Engineering Conference, IEEE Computer Society. p. 338-339.

[7] Heineman, G.T. A Model for Designing Adaptable Software Components. in Proceedings of the 22nd International Computer Software and Applications Conference, IEEE Computer Society. p. 121-127.

[8] Jackson, M., et al., The Real World. Millennial Perspectives in Computer Science, Vol., 2000. p. 157-173.

[9] Jackson, M., Problem frames: analyzing and structuring software development problems. 2001: Addison-Wesley Longman Publishing Co., Inc. 390.

[10] Johnson, B., et al., Flexible Software Design – Systems Development for Changing Requirements. 1st edition. 2005, Boca Raton, FL, USA: Auerbach Publications, Taylor &Francis Group.

[11] Jureta, I.J., et al. Dynamic Requirements Specification for Adaptable and Open Service-Oriented Systems. in Proceedings of the 5th international conference on Service-Oriented Computing. Vienna, Austria, Springer-Verlag. p. 270-282.

[12] Lapouchnian A., S. Liaskos, J. Mylopoulos, Y. Yu. Towards Requirements-Driven Autonomic Systems Design. In Proc. ICSE 2005 Workshop on Design and Evolution of Autonomic Application Software (DEAS 2005), St. Louis, Missouri, USA, May 21, 2005. ACM SIGSOFT Software Engineering Notes 30(4), July 2005.

[13] Luciano Baresi and Liliana Pasquale. Live goals for adaptive service compositions.In proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems.2010.

[14] McKinley, P.K., et al., Composing adaptive software. Computer, Vol.37, No.7, 2004. p. 56-64.

[15] Mieczyslaw M. Kokar, Kenneth Baclawski, and Yonet A. Eracar. Control Theory-Based Foundations of Self-Controlling Software. IEEE Intelligent Systems, 1999.

[16] Mikkonen, T., et al. Managing software evolution with the service concept. in Principles of Software Evolution, 2000. Proceedings. International Symposium on. 2000.

[17] Nauman A. Qureshi and Anna Perini. Engineering Adaptive Requirements. 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems.2009.

[18] Qureshi, N.A. and A. Perini. Requirements Engineering for Adaptive Service Based Applications. in The 18th IEEE International Requirements Engineering Conference. 2010. Sydney, Australia.

[19] Yu, E. Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering. Proceedings of the 3rd IEEE International Symposium on Requirements Engineering (RE'97) Jan. 6-8, 1997, Washington D.C., USA. pp. 226-235.