Achieving Adaptation for Adaptive Systems via Runtime Verification: A Model-Driven Approach

Zhuoqun Yang¹, Zhi Jin², Zhi Li³

¹Institute of Mathematics, Academy of Mathematics and Systems Science, Chinese Academy of Sciences, Haidian Dstr., Beijing 100190, P. R. China
²Key Laboratory of High Confidence Software Technologies (MoE), Peking University, Haidian Dstr., Beijing 100871, P. R. China
³Software Engineering Dept., College of Computer Science and Information Technology, Guangxi Normal University, Guilin, Guangxi 541004, P. R. China

Abstract
Self-adaptive systems (SASs) are capable of adjusting its behavior in response to meaningful changes in the operational context and itself. The adaptation needs to be performed automatically through self-managed reactions and decision-making processes at runtime. To support this kind of automatic behavior, SASs must be endowed by a rich runtime support that can detect requirements violations and reason about adaptation decisions. Requirements Engineering for SASs primarily aims to model adaptation logic and mechanisms. Requirements models will guide the design decisions and runtime behaviors of system-to-be. This paper proposes a model-driven approach for achieving adaptation against non-functional requirements (NFRs), i.e. reliability and performances. The approach begins with the models in RE stage and provides runtime support for self-adaptation. We capture adaptation mechanisms as graphical elements in the goal model. By assigning reliability and performance attributes to related system tasks, we derive the tagged sequential diagram for specifying the reliability and performances of system behaviors. To formalize system behavior, we transform the requirements model to the corresponding behavior model, expressed by Label Transition Systems (LTS). To analyze the reliability requirements and performance requirements, we merged the sequential diagram and LTS to a variable Discrete-Time Markov Chains (DTMC) and a variable Continuous-Time Markov Chains (CTMC) respectively. Adaptation candidates are characterized by the variable states. The optimal decision is derived by verifying the concerned NFRs and reducing the decision space. Our approach is implemented through the demonstration of a mobile information system.

Keyword: Self-adaptive software, model-driven approach, non-functional requirements, verification, probabilistic model checking.

1. Introduction
The self-adaptive system (SAS) is a novel computing paradigm in which the software is capable of adjusting its behavior in response to meaningful changes in the environment and itself [1]. The ability of adaptation is characterized by self-* properties, including self-healing, self-configuration, self-optimizing and self-protecting [2]. Innovative technologies and methodologies inspired by these characteristics have already created avenues for many promising applications, such as mobile computing, ambient intelligence, ubiquitous computing, etc.

Modern software systems interact with other systems, devices, sensors and people intensively. Such an operational environment may be inherently changeable, which makes self-adaptiveness become an essential feature. Context can be defined as the reification of the environment [3] that is whatever provides as a surrounding of a system at a time. It provides a manageable and manipulable description of the environment. Context is essential for the deployment of self-adaptive software. As the environment is changeable, the context is unstable and ever changing and the system is desired to perform different behaviors accordingly.

Requirements Engineering (RE) for SASs primarily aims to identify adaptive requirements, specify adaptation logic and build adaptation mechanisms [4]. At this stage, both functional requirements (FRs) and non-functional requirements (NFRs) [5] are captured within well-formed requirements models. Conducting context analysis at requirements phase will be worthwhile at the design and development phases, because contexts may influence the decisions about what to build and how to build them.

Research challenges of RE for SASs are composed of several aspects, including modeling adaptation mechanism, dealing with uncertainty, achieving adaptation and requirements verification [4]. Many research works in the literature have shown remarkable progress in providing solutions to these challenges. For modeling adaptation mechanism, Qureshi et al. [6] provide a requirements modeling language with diagrammatic syntax for SASs. Cheng et al. [7] introduce a goal-based modeling approach to develop the requirements for dynamically adaptive systems, while explicitly factoring uncertainty into the process and resulting requirements. For dealing with requirements uncertainty, a research agenda is provided in [8]. The author argues requirements for self-adaptive systems should be viewed as runtime entities that can be reasoned over in order to understand the extent to which they are being satisfied and to support adaptation decisions that can take advantage of the systems’ self-adaptive machinery. FLAGS [9] is proposed for mitigating the requirements uncertainty by extending the goal
model with adaptive and fuzzy goals. RELAX [10] is a formal requirements specification language, which is defined in terms of temporal fuzzy logic, describing relaxed requirements and the environment. For achieving adaptation, FUSION [11] uses online learning to mitigate the contextual changes in context and reconfigure features from feature pool to unanticipated changes. The method utilizes linear equations to simulate the system for reducing time complexity of decision making. POISED [12] improves the quality attributes of a software system through reconfiguration of components to achieve a global optimal configuration for the software system. Besides, some other approaches to achieving adaptation are based on reasoning with goal model [13, 14]. For requirements verification, Goldsby et al. [15] provide AMOEBA-RT, a run-time monitoring and verification technique that provides assurance that dynamically adaptive software satisfies its requirements. Filieri et al. [16] provide a model checking based approach for verification, in which reliability models are given in terms of Discrete Time Markov Chains which are verified against a set of requirements expressed as logical formulae. Epifani et al. [17] lay the foundations for an iterative model-driven development, which aims at verifying that an implementation satisfies non-functional requirements. In their approach, if the resulting running system behaves differently from the assumptions made at design time, the feedback to the model shows why it does not satisfy the requirements and lead to a further development iteration. Ghezzi et al. [18] put forward the proposal of quantitative verification at runtime for self-adaptive service-based software. The same author proposes a model-based approach that enables software engineers to assess their design solutions for software product lines in the early stages of development [19].

During implementing self-adaptation, two concerns that are lacking in discussing should be put forward: reliability and performance. Firstly, as discussed earlier, contexts are intrinsically changing which may cause the violation of the system-to-be. For example, in mobile computing systems domain, the precision of locating with GSM is highly dependent on the status of network. The higher bandwidth is, the more precise location the system will get. In other words, the low bandwidth will affect the reliability of GSM. Hence, self-adaptive systems need to be built with the mechanism of monitoring contextual changes, analyzing violations of NFRs and making adaptation decision. Secondly, performance of SASs is another essential concern for SASs, especially when systems are supposed to be deployed in the time/resource-limited environment. System performance comes with several dimensions, such as utility, time costs and energy costs. Specifically, we use utility to describe the convenience of system behaviors for users. For example, in mobile computing domain, the utility of searching by voice is higher than searching by text, because the interaction between system and users in the former scenario makes searching more convenient. Time costs intuitively refer to the time consumed by performing certain system behavior, e.g. monitoring contexts and searching the data base, while energy costs describe the energy consumed by these tasks. Thus, when implementing the adaptation, we should also consider these performance dimensions.

In this paper, we provide a model-driven approach to achieving self-adaptation from the requirements engineering perspective. Modeling is considered as an important conceptual tool in RE stage of software development. Requirements models are built to better understand and reason about the qualities a system should exhibit in order to fulfill its goals. They are also used to support systematic interaction with the stakeholders in requirements elicitation and to help crystallize design decisions and evaluate the trade-offs among them. To modeling adaptation mechanisms, we build adaptation goal model by extending traditional goal model with new modeling elements representing the monitor-analyze-plan-execute process, which is known as MAPE control loop [20]. Then, the reliability and performance are specified as tags to each system task. Particularly, to formally describe reliability of SASs, we consider the uncertainty of system tasks, which can be quantitatively expressed with the failure rate of each task. Different probability distribution will lead to different system reliability. Thus, we choose to use Bayesian methods for reasoning with these probabilities and performance constraints. To this end, we investigate the system behavior from the requirements model by applying several operators adopted from Communicating Sequential Processes (CSP). With the well-formed transformation relations and algorithm, the requirements model can be expressed with the corresponding processes and behavior model, which displays as a Labeled Transition System (LTS). We use the Process Analysis Toolkit (PAT) to verify the properties of the generated process-based LTS and the results imply the reasonability and consistency of the transformation process. Thereafter, the tagged adaptation goal model and the LTS are integrated into probabilistic Markov models, i.e. the Discrete-Time Markov Chains (DTMCs) and Continuous-Time Markov Chains (CTMCs). The non-functional properties are specified with Probabilistic Computation Tree Logic (PCTL). We propose to achieve structural adaptation through verifying PCTL properties with DTMCs and to derive parametric adaptation through verifying Reward properties with CTMCs. The verification processes are implemented through model checking within PRISM. An application from the mobile computing domain is leveraged as illustration throughout the approach.

Our contributions are multifold. First is that we present a general method for describing adaptation mechanisms within requirements models of SASs. Compared with the related work, our model is more concise but easier to be manipulated. The well-formed structure provides the possibility of generating system behavior model. Second, we propose a flexible notation for specifying the reliability and performance of SASs, which are uncertain and varying according to related contexts. These notations can help characterize adaptation behaviors. Third, the approach provides processes for transforming requirements model to corresponding behavior model, which can be applied to SASs of any other domains that are represented with goal models. It can be leveraged as the basis for behavior analysis in following stages of software development. Fourth, the approach presents a step-by-step way to deriving adaptation decisions among alternative candidates. Requirements are consid-
ndered as runtime properties and the verification-based adaptation is conducted based on strict mathematical foundations. Last but not least, the whole approach is driven by requirements models and behavior models. Thus, the approach bridges the gap between static and dynamic attributes of SASs and the gap between adaptation logic and adaptation behavior.

The paper is structured as follows. Section 2 illustrates the motivating example and the overall approach, in order to deliver an easy understanding of this paper. Section 3 introduces the overview of the proposed approach. The concepts and modeling notations of adaptation goal model are presented in Section 4. Section 5 provides the tagged adaptation goal model by specifying reliability and performance characteristics. Section 6 presents the sequential adaptation goal model with CSP operators and transformation to the behavior model. Reliability-related behavior model (DTMCs) is presented in Section 7, while Performance-related behavior model (CTMCs) is described in Section 8. Section 9 provides the evaluation of our approach with the motivating example. Related works are discussed in Section 10, followed by conclusion and future work in Section 11.

2. Motivating Example

To illustrate the proposed approach, we consider adopting the push notification technology from the mobile computing domain. Typically, pushing notifications is a technique used by apps to alert smartphone owners on content updates, messages, and other events that users may want to be aware of. This technique has been successfully developed as API on iOS systems, such as Prowl (http://www.prowlapp.com) and Pushover (https://pushover.net). These applications focus on receiving the needed information timely, no matter where the user is. However, in the location-related scenarios, the application’s performance tightly related to the user’s location, e.g. learning application on smartphone [21, 22].

![Figure 1 Goal Model of Ambient business community MobIS](image)
The ambient business community Mobile Information System (MobIS) is the software deployed on smartphones to support pushing business information to nearest customers in a smart business community. The objective of pushing business information is to notify users of surrounding information and events, such as the goods on sale, goods’ description, comments of customers, etc. Figure 1 presents a requirements model built with strategic rationale (SR) diagram in i* framework [23], which is defined to model and reason about both the system and its organizational environment.

The top goal of MobIS is user gets the ambient business information, which can be decomposed into two subgoals with Decomposition-link: user info is recorded and user has access to business info. The Decomposition-links describe both AND-relation and OR-relation, which are displayed as and/or notations. user info is recorded is further decomposed into user is registered and user info is prepared. The goal user is registered can be achieved through two tasks: register by SMS and register by Internet. The Means-Ends links are used to provide alternative operational tasks for the target goal. User information consists of two parts: location information and preferences information. The former one aims to recognize user’s location (by GPS or GSM) while the later one identifies personal interests. Users can access business information in either online mode or offline mode. For online mode, MobIS sends request timely for information of either media form or text form. To receive information, MobIS needs to load data into buffer and then data persists into database. For offline mode, user can search local database either by voice or text. Besides, user can filter the retrieved information. Four softgoals that capture NFRs need to be concerned at runtime: High reliability, Low energy costs, Low time costs and High Utility. Actually, each task has different contribution to these softgoals. The contribution-links in Figure 1 are partial contribution relations. The quantitative descriptions of these contributions and instantiations of NFRs will be provided in Section 7 and Section 8.

Two environment actors are considered in this example. Network describes the external environment that changes automatically, with the task of show bandwidth to sensors of outside systems. Memory is the internal environment whose value can be tuned according to adaptation decisions, with the task of prepare spared memory for self-adaptation. MobIS needs network to provide bandwidth. This relation is described by Task-dependency. While MobIS needs claims memory from memory pool. It is presented as Resource-dependency.

3. Approach Overview

![Figure 2 Generating relations between models in our model-driven approach]

Models are the core concepts and artifacts in our model-driven approach. The approach contains several steps and each step generates the corresponding model. The generating relations between models are described in Figure 2. Models in the line-blocks capture static characteristics of SASs while models in the dash-blocks describe dynamic characteristics of SASs. The concrete steps are stated as follows.

**Building Adaptation Goal Model (AGM)** is the initial step of our approach. Traditional goal model only capture business logic (Figure 1), while goal model of SASs needs to be endowed with adaptation logic. We build the adaptation mechanism adopted from MAPE loop by adding incremental tasks to traditional tasks that need to be reconfigured at runtime. In this step, we consider both modeling structural adaptation logic and parametric adaptation logic.
Building Tagged Adaptation Goal Model (TAGM) is conducted based on the completion of AGM. In this step, tags of adaptation concerns, i.e. reliability and performance, are described within AGM. These tags will be used to compute the adaptation decisions at runtime.

Building Sequential Adaptation Goal Model (SAGM) focuses on providing an efficient way to transforming goal model to corresponding behavior. SAGM is generated by adding several sequential operators to AGM, which are adopted from CPS. By this means, we explore the transformation patterns of different decomposition patterns of goal model. These transformation relations are used as the basis of deriving behavior model.

Transforming to Behavior Model aims at transforming the system goal model to system behavior by using the above transformation patterns. The derived behavior model is expressed as a Labeled Transition System (LTS). Business logic can be verified with the generated LTS, which confirms the reasonability and consistency of the transformation process.

Generating Reliability-related Model integrated reliability tags of TAGM with LTS. The resulting model is expressed as a variable DTMC, which can be used for Bayesian reasoning with the transition probabilities.

Generating Performance-related Model merges performance tags of TAGM with LTS. This step is completed by producing a Rewarded variable CTMC, which can be used for Bayesian reasoning with the transition rate.

Runtime verification is the last step of our approach. Reliability requirements are expressed as PCTL properties, which can be verified within the DTMC. Performance requirements are described as CSL properties, which can be verified within the CTCM.

4. Modeling Adaptation Logic

This section presents the concept and modeling elements of AGM. Then we describe three kinds of adaption scenarios and discuss how to model adaptation mechanisms for each scenario based on the proposed modeling notations.

4.1 Adaptation Goal Model

Definition 1 (Adaptation Goal) Adaptation goals refer to the goals that are related to adaptation requirements and should be achieved with adaptation tasks.

Definition 2 (Adaptation Task) Adaptation tasks refer to the tasks that should be implemented through making adaptation decision.

Definition 3 (Adaptation Goal Model) Adaptation Goal Model is defined as

\[ AGM = (G, AG, SG, T, AT, M, A, P, E, RM, R_D, R_C) \]

where \( G \) is the set of ordinary goals, \( AG \) is the set of adaptation goals, \( SG \) refers to the set of softgoals, \( T \) refers to the set of ordinary tasks, \( AT \) refers to the set of adaptive tasks. \( M, A, P \) and \( E \) consists of the tasks that perform monitoring, analyzing, planning and executing correspondingly, which characterize the adaptation mechanisms. \( RM: G \times T \cup AG \times AT \) is the extended Means-Ends relation. \( R_D: G \times G \cup T \times T \cup AT \times \{M, A, P, E\} \) is the extended Decomposition relation. \( R_C: T \times SG \cup \{M, A, P, E\} \times SG \) is the extended Contribution relation. These modeling elements and relations are comprised in the metamodel presented in Figure 3.

![Figure 3 Metamodel of adaptation goal model](image)

4.2 Adaptation Mechanism

Based on the modeling elements proposed above, we discuss how to modeling adaptation logic and mechanisms within the requirements model. To this end, we first describe the adaptation scenarios considered in our approach, then for each scenario, we model the related adaptation mechanism.

Recall the self-* properties introduced in Section 1, which includes self-healing, self-configuration, self-optimizing and self-protecting. In our approach, we consider the properties of self-healing, self-configuration and self-optimizing. For self-protecting, readers can refer to [24] for more details about self-protecting software systems. For each of the former three self-* properties, we explore the adaptation scenario (AS) related with reliability and performance. Table 1 shows the causes, effects and countermeasures in each scenario. Notice that AS1 describes the structural adaptation while AS2 describes the par-
ametric adaptation. Differences of the two are introduced in [25]. However, in this paper, structural adaptation is discussed in a broader sense, which means the adaptation is implemented based on alternative task structures, either by configuring the specific components, e.g. GPS or GSM, or by configuring the optional tasks, e.g. require information in media form or text form. Another characteristic that should be noticed is that for an optimal adaptation decision, self-configuration and self-optimizing are achieved simultaneously.

For each scenario, an adaptation goal and corresponding adaptation mechanisms should be figured out in the AGM. In our BobIS example, the modeling results are presented in Figure 4. AG1 is the adaptation goal for AS1, which is generated from the original goal user location is identified. AG1 can be achieved by implementing AT1, which is further decomposed with MAPE tasks. AG2 is a newly refined adaptation goal, considering the failure of locating task. Once a failure is detected, AT1 should be invoked to derive the correct location. Different from the above two adaptation goals, AG3 and AG4 are achieved through parametric adaptation. For AG3, the sending time interval is set to a small value for getting newer information. In this situation, buffer needs to be set to an appropriate value for loading information timely. The larger buffer value is, the faster system processes the load data. With these modeling elements, a traditional goal model is converted to the adaptation goal model. The specifications of AGM and general algorithms of MAPE mechanisms are described in our primary work [26].

### Table 1 Adaptation scenarios for different self-* properties

| AS  | Self-* property                      | Cause                 | Effect                                           | Countermeasure                                                   | Example   |
|-----|--------------------------------------|-----------------------|--------------------------------------------------|------------------------------------------------------------------|-----------|
| AS1 | self-configuration, self-optimizing   | Contextual changes    | Decrease system reliability and increase operational costs | Monitor contextual changes and generate the optimal configurations of system structures | AG1       |
| AS2 | Self-healing                         | Task failures         | Lost connection with following related tasks and decrease system reliability | Monitor the implementation of related tasks and redo the task if failure is detected | AG2       |
| AS3 | self-configuration, self-optimizing   | Contextual changes    | Decrease system reliability and increase operational costs | Monitor contextual changes and generate the optimal configurations of related parameters | AG3, AG4 |

![Figure 4 Modeling of adaptation goals and adaptation mechanisms.](image-url)
5. Specifying Reliability and Performance

In adaptation goal model, each goal is refined to operational tasks that can be performed by SASs. However, the reliability and performances are varying according to contexts at runtime, as we discussed in Section 1. This section presents the specifications of reliability and performance in AGM. With these specifications, sequence diagrams of SASs are used to display the performance of adaptation processes.

5.1 Tagged Adaptation Goal Model

Definition 4 (Tagged Adaptation Goal Model) A Tagged Adaptation Goal Model (TAGM) is generated by extending the tasks of AGM with reliability and performances tags. It is denoted as

\[ \text{TAGM} = (G, AG, SG, \{T, TAG\}, AT, \{M, TAG\}, \{A, TAG\}, \{P, TAG\}, \{E, TAG\}, R_M, R_P, R_C). \]

For a certain task \( t \) with parameter \( p \), \( \text{TAG}(t, p) = [c, FP(c), U(c), TC(c, p), EC(c, p)] \), where \( c \) refers to the context value, \( FP(c) \) is the failure probability of task in context \( c \), \( U(c) \) is the utility of task in context \( c \), \( TC(c, p) \) is the time costs with parameter \( p \), \( EC(c, p) \) is the energy costs with parameter \( p \). For structural tasks, \( TC(c, p) \) and \( EC(c, p) \) are simplified as \( TC(c) \) and \( EC(c) \). For a continuous system, \( FP, U, TC \) and \( EC \) are the functions of context \( c \). It is hard to identify the mathematic formulae between SASs and contexts [27]. However, for a discrete system, we can define the values of tags according to contextual classification. In RE stage, a well-conducted contextual modeling and analysis approach is proposed in [28]. Figure 5 present the modeling of tags in TAGM of MobIS, according to the classification of network status.

5.2 Tagged Adaptation Sequence Diagram

Sequence Diagram is an effective model for describing the dynamic interaction between objects and displaying the behaviors of the system-to-be. Hence, for modeling SASs, the processes of adaptation mechanisms can be clearly figured out within the sequence diagram.

Definition 5 (Tagged Adaptation Sequence Diagram) A Tagged Adaptation Sequence Diagram (TASD) describes the interaction between MAPE processes. It is defined as \( \text{TASD} = (O, \{MES, TAG\}, FRA) \), where \( O \) in the object set, \( MES \) is the set of messages, \( TAG \) is the tags related to the messages, \( FRA \) is the set of fragments including \( loop \) and \( alt \). To describing adaptation process, the object is denoted as \( O = (AT, M, A, P, E, C, R) \), where \( C \) is the set of contexts related to the \( AT \), \( R \) is the related resources.

Figure 6 presents the TASD of adaptation task \( AT_1 \). The \( alt \) fragment expresses the messages sent according to different location decisions. Figure 7 presents the TASD of adaptation task \( AT_2 \). The loop fragment indicates requires are sent after each time interval.
6. Modeling Adaptation Behavior

Though sequence diagram can express the dynamic interaction processes of adaptation mechanisms, it is hard to analyze dynamic behavior at runtime for the lack of formalization. To efficiently formalize the behavior of adaptation mechanisms and the behavior of system, this section provides a CSP-based modeling method that transforms the requirements model to the corresponding behavior model.

6.1 Sequential Adaptation Goal Model

Definition 7 (Sequential Adaptation goal Model) A Sequential Adaptation Goal Model (SAGM) is defined as

\[ SAGM = (GP, TP, A, aTP, Traces(TP), Chan, Op, \Delta) \]

- \(GP\) denotes the set of goal processes that are conducted for achieving goal \(G\). \(GP\) is specified by what SASs can do.
- \(TP\) denotes the set of task processes that are performed by task \(T\). \(TP\) is defined by the system behavior.
- \(A\) denotes the set of events (tasks) engaged in \(TP\). Notice that \(\tau \in A\) is the hidden event.
- \(aTP\) denotes the alphabet of \(TP\), which contains the events possibly engaged in \(TP\), e.g. in Figure 4(a), \(aTP_1 = \{GaugeBandwithRate, AnalyzeSatofNFRs, MakeLocationDecision, LocateByGPS, LocateByGSM\}\).
- \(Traces(TP)\) is a finite sequence of events (tasks). A \(TP\) can be specified by the set of traces denoting its possible behavior, e.g. the traces of \(AT_3\) are:

\[ (GaugeBandwidthRate, AnalyzeSatofNFRs, MakeLocationDecision, LocateByGPS) \]

\[ (GaugeBandwidthRate, AnalyzeSatofNFR, MakeLocationDecision, LocateByGSM) \]

- \(Chan\) is the set of channel between processes. The message set on channel of \(TP\) is defined as \(aC(TP) = \{v|chan, v \in aTP\}\), where \(chan\) is the name of channel and \(v\) is the value of the delivered message.
- \(Op\) is the set of operators, including: Prefix (\(\rightarrow\)), Sequential Composition (\(;\)), Parallel Composition (\(\parallel\)) and Choice (\(\square\)).
  - \(\rightarrow\): A process which may participate in task \(t\) then act according to TP is written as \(t \rightarrow TP\).
  - \(;\): The sequential composition of GP1 and GP2, written GP1; GP2, acts as GP1 until GP1 terminates and then proceeds to act as GP2.
  - \(\parallel\): The parallel composition of GP1 and GP2, written GP1||GP2, acts both GP1 and GP2 concurrently.
  - \(\square\): Choice operator allows system a choice of behavior according to what tasks are conducted. The process \((t1 \rightarrow TP1)\square(t2 \rightarrow TP2)\) acts as TP1 if \(t1\) occurs, while acts as TP2 if \(t2\) occurs.
- \(\Delta = R_m \times Op \cup R_o \times Op\) is the mapping relation. \(R_m\) and \(R_o\) have the same meaning within definition 3. This relation is detailed in Table 2.

Summariy, in SAGM, goals are mapped to goal process; tasks are mapped to task process and tasks events (only for leaf-node tasks); AND-decompositions are mapped to Sequential and Parallel operators; OR-decompositions and Means-Ends links are mapped to Choice operators; Dependency relations are mapped to communication between task processes.

To generate the behavior of SASs based on SAGM, we need to further investigate the usages of the operators. By investigating the relations of goals and tasks in the goal model presented in Figure 1, we split these relations into some decomposition and dependency patterns. Each pattern can be transformed to process behavior with the proposed operators. Table 2 elaborates the relation patterns, which can be specified with proposition logic in the goal model. After appending the process operators, relation patterns are specified with CSP descriptions that can be used to derive corresponding behavior models, presented as Labeled Transition Systems (LTSs). Notice that, in behavior patterns, \(\tau\) is the hidden event of related processes.
A LTS is defined as $\mathcal{LTS} = (\mathcal{S}, s_0, \Sigma, \to_s)$, where $\mathcal{S}$ is a finite set of states, $s_0 \in \mathcal{S}$ is the initial state, $\Sigma$ is a finite set of actions, $\to_s \subseteq \mathcal{S} \times \Sigma \times \mathcal{S}$ is a transition relation. The behavior patterns in Table 2 imply that a transition represents a certain system task while a state refers to completion of the task.

**Table 2 CSP descriptions and behavior of relation patterns**

| Relation pattern | Proposition logic | Operator | CSP Specifications | Behavior pattern |
|------------------|-------------------|----------|--------------------|------------------|
| $g_1\wedge g_3 \rightarrow g_1$ | : | $g_2 \land g_3 \rightarrow g_1$ | $\tau_1$ | $\tau_2$ |
| $g_1\wedge g_3 \rightarrow g_1$ | $||$ | $g_2 \wedge g_3 \rightarrow g_1$ | $\tau_1$ | $\tau_2$ |
| $g_1\lor g_3 \rightarrow g_1$ | $\Box$ | $g_2 \lor g_3 \rightarrow g_1$ | $\tau_1$ | $\tau_2$ |
| $t_2 \land t_3 \rightarrow t_1$ | $;$ | $t_2 \land t_3 \rightarrow t_1$ | $\tau_1$ | $\tau_2$ |
| $t_2 \lor t_3 \rightarrow t_1$ | $\Box$ | $t_2 \lor t_3 \rightarrow t_1$ | $\tau_1$ | $\tau_2$ |
| $t_2 \lor t_3 \rightarrow g_1$ | $\Box$ | $t_2 \lor t_3 \rightarrow g_1$ | $\tau_1$ | $\tau_2$ |
| $t_2 \rightarrow t_1$ | $\Box$ | $t_2 \rightarrow t_1$ | $\tau_1$ | $\tau_2$ |

### 6.2 Behavior Model

#### 6.2.1 Behavior of Adaption Mechanisms

According to the above CSP descriptions and behavior patterns, we can derive the behavior patterns of structural adaptation and parametric adaptation, which are provided in Figure 8. It shows that structural adaptation has alternative behavioral structures for executing the adaptation decision, while parametric adaptation has only one structure with alternative parameters.

![Figure 8 Behavior patterns of structural adaptation (a) and parametric adaptation (b)](image)

Based on these two patterns of MAPE processes, we can generate the behavior models of adaptation mechanisms in our Mobiles example (Figure 4). Figure 10(a) presents the behavior for achieving $AG1$. Two alternative behaviors are $locateByGPS$ and $locateByGSM$, which have been modeled as executing tasks in the AGM. Figure 10(b) shows the behavior for achieving $AG2$. Actually, it contains both the adaptation loops of $AG1$ and $AG2$. The right loop acts when the location is accurate, while the left loop acts when the locating task fails. Once the locating failure is detected, $AG1$ is invoked again to relocate users. Figure 10(c) describes the adaptation behavior for $AG3$. Notice that the self-loop in the LTS is produced by the status of related task. For instance, as depicted in Figure 7, the task $sendReq$ is performed each $T$ time unit. If there is no need to reconfigure the value of $T$, $sendReq$ will be continuously performed. Hence, the behavior is denoted as a self-loop transition. Figure 10(d) depicts the behavior for $AG4$. Similarly, the task $loadInfor$ is also a self-loop transition.
6.2.2 Behavior of Self-Adaptive Systems

This subsection presents the transformation from SAGM of a SAS to its behavior model. Firstly, we need to derive the CSP specifications of SASs. Figure 10 presents the specifications of the BobIS example based on the idea of iterative Depth-First Search (DFS). It is produced by a top-down process. The process operators imply the mapping from AGM to Op of SAGM. For example, “MobIS=RecordU;UAccessI” implies the AND-decomposition of the root goal is mapped to the sequential composition operator. “UAccessI=GetOnline[]GetOffline” implies the OR-decomposition of the goal user has access to info is mapped to the choice operator.

```csp
//CSP Specification of SAGM of MobIS
//decomposition of root node
MobIS = RecordU;UAccessI;
//left child node
RecordU = RegistU;PrepareU;
  //left subtree
  RegistU = monitorBandWidth->analyzeNFRs->makeDecision->(R1||R2);
  R1 = registBySMS->PrepareU;
  R2 = registByInternet->PrepareU;
  //right subtree
  PrepareU = IdentifyLocation;IdentifyPrefer;
  IdentifyLocation = locate;CheckLocation;
  locate = monitorBandWidth->analyzeNFRs->makeDecision->L1||L2;
  L1 = locateByGPS->CheckLocation;
  L2 = locateByGSM->CheckLocation;
  CheckLocation = monitorLocationData->analyzeLocation->makeCheckDecision->(IdentifyPrefer||IdentifyLocation);
  IdentifyPrefer = describeInterest->UAccessI;
//right child node
UAccessI = GetOnline[]GetOffline;
  //left subtree
  GetOnline = SendReq;ReceiveI;
  SendReq = ConfigForm;ConfigTime;
  ConfigForm = F1||F2;
  F1 = reqMedia->ConfigTime;
  F2 = reqText->ConfigTime;
  ConfigTime = decideTime->Send;
  Send = sendReq->(Send||ReceiveI);
  ReceiveI = LoadInfo;PersistInfo;
  LoadInfo = decideBuffer->Load;
  Load = loadInfo->(Load||PersistInfo);
  PersistInfo = persistToDB->PrepareU;
//right subtree
GetOffline = SearchDB;FindInfo;
  SearchDB = S1||S2;
  S1 = searchByText->FindInfo;
  S2 = searchByVoice->FindInfo;
  FindInfo = filterInfo->PrepareU;
```

Figure 10 CSP description of MobIS
To check whether these specifications correctly describe the business logic and adaptation logic of the MobIS, we apply the Process Analysis Toolkit (PAT) to the simulation and verification of the above specifications. After simulation, the generated LTS is depicted in Figure 11. It merges four loops of adaptation mechanisms presented Figure 9. With the LTS, we can verify several properties of the system.

Figure 11 Generated LTS from CPS specifications of MobIS

Four kinds of properties of the LTS are verified, as presented in Figure 12. Safety properties capture primary demands of systems behavior. Deadlock-free depicts the system has no deadlock state. Non-terminating depicts the system has no terminated states. Divergence-free refers to system behavior can be recursively controlled. Deterministic property means for any system state, there is no two out-going transitions leading to different states but with the same task. Refinement properties describe the decomposition relations in AGM. Liveness and reachability properties are defined based on Linear Temporal Logic (LTL). Liveness properties represent the properties that should always be satisfied. The liveness assertion in Figure 12 means that once the process IdentifyLocation acts, the event monitorLocationData can always be engaged. Reachability properties capture the states that can be finally reached in a certain behavior path. The assertion in Figure 12 depicts that once online mode is chosen, MobIS can finally persist information to database. While, once offline mode is chosen, user can finally filter the searched information. The correctness of these verified properties implies the reasonability of the transformed LTS and the consistency of it with business logic and adaptation logic.

Figure 12 Properties and verification results

7. Reliability-related Adaptation

Based on the above extended requirements models and behavior models, we elaborate how these models can be used for deriving runtime adaptation decision. We divide the adaptation process into two phases of adaptation. The first phase of adapta-
tion is performed according to reliability requirements, while the second step is conducted according to performance requirements. This section presents the reliability-based adaptation.

7.1 Probabilistic Behavior Model

Discrete Time Markov Chains (DTMCs) are a widely accepted formalism to model reliability of systems. In our work, these components are represented with system tasks that are conducted by related components. Particularly, DTMCs are proved to be useful for predicting reliability [30]. The adoption of DTMCs implies that the modeled system meets the Markov property[31]. DTMCs are discrete stochastic processes with the Markov property, according to which the probability distribution of future states depends only upon the current state. They are defined as a Kripke structure with probabilistic transitions among states. States represent possible configurations of the system. Transitions among states occur at discrete time and have an associated probability. Traditional DTMCs have fixed structures, which is not appropriate for SASs. Hence, we extend DTMCs with a set of variable states. Formally, a Variable DTMC is a tuple \((S_I, S_v, s_0, P, L)\) where

- \(S_I\) is a finite set of invariable states.
- \(S_v\) is a finite set of variable states.
- \(s_0 \in S\) is the initial state.
- \(P: (S_I \cup S_v) \times S_I \rightarrow [0, 1]\) is a set of transition probability matrix, where \(\sum_{s' \in S} P(s, s') = 1, \forall s \in S_v\). For a given structure, the element \(P(s_v, s_i)\) represents the probability that the next state of the process will be \(s_i\) given that the current state is \(s_v\).
- \(L: (S_I \cup S_v) \rightarrow 2^{AP}\) is a labeling function. \(AP\) is a set of atomic propositions. The labeling function associates to each state the set of atomic propositions that are true in that state.

A state \(s \in S\) is said to be an absorbing state if \(P(s, s) = 1\). If a DTMC contains at least one absorbing state, the DTMC itself is said to be an absorbing DTMC. In the simplest model for reliability analysis, the DTMC will have two absorbing states, representing the correct accomplishment of the task and the task’s failure, respectively. The use of absorbing states is commonly extended to modeling different failure conditions. For example, different failure states may be associated with different system tasks.

Recalling the behavior models in the above sections, TASD describes the reliability and performance information of system behaviors, the generated LTS captures the processes of system behavior. Hence, we elaborate how to derive the corresponding DTMC by integrating TASD and LTS. The generating algorithm is as follows.

```
Algorithm Generate Variable DTMC

Input: TASD, SAGM
Output: DTMC

1: for all \(w_i \in S_I\) find next state \(w_{i+1}\)
2: if (\(|w_{i+1}| = 1\)) // no branched transitions
3: if (\(|\Sigma(w_i, w_{i+1})| = 1 \& \& \Sigma(w_i, w_{i+1}) \text{ not for } selfHealing\))
4: add \(s_i\) to \(S_v\), label \(\Sigma(w_i, w_{i+1})\) to \(s_i\), add \(f_0\) to \(S_I\) // \(f_0\) is the failure state related to \(s_i\)
5: end if
6: if (\(\Sigma(w_i, w_{i+1}) > 1 \& \& \Sigma(w_i, w_{i+1}) \text{ not for } selfHealing\))
7: add \(s_i\) to \(S_v\), label \(\Sigma(w_i, w_{i+1})\) to \(s_i\), add \(f_0\) to \(S_I\)
8: end if
9: if (\(\Sigma(w_i, w_{i+1}) \text{ for } selfHealing\))
10: add \(h_0\) to \(S_v\), label \(\Sigma(w_i, w_{i+1})\) to \(h_0\), add \(f_0\) to \(S_I\) // \(f_0\) is the failure state related to \(h_0\)
11: end if
12: if (\(s_0\) is initial state)
13: \(s_0 = s\), find o performing \(L(s_0)\) in TASD, add \(T(s_0, f_0)\) with \(P(s_0, f_0) = TAG(o, FP)\)
14: else if (\(s_0\) is not the last state)
15: find o performing \(L(s)\) in TASD, add \(T(s, f_0)\) with \(P(s, f_0) = TAG(o, FP)\)
16: add \(T(s, f_0)\) with \(P(s, f_0) = 1-TAG(o, FP)\) // \(o_1\) performs \(L(s, i)\)
17: else
18: add self-loop with \(P(s, s) = 1\) // \(s\) is an absorbing state
19: end if
20: if (\(h_0\) is initial state of selfHealing process)
21: substitute the related failure state with \(h_0\) and keep the incoming transition
22: find o performing \(L(h_0)\) in TASD, add \(T(h_0, f_0)\) with \(P(h_0, f_0) = TAG(o, FP)\)
23: else if (\(h_0\) is not the last state of selfHealing process)
24: find o performing \(L(h)\) in TASD, add \(T(h, f_0)\) with \(P(h, f_0) = TAG(o, FP)\)
25: add \(T(h, f_0)\) with \(P(h, f_0) = 1-TAG(o, FP)\) // \(o_1\) performs \(L(h, i)\)
26: end if
27: else
28: for each branches perform Algorithm 1
29: end if
```
In our MobIS example, after applying the generating algorithm, we can derive the variable DTMC presented in Figure 13. The gray-colored states refer to the variable state. The transition probabilities from these states are probability vectors. This DTMC contains five absorbing states, including four failure states and a success state. Though, state 15 has two outgoing transitions, system should choose one path for operation at runtime. Hence, the adaptation decision should be made by configuring each variable state.

![Variable DTMC of the MobIS](image)

**Figure 13 Variable DTMC of the MobIS**

The failure probabilities of transitions are provided in Table 3 and Table 4. Specifically, if the failure probability of $S_i$ is FP, the success probability is 1-FP. For example, FP$_{05} = 0.0005$ means that $P(S_0, S_2) = 0.0005$ while $P(S_0, S_1) = 0.9995$. The failure probabilities of variable states are identified according to context, as discussed in Section 5.1.

**Table 3 Failure probabilities (FP) of invariable states**

| S   | FP   | S   | FP   | S   | FP   | S   | FP   | S   | FP   |
|-----|------|-----|------|-----|------|-----|------|-----|------|
| $S_0$ | 0.0005 | $S_1$ | 0.0001 | $S_2$ | 0.0002 | $S_3$ | 0.0005 | $S_4$ | 0.0005 |
| $S_5$ | 0.0005 | $S_6$ | 0.0001 | $S_7$ | 0.0004 |
| $S_8$ | 0.0001 | $S_9$ | 0.0005 |
| $S_{10}$ | 0.0005 | $S_{11}$ | 0.0005 | $S_{12}$ | 0.001 | $S_{13}$ | 0 |
| $S_{14}$ | 0.0005 | $S_{15}$ | 0.05 |

**Table 4 Failure probabilities (FP) of variable states**

| Context = C1 (Bandwidth <= 100kbps) | S   | S$_1$ | S$_2$ | S$_3$ | S$_4$ | S$_5$ | S$_6$ |
|-------------------------------------|-----|------|------|------|------|------|------|
| Option | SMS | Internet | GPS | GSM | Media | Text | Voice | Text |
| FP     | 0.02 | 0.1   | 0.05 | 0.15 | 0.08 | 0.02 | 0.05 | 0.02 |

| Context = C2 (Bandwidth > 100kbps) | S   | S$_1$ | S$_2$ | S$_3$ | S$_4$ |
|-------------------------------------|-----|------|------|------|------|
| Option | SMS | Internet | GPS | GSM | Media | Text | Voice | Text |
| FP     | 0.15 | 0.05   | 0.1 | 0.02 | 0.05 | 0.01 | 0.05 | 0.02 |

### 7.2 Reliability Requirements

Formal languages to express properties of systems modeled through DTMCs have been studied in the past and several proposals are supported by model checkers to prove that a model satisfies a given property. In this paper, we focus on Probabilistic Computation Tree Logic (PCTL) [32], a logic that can be used to express a number of reliability properties.

PCTL is defined by the following syntax:

$$\Phi ::= \text{true} \mid a \mid \Phi \land \Phi \mid \neg \Phi \mid P_{\leq p}(\Psi)$$

$$\Psi ::= X\Phi \mid \Phi ^{\leq t}\Phi$$

where $p \in [0, 1]$, $\preceq \in \{<, =, >, \geq\}$, $t \in \mathbb{R}_{\geq 0}$ and $a$ represents an atomic proposition. The temporal operator $X$ is called *Next* and $U$ is called *Until*. Formulae generated from $\Phi$ are referred to as *state formulae* and they can be evaluated to either true or false in every single state, while formulae generated from $\Psi$ are named *path formulae* and their truth is to be evaluated for each execution path. Semantic details of state formulae and path formulae are omitted here for simplicity.

From the $\text{Next}$ and $\text{Until}$ operators it is possible to derive others. For example, the *Eventually* operator (often represented by the $\diamond^{st}$ symbol) is defined as: $\diamond^{st}\Phi \equiv \text{true} \land \Phi$. It is customary to abbreviate $U^{\preceq 0}$ and $\diamond^{s\infty}$ as $U$ and $\diamond$. 
PCTL can naturally represent reliability-related properties for a DTMC model of the application. For example, we may easily express constraints that must be satisfied concerning the probability of reaching absorbing failure or success states from a given initial state. These properties belong to the general class of reachability properties. Reachability properties are expressed as \( P_{\text{reach}}(\diamond \Phi) \), which expresses the fact that the probability of reaching any state satisfying \( \Phi \) has to be in the interval defined by constraint \( \mu \ p \). In most cases, \( \Phi \) just corresponds to the atomic proposition that is true only in an absorbing state of the DTMC. In the case of a failure state, the probability bound is expressed as \( \leq x \), where \( x \) represents the upper bound for the failure probability; for a success state it would be instead expressed as \( \geq x \), where \( x \) is the lower bound for success.

To derive the adaptation decisions in our approach, we consider two reliability requirements in the MobIS example, which can be described with PCTL:

- **R1**: \( P_{2.685} \) [true \( U \) "success"] it refers to “Probability of success shall be greater than 0.85”.
- **R2**: \( P_{2.960} \) ["locate" \( U \) "success"] it refers to “Probability of success after locate task shall be greater than 0.9”.

## 8. Performance-related Adaptation

This section provides the second phase of adaptation, which is performed based on performance requirements. Similar to Section 7, we first describe the leveraged behavior model and then the concerned requirements.

### 8.1 Real-time Behavior Model

Continuous Time Markov Chains (CTMCs) are stochastic models that allow one to express the execution time of each step of the process. Each execution time is characterized by means of an exponential distribution. The classical formalization of CTMCs provided in [33]. However, except for the execution time, we consider the performance about utility and energy costs. Meanwhile, alternative structures and behaviors should also be aggregated into the CTMC. Hence, we extend a CTMC with variable states and rewards. A Rewarded Variable CTMC is defined as a tuple \((S_i, S_s, s_0, R, \mathcal{R}, L)\) where:

- \( S_i, S_s \) and \( L \) are defined as for DTMCs.
- \( \mathcal{R} \): \((S_i \cup S_s) \times S_i \rightarrow \mathbb{R}_{\geq 0}\) is the set of rate matrix. For a given structure, the element \( R(s_i, s_j) \) represents the rate at which the process moves from state \( s_i \) to state \( s_j \).
- \( \mathcal{R} \): \((S_i \cup S_s) \times \mathcal{T} \rightarrow \mathbb{R}_{\geq 0}\) is the set of tagged rewards that assign a non-negative real value to transition \( Tr \).

An absorbing state \( s_i \) is a state that \( R(s_i, s_j)=0 \) for all states \( s_j \). For any pair of states \((s_i, s_j)\), \( R(s_i, s_j)>0 \) if there exists a transition from \( s_i \) to \( s_j \). Furthermore, \( 1 - e^{-R(s_i, s_j) T} \) is the probability that the transition from state \( s_i \) to state \( s_j \) is taken within \( T \) time units. Through rates, we model the delay in choosing the transition. If more than one transition exiting state \( s_i \) has an associated positive rate, a race among transitions occurs. The probability \( P(s_i, s_j) \) that control, held by \( s_i \), will be transferred to state \( s_j \) corresponds to the probability that the delay of going from \( s_i \) to \( s_j \) is smaller than the one of going toward any other state; formally, \( P(s_i, s_j) = R(s_i, s_j)/E(s_i) \), where \( E(s_i) = \sum_{s_j \in S} P(s_i, s_j) \). \( E(s_i) \) is called exit rate of state \( s_i \).

Based on the derived DTMC, it is easy to generate the corresponding CTMC with the following steps:

- **Step 1**: Remove all the absorbing states that depict task failures, i.e. only the success state remains.
- **Step 2**: Find related time costs (CT) in tags of TASD and assign 1/TC to related transitions as transition rate.
- **Step 3**: Find related utility (U) and energy costs (EC) in tags of TASD and assign them to related state as rewards.

The generated CTMC is presented in Figure 14. The gray-colored states also represent alternative structures. The transition rates from these states are also vectors. Similarly, the adaptation decision should also be made by configuring each variable state. The concrete values of transition rate, energy costs and utility are presented in Table 5 and Table 6. Notice that some transitions have no assigned values of utility, because these transitions refer to failures of tasks, e.g. \( T_{1,8} \) refers to the failure of \( \text{monitorBandwidth} \), the utility is assigned to 0.

![Figure 14 Variable CTMC of the MobIS](image-url)
### 8.2 Performance Requirements

In this paper, we consider Continuous Stochastic Logic [33] to state properties on CTMC models. For CTMC there are two main type of properties relevant for analysis: steady-state properties where the system is considered “on the long run”, that is, when an equilibrium has been reached; transient properties where the system is considered at specific time points or intervals. CSL is able to express both steady-state and transient properties by means of the $\mathcal{S}$ and $\mathcal{P}$ operators, respectively. Besides, to compute the reward, we extend the state formulae with related syntax. The syntax of extended CSL is recursively defined as follows:

$$
\Phi ::= \text{true} | a | \Phi \land \Phi | \neg \Phi | S_{\text{rep}}(\Phi) | R_{\text{rep}}(\Psi) | R_{\text{rep}}(I^=) | R_{\text{rep}}(C^{SF})
$$

$$
\Psi ::= X^{SF} \Phi | \Phi U^{SF} \Phi
$$

where $p \in [0,1]$, $\mathcal{S} \in \{<, \leq, >, \geq\}$, $t \in \mathbb{R}_{\geq 0}$ and $\mathcal{A}$ represents an atomic proposition.

$S_{\text{rep}}(\Phi)$ describes the probability of being in a state where $\Phi$ holds when $t \rightarrow \infty$, evaluated on all paths originating in $s$, meets the bound $\mathcal{S}_p$. $R_{\text{rep}}(\Psi)$ refers to the probability that the set of paths starting in $s$ and satisfying $\Psi$ meets the bound $\mathcal{S}_p$. $R_{\text{rep}}(\Psi)$ describes the reward that the set of paths starting in $s$ and satisfying $\Psi$ meets the bound $\mathcal{S}_p$. $R_{\text{rep}}(C^{SF})$ depicts the instantaneous reward at time $t$ meets the bound $\mathcal{S}_p$. $R_{\text{rep}}(C^{SF})$ means the cumulative reward until time $t$ meets the bound $\mathcal{S}_p$. With the above syntax, three considered performance requirements of the MobIS example can be specified as follow:

- **R3:** $R(\text{utility})_{t \geq 30}[$true $U$ "success"] it refers to “Utility of success shall be greater than 30”.

- **R4:** $P_{t \geq 0.85}[^\text{locate}U^{\leq 50} \text{"success"}]$ it refers to “Probability of success from locate task to wait less than 50s is greater than 0.85”.

- **R5:** $R(\text{energy})_{t \geq 180}[C^{\leq 50}]$ it refers to “Accumulative energy costs in 50s shall be less than 180”.

### 9. Evaluation Results

In the above two sections, we have derived the behavior models for computing reliability requirements and performance requirements. This section presents the evaluation of achieving adaptation against these non-functional requirements via verification. Specifically, the adaptation decision should be made for satisfying each of the five NFRs. To this end, we conduct the verification process for each possible path in the DTMC and CTMC. After the verification of each NFR, the number of candidate solutions will reduce accordingly.

For implementing verification, we consider taking the advantage of probabilistic model checking [34], which is an automatic technique for verifying quantitative properties for probabilistic systems. To this end, our approach adopts a well-established stochastic model checker PRISM [35, 36] to support the model checking process.
We performed the verification with PRISM 4.3 on a machine with an AMD FX8350 (3.8GHz) CPU with 4GB of RAM. The quantitative settings of DTMC and CTMC have been provided in Table 3 to Table 6. The values of time interval ($T$) and buffer ($B$) are chosen from 1 to 20 with step size of 2. Hence, the MobIS totally includes 808 alternative adaptation decisions for each kind of contexts. The results presented below depict the reduction of decision space after verifying each NFR in context C2 ((Bandwidth > 100kbps)).

Figure 15 depicts six adaptation candidates are chosen by verifying reliability requirement $R1$. The optional structures can be found in the legend. The probabilities in legend order are 0.8670, 0.8899, 0.9268, 0.9000, 0.9238 and 0.9529. Figure 16 presents four of the six candidates are selected as adaptation decision by verifying reliability requirement $R2$. The probabilities in legend order are 0.9196, 0.9583, 0.9296 and 0.9687.

By verifying $R3$, only the structure {Internet, GSM, Online (Media)} is chosen as the adaptation structure, as presented in Figure 17. It is interpreted as when bandwidth is greater than 100kbps, the user shall register through Internet, be located with GSM and receive pushed media information from business server.

By verifying $R4$, candidates of parameters are determined. In Figure 18, 21 pairs of $(T, B)$ are selected: (5,19), (7,17), (7,19), (9,15), (9,17), (9,19), (11,15), (11,17), (11,19), (13,15), (13,17), (13,19), (15,15), (15,17), (15,19), (17,15), (17,17), (17,19), (19,15), (19,17) and (19,19).

After verifying $R5$, as depicted in Figure 19, 10 pairs of $(T, B)$ satisfy the energy requirement, including (5,19), (7,17), (9,15), (9,17), (11,15), (11,17), (13,19), (15,19), (17,19) and (19,19). Among them, we choose the parameters that cost least energy. Then the time interval is assigned to 9 and buffer is set to 15.

Thus, after verifying each alternative path of the DTMC and CTMC against the NFRs, the optimal adaptation decision is derived as {RegistOption=byInternet, LocateOption=byGSM, InfoOption=Online, InfoFormOption=Media, TimeInterval=9, Buffer=15}.
10. Related Work

Modeling adaptation mechanism. RE for self-adaptive systems demands modeling both business logic and adaptation logic. Qureshi et al. [6] focuses on the question on how to represent requirements for SAS in a way which can be read by non-engineering stakeholders. They propose a modeling language, called Adaptive RML, for the representation of early requirements for SASs. The language has graphical primitives in line with classical goal modeling languages. Cheng et al. [7] introduces a goal-based modeling approach to develop the requirements for a DAS, while explicitly factoring uncertainty into the process and resulting requirements. They introduce a variation of threat modeling to identify sources of uncertainty and demonstrate how the RELAX [10] specification language can be used to specify more flexible requirements within a goal model to handle the uncertainty. As the following work, Baresi et al. [37] propose an approach to represent requirements for service compositions by extends traditional goal models with adaptive goals to support continuous adaptation. They also provide a methodology to trace goals onto the underlying composition, assess goals satisfaction at runtime, and activate adaptation consequently. In comparison with their work, our approach describes each adaptation mechanism as a MAPE loop the modeling process can be manipulated conveniently. The syntax of the modeling method is brief but efficient.

Achieving adaptation. To achieve adaptation, researchers proposed both computing-based methods and reasoning-based method. FUSION was proposed by Elkhodary et al. [11]. The approach uses online learning to mitigate the uncertainty associated with changes in context and tune system behaviors to unanticipated changes. Esfahani et al. [12] proposed POISED for improving the quality attributes and achieve a global optimal configuration of a system by assessing both the positive and negative consequences of context uncertainty. Wang et al. [13] focused on monitoring and analysis aspect. They proposed a framework for diagnosing failure of software requirements by transforming the diagnostic problem into a propositional satisfiability problem, which is solved by SAT solvers. In [14], Wang and Mylopoulos proposed an autonomic architecture consisting of monitoring, diagnosing, reconfiguration and execution component. Comparing with their work, our approach provides a different way to deriving adaptation decision, i.e. by verification. This process is completed though Bayesian reasoning with structured behavior models, i.e. DTMCs and CTMCs. In our approach, verification is used as a decision-making process. Besides, we take into consideration several kinds of NFRs.

Requirements verification. Goldsby et al. [15] provide AMOEBA-RT, a run-time monitoring and verification technique that provides assurance that dynamically adaptive software satisfies its requirements. Filieri and Tamburrelli [16] provide a model checking based approach for verification, in which reliability models are given in terms of Discrete Time Markov Chains which are verified against a set of requirements expressed as logical formulae. Epifani et al. [17] lay the foundations for an iterative model-driven development, which aims at verifying that an implementation satisfies non-functional requirements. In their approach, if the resulting running system behaves differently from the assumptions made at design time, the feedback to the model shows why it does not satisfy the requirements and lead to a further development iteration. Ghezzi et al. [18] put forward the proposal of quantitative verification at runtime for self-adaptive service-based software. Ghezzi and Sharifloo [19] propose a model-based approach that enables software engineers to assess their design solutions for software product lines in the early stages of development. Ghezzi and Tamburrelli [38] present a method and supporting tools to rea-
son about requirements of open-world systems built as integrated services that rely on existing, externally provided services. Besides, Zhao et al. [39] propose the mode-supported Linear Temporal Logic that is an effective way to describe global specifications of adaptive software. The global specifications are defined for adaptive software as requirements from the perspective of global adapting process. The model checking problem is also resolved using Linear Temporal Logic and Labelled Transition System Analyzer. In our approach, we also focus on verification of NFRs. However, we concern more system properties, i.e. utility, time costs and energy costs. Meanwhile, we apply verification as the decision-making method during self-adaptation.

11. Conclusions and Future Work

In this paper, we proposed a model-driven approach to assuring non-functional requirements through runtime adaptation, which is implemented via a series of verification processes. We use the adaptation goal model to describe both business logic and adaptation logic. By tagging reliability and performance to system tasks, we define the related tagged adaptation goal model. The behaviors of adaptation tasks are specified in the tagged adaptation sequential diagrams. To formalize system behaviors, we propose a method for transforming the requirements model to behavior model that is expressed as a LTS. The sequential diagram and LTS are merged into a variable DTMC and a rewarded variable CTMC. Related NFRs are specified as PCTL and CSL properties. The verification is implemented within PRISM, a tool for probabilistic model checking. The approach has been successfully applied to a mobile information system for achieving adaptation decision.

The approach can serve as a general guidance for achieving adaptation of SASs in several aspects:

1. The adaptation goal model provides a graphical representation about adaptation mechanism, which can be easily manipulated by adding MAPE tasks to original goal model. Though the representation is concise, it is powerful for generating the behavior models related to these mechanisms.

2. The tagged adaptation goal model is generated by reliability and performances attributes. Notice that the key idea implied here is these attributes are varying according to the operating environment and system configurations. This occurs in many software-intensive systems and it enables the alternative adaptation decisions.

3. The process of transforming a goal model to its behavior model can be applied to SASs of any other domains. It provides pattern-based CSP specifications, which support simulation and verification within analysis toolkits. Meanwhile, the verification results reveal the correctness of behavior models.

4. Though requirements verification is discussed a lot by the RE community. Few works focus on the performance of utility, time costs and energy costs. Our approach proposes a solution to these research gaps. Another key point should be noticed is that we derive adaptation decision by selecting from the verification results. In other words, the verification processes serve as the decision-making unit for an optimization problem. That is also the reason why we design the variable states in DTMCs and CTMCs. Actually, different paths of DTMCs and CTMCs characterized by the variable states are treated as alternative decision-paths.

5. The whole approach starts from RE stage and is highly based on models, including static models and dynamic (behavior) models. It may help researchers and practitioners who focus on requirements engineering and model-driven development.

In our proposal, by transformation and integration, models in RE stage can help runtime analysis of SASs, i.e. verifying NFRs and achieving adaptation.

Future work includes two directions. First, we concern the other type of uncertainties, i.e. nondeterminism. It characterizes a system that has two or more behaviors at a certain time point. For example, when a request comes into the server queue, the server will either serve the request or wait for another request. This situation is different from the probabilistic attributes of system behaviors. The nondeterminism can be modeled in Markov Decision Processes (MDPs) appropriately. We are interested in how to achieve adaptation decision for the nondeterministic adaptive systems, especially for assuring NFRs. We also concern the verification of NFRs with other uncertain attributes, especially the flexibility provided by introducing fuzziness into NFRs. To this end, we need to build effective adaptation loops, e.g. MAPE loops and reasonably describe contextual changes, fuzzy requirements and adaptation logic. We consider UPPAAL-SMC as a proper solution to this problem.

Reference

[1] Cheng, B. H. C., Lemos, R., Giese, H., Inverardi,P., Magee,J. et al. 2009. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In Software Engineering for Self-Adaptive Systems I. Springer. Vol. 5525, 1-26.

[2] Salehie, M. and Tahvildari, L. 2009. Self-adaptive software: Landscape and research challenges. ACM Trans. Auton. Adapt. Syst. 4, 2, 1-42.

[3] Andrea, A. F., Savigni, A. 2001. A Framework for Requirements Engineering for Context-Aware Services, In Proc. of SSAW’01.

[4] Yang, Z., Li, Z., Jin, Z., and Chen, Y. 2014. A Systematic Literature Review of Requirements Modeling and Analysis for Self-adaptive Systems, In Proc. of REFSQ’14.

[5] van Lamsweerde A (2009) Requirements engineering: from System goals to UML models to software specifications. Wiley, New York
[6] N.A. Qureshi, I.J. Jurtera, A. Perini, Towards a requirements modeling language for self-adaptive systems, in: Proceedings of the 18th International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Springer, 2012, pp. 263-279.

[7] B.H.C. Cheng, P. Sawyer, N. Bencomo, J. Whittle, A goal-based modeling approach to develop requirements of an adaptive system with environmental uncertainty, in: Proceedings of the 12th International Conference on Model Driven Engineering Languages and Systems (MODELS), Springer, 2009, pp. 468-483.

[8] 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 RE ’10. IEEE, 125-134.

[9] Baresi, L., Pasquale, L. and Spoletini, P. 2010. Fuzzy Goals for Requirements-Driven Adaptation. In Proc. of RE ’10. IEEE, 125-134.

[10] Whittle, J., Sawyer, P., Bencomo, N., Cheng, B. H. C. and Bruel, J. M. 2009. RELAX: Incorporating Uncertainty into the Specification of Self-Adaptive Systems. In Proc. of RE ’09. IEEE, 79-88.

[11] Elkhodary, A., Esfahani, N. and Malek, S. 2010. FUSION: a framework for engineering self-tuning self-adaptive software systems. In Proc. of FSE ’10. ACM, 7-16.

[12] Esfahani, N., Kouroshfar, E. and Malek, S. 2011. Taming uncertainty in self-adaptive software. In Proc. of ESEC/FSE ’11. ACM, 234-244.

[13] Wang, Y., McIlraith, S. A., Yu, Y. and J. Mylopoulos. 2009. Monitoring and diagnosing software requirements. Automated Software Engg. 16, 1, 3-35.

[14] Wang, Y. and Mylopoulos, J. 2009. Self-Repair through Reconfiguration: A Requirements Engineering Approach. In Proc. of ASE’09. IEEE, 257-268.

[15] H.J. Goldsby, B.H.C. Cheng, J. Zhang, AMOeba-RT: Run-time verification of adaptive software, in: H. Giese (Ed.), Models in Software Engineering. Springer, 2008, pp. 212-224.

[16] A. Filieri, G. Tamburrelli, Probabilistic verification at runtime for self-adaptive systems, in: J. Cámara, R. Lemos, 21C. Ghezzi, A. Lopes (Eds.), Assurances for Self-Adaptive Systems, Springer, 2013, pp. 30-59

[17] I. Epifani, C. Ghezzi, R. Mirandola, G. Tamburrelli, Model evolution by run-time parameter adaptation, in: Proceedings of the 31st International Conference on Software Engineering (ICSE), IEEE, 2009, pp. 111-121.

[18] Calinescu R, Ghezzi C, Kwiatkowska M, et al. Self-adaptive software needs quantitative verification at runtime. Communications of the ACM, 2012, 55(9):69-77.

[19] Ghezzi C, Sharifloo A M. Model-based verification of quantitative non-functional properties for software product lines. Information & Software Technology, 2013, 55(3):508-524.

[20] Brun, Y., Serugendo, G. M., Gacek, C., Giese, H., Kienle, H., Litou, M., Müller, H., Pezze, M., Shaw, M.: Engineering Self-Adaptive Systems through Feedback Loops. In: Cheng, B. H. C., de Lemos, R., Giese, H., Inverardi, P., Magee, J. (eds.) Software Engineering for Self-Adaptive Systems, LNCS, vol. 5525, pp. 48-70. Springer Heidelberg (2009)

[21] Didac Gil de la Iglesia, Danny Weyns Guaranteeing Robustness in a Mobile Learning Application using Formally Verified MAPE Loops, In SEAMS’13, 83-92.

[22] Ali R, Dalpiaz F, Giorgini P. Location-Based Software Modeling and Analysis: Tropos-Based Approach, International Conference on Conceptual Modeling. Springer-Verlag, 2008:169-182.

[23] E. Yu, Towards modelling and reasoning support for early-phase requirements engineering, in: Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, 1997, pp. 226-235.

[24] E. Yuan, N. Esfahani, S. Malek, A Systematic Survey of Self-Protecting Software Systems, ACM Trans. Auton. Adapt. Syst., 8 (2014) 1-41.

[25] McKinley, P. K., Sadjadi, S.M., Kasten, E. P., Cheng, B.H.C. 2004. Composing Adaptive Software. Computer. 56-64.

[26] Z. Yang, Z. Jin, Modeling and specifying parametric adaptation mechanism for self-adaptive systems, in: Proceedings of the First Asia Pacific Requirements Engineering Symposium, Springer, 2014, pp. 105-119

[27] Souza, V. E. S., Lapouchnian, A. and Mylopoulos, J. 2011. System identification for adaptive software systems: a requirements engineering perspective. In Proc. of ER ’11. Springer. Vol. 6998, 346-361.

[28] R. Ali, F. Dalpiaz, P. Giorgini, A goal-based framework for contextual requirements modeling and analysis, Requir. Eng., 15 (2010) 439-458.

[29] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall International, 1985.

[30] Immonen A, Niemela E (2008) Survey of reliability and availability prediction methods from the viewpoint of software-architecture. Softw Syst Model 7(1):49–65

[31] Cheung RC (1980) A user-oriented software reliability model. IEEE Trans Softw Eng 6(2):118–125

[32] Baier C, Katoen J-P (2008) Principles of Model Checking. MIT Press, Cambridge

[33] Baier C, Haverkort B, Hermanns H, Katoen J-P (2003) Model-checking algorithms for continuous-time markov chains. IEEE Trans Softw Eng 29:524–541
[34] V Forejt, M Kwiatkowska, G Norman, D Parker, Automated verification techniques for probabilistic systems, Formal Methods for Eternal Networked Software Systems, LNCS, 6659:53-113, Springer, 2010.

[35] Hinton A, Kwiatkowska M, Norman G, Parker D (2006) Prism: A tool for automatic verification of probabilistic systems. In: Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, vol 3920, pp. 441–444.

[36] Kwiatkowska M, Norman G, Parker D (2004) Prism 2.0: a tool for probabilistic model checking. In: Quantitative Evaluation of Systems, 2004, QEST 2004. Proceedings. First International Conference, pp 322–323.

[37] L. Baresi, L. Pasquale, Adaptive goals for self-adaptive service compositions, in: Proceedings IEEE International Conference on Web Services (ICWS), IEEE, 2010, pp. 353-360.

[38] Zhao, Z. Li, H. Shen, D. Ma, Development of global specification for dynamically adaptive software, Computing 95 (2013) 785-816.

[39] Ghezzi C, Tamburrelli G. Reasoning on Non-Functional Requirements for Integrated Services, Conference, In Proceedings of 17th IEEE International Conference on Requirements Engineering. IEEE, 2009:69-78.