A Preliminary Study on Requirements Modeling Methods for Self-Adaptive Software Systems

Tianqi Zhao, Haiyan Zhao and Wei Zhang

ABSTRACT
Internetware denotes a kind of complex distributed software system, which executes in an open, uncertain and dynamic environment, and adapts itself to changes in the environment. An important problem related to the development of Internetware applications is how to define their requirements. Traditional requirements modeling methods work well with software applications deployed in predictable environment, but cannot deal with Internetware applications, which have to identify and adapt themselves to the unpredictable situations of their environment. The self-adaptation characteristic of Internetware applications introduces challenges to the effective modeling of the requirements of Internetware applications. In this paper, we carry out a preliminary study on requirements modeling methods for self-adaptive software systems. In particular, we focus on how existing requirements modeling methods address the challenges caused by self-adaptation and what are the advantages and disadvantages of their solutions. By doing this study, we aim to identify the essential capabilities or properties that a requirements modeling method should possess so as to support the requirements modeling of self-adaptive software systems like Internetware.

Categories and Subject Descriptors
D.2.1 [Software Engineering]: Requirements/Specifications - Languages, Methodologies

General Terms
Languages, Theory, Verification.

Keywords
Self-adaptive system; requirement modeling method; requirement representation; requirement verification; requirement evolution

1. INTRODUCTION
An Internetware application is a software system that consists of a set of self-contained, autonomous entities situated in the distributed nodes of Internet and a set of accompanied coordinators connecting these entities, either statically or dynamically, in various kinds of interaction styles [1], and usually executes in an open, dynamic and uncertain environment. In order to handle uncertainty in the environment, an Internetware application should have the ability to tolerate and adapt to changes of the environment, both autonomously and dynamically.

As described in [14], self-adaptation is widely recognized as a feasible approach to tackle some of the hard challenges in engineering and managing complex distributed software systems. Self-adaptation means the ability of a system to automatically modify its behavior in order to satisfy certain objectives. Internetware can be regarded as a kind of complex distributed system with self-adaptive property.

Although self-adaptive systems have obvious advantages, the requirements for such systems are more difficult to capture reason over and manage. Traditional requirements modeling methods work well with software operating in predictable environment, but cannot deal with self-adaptive systems, which have to identify and adapt themselves to the unpredictable situations of their environment. The requirements models of self-adaptive systems need to tackle new challenges in order to handle uncertainty and support adaptation behaviors.

As described in Table 1, self-adaptive systems differ from traditional non-adaptive systems in the following nine aspects. First of all, the problem domain of a non-adaptive system is usually well-understood, while the domain knowledge of a self-adaptive system usually cannot be fully understood at development phase. At development time, we cannot anticipate the entire set of possible environmental conditions. As a result, we cannot anticipate all possible adaptation behaviors of self-adaptive systems. Third, the context in which a non-adaptive system operates is static or evolves slowly at run-time. However, a self-adaptive system often executes in an unpredictable changing environment. Moreover fourthly, the degree of uncertainty is relatively low in non-adaptive systems, while self-adaptive systems are subject to uncertainty issues which might arise from both exterior and interior. Fifth, the requirements of self-adaptive system may need to change and evolve at run time in response to the changing environment, differing from traditional non-adaptive systems where requirements seldom evolve. This is mainly caused by a lack of knowledge at development phase and the inherent uncertainty associated with the unknown environment. Sixth, the verification of requirements for a non-adaptive system can be implemented using static verification techniques while a self-adaptive system needs dynamic verification techniques and run-time monitoring [5]. As such, a self-adaptive system must...
continuously monitor changes in context [6]. Then seventh aspect, unlike non-adaptive systems which have scarcely adaptation properties, self-adaptive systems have the capability to modify their behavior and structure when it is observed divergence between requirements and running status occurs. Such adaptation behavior mainly includes self-healing, self-reconfiguration and self-optimization [20]. The eighth different aspect, trade-offs between requirements in a traditional non-adaptive system are performed at development time. But as to self-adaptive systems, a new trade-off between several potentially conflicting requirements will be needed when context changes. So behaviors of making multi-objective decisions between requirements shall be shifted to run-time. Finally, due to the high degree of uncertainty in self-adaptive systems, certain development and change activities are shifted from development-time to run-time [2]. Accordingly the traditional boundary between development-time and run-time blurs.

### Table 1: Comparison between self-adaptive system and non-adaptive system

<table>
<thead>
<tr>
<th>Property</th>
<th>Non-adaptive system</th>
<th>Self-adaptive system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development-time</td>
<td>Information about</td>
<td>Adequate</td>
</tr>
<tr>
<td></td>
<td>environment</td>
<td>Lack</td>
</tr>
<tr>
<td></td>
<td>Information about</td>
<td>Adequate</td>
</tr>
<tr>
<td></td>
<td>system behavior</td>
<td>Lack</td>
</tr>
<tr>
<td>Run-time</td>
<td>Context</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Context dynamic</td>
<td>Changes happen slowly and predictable</td>
</tr>
<tr>
<td></td>
<td>Degree of uncertainty</td>
<td>Low</td>
</tr>
<tr>
<td></td>
<td>Requirements evolution</td>
<td>Seldom</td>
</tr>
<tr>
<td></td>
<td>Requirements Verific</td>
<td>Static verification technique</td>
</tr>
<tr>
<td></td>
<td>Capability of adaptation</td>
<td>Scarcely</td>
</tr>
<tr>
<td></td>
<td>Requirements trade-off phase</td>
<td>Requirement and design phase</td>
</tr>
<tr>
<td>Development-time and run-time</td>
<td>Clarity</td>
<td>Blur</td>
</tr>
<tr>
<td>boundary</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As described above, self-adaptive systems like Internetware bring new challenges to traditional requirements engineering, where there is no explicit support for uncertainty or adaptation. To perform requirements modeling for a self-adaptive system, the specification of requirements should cope with the incomplete information and possible evolution. The requirements must allow themselves to be dynamically observed to ensure the satisfaction of certain objectives. The model of a self-adaptive system should also support run-time adaptation behaviors.

A lot of research has been conducted for requirements modeling of self-adaptive systems [7, 8], and some promising approaches have been proposed. In this paper, we carry on a preliminary study on requirements modeling methods for self-adaptive systems. In particular, we focus on three basic aspects which should be adequately addressed in the self-adaptive requirements models. The three aspects are adequate information for requirements representation, supporting information for requirements verification and guidance information for requirements evolution. Among the three aspects, information for requirements representation should be able to support flexibility and variability of run-time requirements, information for requirements verification should support dynamic verification techniques and run-time monitor, and guidance information for requirements evolution should support adaptation process and selection of adaptation behavior. By comparing the ways how the existing modeling methods deal with the three aspects, we aim to identify the essential capabilities or properties that a requirements modeling method should possess to deal with self-adaptive systems like Internetware.

The rest of the paper is structured as follows. In section 2 we introduce related backgrounds and elaborate key points in requirements modeling approach for self-adaptive systems, concluding by dimensions in the three aspects above. We also give a brief introduction of seven requirements modeling methods which we will inspect in this paper. In the third section, we detailed introduce how those seven modeling methods deal with features and concepts of the three aspects. In the last section we give a table to show conclusion and comparison.

### 2. BACKGROUND

This paper concentrates on self-adaptive systems using feedback loops controlling their dynamic behavior. The feedback loop highlights four key activities: collect, analyze, decide and act. The architecture of self-adaptive systems can be described using a three layers model. The three layers of this architecture, namely goal management layer change management layer, and component management layer, could well supports self-adaptive mechanism.

Through the analysis of feedback loop and self-adaptive architecture, we motivate the need for finding suitable requirements modeling approach for self-adaptive systems. Finally we propose essential features and concepts of the three aspects, which are presented in section 1. In the last part of this section, we give a brief introduction of seven requirements modeling method which we focus on in this paper.
2.1 Feed-back Loop

Feedback loops have been recognized as important factors in software process management and improvement or software evolution. It provides the generic mechanism for self-adaptation. Figure 1 shows the generic feedback loop, which consists of four key activities: collect, analyze, decide and act. The generic model of a control loop provides an overview of the four activities around the control loop. First, the feedback cycle collects relevant data which reflect current status of the system from sensors or probes. Next, the data will be cleaned and filtered, finally be analyzed to infer trends and identify symptoms. Thirdly, the planner will predict the future to decide on how to act on the executing system. Finally, the system will act via available actuators and effectors.

The feedback behavior is a crucial feature of self-adaptive systems. In IBM's blueprint for building autonomic systems, MAPE-K feedback loops which can be used to engineering self-adaptive systems have been proposed. MAPE-K stands for monitor-analyze-plan-execute over a knowledge base, of which the phases map readily to generic feedback loops.

![Figure 1. Autonomic control loop [11]](image)

MAPE-K feedback loops highlight main activities at run phase of self-adaptive systems [12]. To enable self-adaptation, a software system would first collect the necessary information that can reflect the current status of the system and the operating environment. Then the collected information will be further analyzed, in order to see to what extent the requirements are satisfied. Based on the analyzing results, the software system will decide whether to adjust its behaviors and what kinds of adjustments are needed. Finally, the adjustments will be implemented according to the plan which is made by the self-adaptive system itself.

2.2 Requirements Engineering for Self-adaptive Systems

Requirements engineering is concerned about what a system should do and within what constraints it must do it. At the heart of requirements engineering is to formulate the system-to-be's requirements model, comprising goals, domain assumptions and requirements. For self-adaptive systems, requirements engineering should be able to discover, reason about and manage requirements at run-time due to significant uncertainty which exists at design-time. So the requirements of self-adaptive systems should become run-time entities to support run-time requirement engineering.

The MAPE-K loops have highlighted key activities of self-adaptive systems at run time, which are monitor, analyze, plan and execute. The feedback cycle starts with monitor. Run-time monitor is necessary to discover deviations between requirements model and system’s behaviors. With the data collected, systems will then structure, reason about and analyze the raw data to verify and validate system's requirements. Run-time monitoring and dynamic verification techniques are necessary to be used to verify to what extent self-adaptive systems' requirements are satisfied. Because the system cannot monitor everything, it is necessary to specify exactly what should the self-adaptive system monitor. It is the requirements that induce the greatest need for monitoring and the greatest challenges to determine what to monitor for. With raw data collected from sensors and components, dynamic verification techniques will then be performed to verify adherence to requirements.

Requirements for self-adaptive systems, which become run-time artifacts, will evolve when more information are acquired by monitoring. Requirements evolution is mainly triggered by changes in the operational context, although it can also be triggered by stakeholders. Some of the requirements need to be maintained at run-time, while some requirements are allowed to change and evolve. As a result, it is important to decide which requirements to be maintained and which are evolvable. During the evolution, trade-offs between conflicted requirements will be performed. Self-adaptive systems should also react accordingly and employ suitable adaptation behavior to guarantee its synchronization with the updated model.

Requirements models for self-adaptive systems can use information provided as feedback to update itself and keep synchronization with the architecture. So the representation of requirements should be causally connected to the running systems. The representation of requirements should cope with changing environment and possible requirements evolution. It is necessary for requirements representation to support dynamic changes of requirements and relationships between them.

There are therefore following information in three areas to be addressed in the requirements models for self-adaptive systems.

- **Representation of requirements** --- how to specify dynamic functional or non-functional requirements, and how to specify dynamic relationship between requirements in order to reason over at run-time [9].
- **Verification of requirements** --- what to monitor for, how to verify to what extent the requirements are satisfied. The aspects to be monitored might be environment property, derivation events, system components, system requirements and etc.
- **Evolution of requirements** --- when and how to evolve requirements, and how to make adaptation decisions [10] when more than one adaptation alternatives are applicable.

2.3 Related Requirements Modeling Approach

Requirements engineering for self-adaptive systems is a wide open research area and a number of approaches have been considered. This preliminary study mainly takes seven approaches into consideration.

A few research extended goal-oriented modeling approach to support self-adaptive systems. Cheng and Pete [3] have proposed a method using goal-oriented models as foundation to specify adaptation behaviors. In a goal-oriented model for self-adaptive systems, high-level goals are to be maintained at run-time while different goal paths to realize them will be selected dynamically according to the changing context. Sawyer and Bencomo have also come up with LOREM (level of requirements engineering for modeling) process to model the requirements of self-adaptive systems evolution.
systems based on the idea of requirements levels. Here the self-adaptive systems are regarded as a collection of static systems.

A KPI based model [28], which use key performance indicators as its foundation, have been proposed by Rosa to support adaptation behavior. Goals and deviation events are specified in formal language with related KPI. The set of adaptation behaviors to employ when certain deviation events occur are also predefined. Probabilistic models [29] can be used to verify non-functional requirements of the self-adaptive system. By using a Markov model, the probability of reaching a possible state can be specified. Additionally a Markov model with reward can address the properties (energy consumption, CPU) of a system. Those models can then fed into parametric model checker to evaluate requirements satisfaction.

Requirements of self-adaptive systems might have different levels of flexibility, and a language based on fuzzy semantic have been proposed to relax requirements. This requirements language, called RELAX [26, 27], can be used together with other methods to deal with uncertainty in the requirements.

To benefit the monitoring of requirements, Robinson [32] proposed a monitor framework. This framework mainly establishes the relationships among requirements, assumptions made about the current state of the environment, and the set of remedial evolutions available.

AOM (aspect oriented modeling) [38] aims at derive a wide range of modes by weaving aspects into an explicit model. It can also be used to model self-adaptive systems, where the aspect will be dynamically weaved into the explicit model.

3. REQUIREMENTS MODELING METHODS FOR SELF-ADAPTIVE SYSTEMS

We have identified essential concepts and features of three aspects to address in requirements models for self-adaptive systems. The three aspects, which have been presented above, are representation of requirements, verification of requirements and evolution of requirements.

For each aspect, we inspect the seven models which we briefly introduced in the second section, and conclude how these models address essential information. The seven modeling methods we take into consideration in this preliminary study are goal-oriented methods, LOREM, KPI-based method, probabilistic method, RELAX, monitor framework and AOM.

One or more representative papers are selected for each method. We think the selected papers can well address the challenges mentioned above in requirements engineering of self-adaptive systems. Those papers are chosen because, when compared with other papers using the similar method, they give a more detailed description of the three aspects which we want to inspect.

3.1 Representation of Requirements

The specification of requirements is actually a challenging task due to inherent uncertainty and it is argued that requirements of self-adaptive system should become run-time entities. To handle unknown uncertainties, the representation of requirements should cope with the incomplete information of both the environment and the corresponding adaptation behaviors.

First, for a self-adaptive system, different contexts lead to different trade-off between potentially conflictive requirements, and changing context can even result in new requirements. Therefore the specification of requirements should support the variability of requirements, including functional requirements and non-functional requirements. Next, among the requirements of self-adaptive system, some requirements need to be maintained while other requirements are allowed to evolve and change at run time. Thus requirements for self-adaptive system may involve degree of flexibility. Finally, to support reason over of requirements, the relationship between requirements which might dynamically change should also be specified. Functional requirements might be decomposed to sub requirements until the leaf requirements could be assigned to components, and also functional requirements might be conflicted from each other. On the other hand, functional requirements can help or hurt a non-functional requirement, and this help/hurt relationship might dynamically change during system execution.

Goal based modeling notations, such as i* [15] and KAOS [16] have been extended to be applied in requirement modeling of self-adaptive systems. In [18], it was noted that a DAS is a collection of target systems, each of which handles a combination of environmental conditions. The LOREM process (levels of RE for modeling) [19] uses this idea to represent the requirements of individual target systems. To handle degree of flexibility, requirements can be represented by textual language based on fuzzy semantic. For requirements specified in the form of formula, parameters which cannot be determined at development phase might be included. Those parameters can be tuned at run-time according to changing environment. Additionally, methods of requirements specification based on probabilistic and probability can also be used to tame uncertainty.

Goal-oriented Models

A lot of current research represent requirements based on goal-oriented modeling methods such as i* and KAOS. These studies [20, 22-24] conduct extensions on existing goal models to address more essential information in order to support self-adaptive property. Goal oriented models [21] specify functional requirements in terms of hard-goals and specify non-functional requirements in terms of soft-goals. For example [20], a stakeholder goal may be specified as “collect timetables”. High level goals can be decomposed into sub goals, e.g. “collect timetables” might be decomposed into two sub goals “collect time table from agents” and “collect timetable from users”. The decomposition of high level goals includes AND-decomposition, OR-decomposition and XOR-decomposition. If one goal is AND-decomposed into sub goals, it will be satisfied only if all of its sub goals are satisfied. If the high level goal is OR-decomposed into sub goals, it will be regarded as being satisfied as long as one or more of its sub goals are satisfied. As a result, there may be more than one path to realize high-level objectives in a goal-oriented model. Non-functional requirements will be specified as soft goals in a goal-oriented method, such as “Minimal disturbance”. In goal models, goals can be related to soft goals through help(+), hurt(-) relationships. For example, “collect timetable from agents” helps the soft goal “minimal disturbance” while “collect timetable from users” hurts this soft goal.

As to a self-adaptive system, the high level goals, which represent the key services supplied by the system, are need to be maintained while non-critical goals could well be relaxed. Those goals which allow variability and flexibility can be recognized as optional goals. By identifying possible behaviors as optional sub goals and performing trade-off dynamically at run-time, a goal model can
adapt itself to changing environment. Actually, subgoal refinement is a less cost approach to handle uncertainty. Moreover, relationships between hard and soft goals of self-adaptive systems might also be dynamic and autonomous changing according to the changing context, thus this kind of relationship might be specified as a function if it is necessary. If a goal is likely to be satisfied only if certain conditions are met, it can be addressed with a function with the format “if (condition)” . Additionally, n-line goal refinement can be implemented, which means dynamic adaptation systems might perform the requirement process at run-time.

LOREM
Runtime requirements of self-adaptive systems can also be represented using the LOREM process (levels of RE for modeling) [19], where self-adaptive systems are regarded as a collection of target static systems. This method is based on the theory of requirement levels. Berry clarified requirements as four levels in [25]. Level 1 requirements are the traditional RE work done for a system. It identifies all possible static systems that can be executed by self-adaptive systems. Actually a steady-status system is an instance of the system which handles a combination of environmental conditions. Each steady-status system can be represented by a behavior model such as a goal-oriented model.

Level 2 requirements are the RE work done by adaptive systems at runtime, which describe the adaptive logic between the configuration as separate concerns. Level 3 requirements are the RE work done to select and configure DAS adaptation infrastructure. It identifies what adaptation infrastructure capabilities are needed to support level1 and level 2 models. Level 4 requirements are RE research into adaptation to identify the adaptation infrastructure need.

Requirements are clarified as 4 levels in LOREM approach, where they will be represented using corresponding requirements models of each level. For example, level 1 requirements will be represented using a collection of goal oriented models, while level 2 requirements will be represented using a collection of scenario based models.

RELAX
Requirements of self-adaptive system can also be represented by a formal language named RELAX [26,27]. RELAX is a requirements description language based on fuzzy set theory. It is able to deal with varying degree of uncertainty. RELAX introduces fuzzy semantic terms to add degree of flexibility to requirements. It includes fuzzy definition of timing, frequency, selection and etc. For example, by using a fuzzy term of timing, the requirements “This computer shall restart after Ben come in” will be transformed into “This computer shall restart as early as possible after Ben come in”. Which address uncertainty and it is called that the requirements are RELAXED. It is noted that the RELAX process can only be performed if partial satisfaction of requirements is allowed.

KPI Based Models
In the KPI based model [28], goals are acceptable system behaviors described in terms of KPI values. KPI, which stands for key performance indicator, are the foundation of this kind of adaptation model. KPIS are metrics that capture particular aspects of system performance and allow us to describe the system behavior. The other three elements of the adaptation model, namely component specification, adaptation specification and goal policy all depend on the set of KPIS.

Functional requirements in the KPI based model are also specified in term of goals. Goals are established for some or all of those KPIS, and the priority of goals will be defined. There are two batches of goals, which are exact goals and optimization goals. There are three kinds of exact goals, namely above goal, below goal and between goal. For instance, the goal “Goal goalName: kpiNameAbove threshold down”, called an above goal, will be met only if the values above the threshold. Also there are three kinds of optimization goals, namely close goal, minimize goal and maximize goal. For example, the goal “Goal goalName: MaximizekpiNameMinGain value Every period”, called a maximize goal, states that largest is the best.

Probabilistic Models
Probabilistic models can be used to verify if non-functional requirements are well satisfied [29].

In order to analyze the system against expected NFRs, the activity of the system will be described by sequence diagram first. To represent varying behaviors, the sequence diagram here is extended with stereotypes “variation point” and “variable”. Variation point represents a combination of alternatives for an adaptive part in the system. Variable stands for external service who may implement selection at run time. To benefit the verification of specified non-functional requirements, every transition in the extended sequence diagram will be annotated with tags stating quality data related to the NFRs. For example, if energy efficiency is one of the determined NFRs of the system, every transition in the sequence diagram will be annotated with a tag “energy=E” which expressed the amount of energy consumed by this transition.

Then the extended sequence diagrams will be transferred into parametric Markov models like parametric DTMCs by approach described in [30]. Here quantified non-functional requirements we would like to state are described by formulae written in formal language PCTL (Probabilistic Computation Tree Logic) and cost/reward formulae. For example, $R<=1000[	ext{F(State=C)}]$ represents the following property: ”The total energy consumption of eventually reaching state C is less than or equal to 1000”. $P>0.95[	ext{F(State=End)}]$ states that the probability of reaching the final state shall be greater or equal to 0.95.

3.2 Verification of Requirements
Self-adaptive systems execute in rapid changing context and the change is always unpredictable. These changes might lead to inconsistencies between requirements and architecture. As a result it is important for self-adaptive system to trace requirements and verify to what extent the requirements and expected properties are satisfied. Dynamic verification techniques and run-time monitoring shall be used to verify requirements conformance [8]. Dynamic monitoring occupy central role in verification methods and techniques. Requirements monitors should be installed to gather and analyze pertinent information about the system’s run-time environment [31]. The gathering information can be used to detect divergence events and judge how the requirements are met.

The first step of dynamic monitoring should be the precise determination of what to monitor for. It is the system requirements that induce the need for monitoring and determine what information is essential to be gathered [32]. The object to be monitored might be properties of operational environment or the system itself. the information of which could be collected from a sensor or a related component. Models of self-adaptive systems
might specify monitors in forms such as textual language, formulae and elements of a graph. The models would also address information stating which requirements the monitor is related to and which component the monitoring task is assigned to.

With the data collected through monitoring, requirements verification techniques will then be implemented. Verification approaches would be defined according to the characteristics of different modeling methods. Traditional verification methods, such as testing, model checking, formal verification, static and run-time analysis and program synthesis can be used in dynamic requirements verification. Static verification techniques and dynamic verification techniques can be combined to guarantee requirements conformance [33].

Goal-oriented Models
A goal-oriented model can be extended to address supporting information for run-time monitoring and requirements verification. In a goal-oriented model, the leaf-level goals could be assigned to specified components, which describes a leaf-goal can be realized by this component and the monitor of this leaf-goal can be performed by querying the related component directly. The high-level goals cannot be directly assigned to an agent. Therefore the verification of the satisfaction of high-level goals can be implemented by using propagation algorithm, which takes advantage of the refinement relationship and constraint relationship between goals. In order to verify the satisfaction of soft-goals at run-time, a subjective soft-goal which is not measureable might be quantified at first. For example, a soft goal like “cost efficiency” might not be observable directly, thus we shall implement a quantization process of this soft-goal. Besides, precondition of a hard goal identified in the goal model should also be monitored. The goal is only likely to be met when the precondition is true.

For a dynamic adaptation system, a variation of threat model can be used to identify sources of uncertainty, which could well benefit dynamic monitoring process. The obstacles that warrant the realization of specified goals are captured by the extended goal models, which might help to determine what to monitor for. In a goal oriented modeling method, high-level goals are identified first and then goal refinement is performed until leaf goals have been derived. On the contrast, identification of uncertainty factors which might prevent goals satisfaction will start from leaf-goals and propagate from the bottom up.

RELAX
RELAX specifies what to monitor by syntax 

\[ \text{"ENVIRONMENT <property">" and "MONITOR<property>" } \]

“Environment” refers to environment conditions which potentially affect satisfaction of specified requirements. The environment condition might be not observable. Thus we can conduct a quantization process first, and afterwards the quantized value which is observable and measurable will be specified as a monitor. That is to say ENVIRONMENT captures the “state of the world” and MONITOR defines properties to be monitored. For example, a relaxed requirement described by RELAX language is as followings, “A device SHALL be allowed to connect AS EARLY AS POSSIBLE after it requests it. MONITOR: average device connection times. ENVIRONMENT: number of devices requesting a connection.”. The environment “number of devices requesting a connection” is related to this requirement. And its observable property is “average device connection times”.

A Monitor Framework
This model [32] identifies high-level requirements and then decomposes them to sub-requirements. For each leaf-level requirement which could well be assigned to an agent such as system components, environment conditions or human, assumptions will be made about the current state of related agent. It is necessary to gather information on how the assumptions and requirements are met or not met by the current status. To benefit this, assumptions would be described with a quantified variable parameter. Accordingly the threshold of these parameters to guarantee requirements satisfaction will be predetermined. Requirements monitors shall be installed to gather the values of those parameters and detect divergences from pertinent assumptions that adversely affect adherence to requirements. If the monitoring results show the value exceeds the threshold, predefined remedies will then be employed.

KPI Based Models
KPIs, which are the foundation of KPI based models, induce the greatest need for monitoring. KPI based models [28] specified both goals and corresponding divergence events in the form of formula with KPIs. The divergence events would be defined before the execution of systems. One or two events will be generated for each goal. For instance, an above goal “k Above x” creates a deviation event “kpiBelow (k, x)”, which means that the value of k below threshold x is a deviation event need to be detected. As a result it is necessary to collect data used to calculate KPIs. KPIs might be related with one or more components, and pertinent information could be gathered from corresponding components. The KPI based adaptation model specifies components in predefined forms, which comprise the information about relationship between components and KPIs. At run-time, the monitoring mechanism will collect useful data from corresponding components and calculated value of KPIs. Then analyzer will analyze KPIs to detect whether predefined divergence events occur and verify whether goals are well satisfied.

Probabilistic Models
In order to verify the satisfaction of non-functional requirements such as “P=0.95[F(state=Success)]”, the sequence diagram which described the activity of this self-adaptive system will be transformed into parametric Markov models like DTMCs(discrete time Markov chain). Parametric DTMCs can be used to verify reliability properties, and parametric DTMCs with rewards can be used to verify cost properties such as energy and CPU. Parametric DTMCs can be regarded as state machines, where transitions are annotated with tags stating the probabilities of reaching target states. Due to the uncertainty of changing environment, the probabilities of transitions are not constants but might be varied at run-time. When different alternatives of a variation point are selected, the probability of a same transition differs. Thus, the probabilities of transitions will be expressed using parameters.

With the parametric DTMCs, the formula of verification for specified non-functional requirements can be generated. For example, for the NFR “\( P=0.95[F(\text{state}=\text{Success})]\)”, the corresponding property formula might be “Reliability=\( P^3*2*P1*P3^2*P2)\/(P3*P2^2-1) \)”. Here, P1, P2 and P3 are parameters in parametric DTMCs. They will be initialized at design phase and updated according to changes in operational context at run-time.
The parametric formulae will then be evaluated to analyze the current satisfaction of non-functional requirements.

### 3.3 Evolution of Requirements

Two issues of this aspect are considered here: the evolution of requirements, the synchronization between architecture and requirements model.

Self-adaptive systems have the ability to introspect their requirements model at run-time. Since domain knowledge and environment information of self-adaptive systems are incomplete at development phase, the requirements might dynamically evolve at run-time when more essential information is monitored. Requirements evolution is mainly triggered by changes in the environment, but it may also be triggered by stakeholders. Context changes might lead to new requirements which are not captured at design phase, or new trade-offs between potentially conflicted requirements. So when to evolve requirements and how to evolve requirements are important issues for self-adaptive systems. Guidance information for when and how to evolve requirements should be addressed in the self-adaptive requirements model.

At run-time, internal dynamic and external dynamic changes may lead to inconsistence between requirements and architecture of the system. Context changes are the main sources that may cause requirement violation [35]. If requirements evolve according to context changes, system will modify its behavior and structure according to the changed requirements model. By doing this self-adaptive systems could well adapt themselves to the changing environment. On the other hand, if the architecture of system changes from below layer, the requirements model need to be adapted in order to maintain synchronization. As such, the requirements model should address guidance information for how to adapt the system architecture to satisfy updated requirements model.

Among the requirements modeling methods we inspected, some modeling methods determine that evolution should take place when predefined divergence events occurs, for example, when monitor exceeds predefined threshold. To benefit the determination of how to evolve requirements model, some requirements modeling methods specify adaptation scenarios by an adaptation scenario model or by a formal language. As to how to evolve when certain deviation events occur, some modeling methods defined evolution methods bound to deviation events statically at development time. While other modeling methods will dynamically select optimal requirements evolution method. To carry on dynamic selection between candidate methods of evolution, multi-objective decision techniques are needed. The algorithms to choose most appropriate evolution method might be designed based on the candidates’ contribution to expected properties and non-functional requirements.

**Goal-oriented Models**

Self-adaptation behaviors such as self-healing, self-optimizing and self-configure can be described by an extended goal-oriented model. When it is monitored a leaf-level goal is failed or the pre condition of a specified goal is not met, the goal model of self-adaptive system will heal and reconfigure itself by finding a new path to realize top-level goals. These kinds of adaptation behaviors are mainly triggered by unanticipated changes in operational context. The monitor will continuously monitor environment conditions and the system’s running status in order to guarantee the most suitable and efficiency goal paths are selected. It means that a trade-off analysis between several potentially conflicting goals is needed at run-time to optimize the system performance.

A key characteristic of a dynamic adaptive system is that there may be numerous approaches to realize its high-level objectives. Once an adaptive change has been triggered, the selection between different goal paths will be performed. For example, a “top-down” goal reasoning algorithm [36] which based on alternatives’ contribution to high priority soft-goals, can be used to find the most appropriate particular realization at a particular point in time. Thus it is necessary to specify goals’ contribution relationship with soft-goals and priority relationship between soft-goals. Then multi-objective decision between different alternatives for satisfying the overall objectives of a system can be made dynamically at run-time.

**LOREM**

The LOREM based model [19] specifies evolution of requirements using adaptation scenario models. The scenario adaptation models are corresponding to the level 2 of requirements mentioned in previous section. It describes when and how to adapt from source static model to target static model. The scenario adaptation model includes three parts, which are monitor mechanism, adaptation mechanism and decision mechanism. Monitor mechanism specify what resources shall be monitored. The monitoring of resource such as monitoring of flow rate and monitoring of water depth are specified in the monitor mechanism as tasks. Decision mechanism specifies when the evolution takes place. When to evolve is predefined and it is related to the monitoring properties. For example, the evolution is defined to take place when flow rate exceeds threshold. Additionally, the adaptation mechanism decides how to evolve. Specifically, what requirements or goals need to be adjusted in order to adapt to the target static system. The scenario adaptation models shall be constructed for every possible adaptation scenario. They are constructed on the basis of behavior model of each static system and adaptation infrastructure.

**KPI Based Models**

The KPI based adaptation model [28] identifies possible adaptation behaviors before execution of the system. The description of each possible adaptation behavior specifies basic changes like addition/removal of components and connectors as well as complex change operations. The description also specifies when to employ this adaptation behavior and its impacts on related KPIs. Those specified adaptation behaviors would be selected to perform when defined pertinent events occur. For every possible event, a set of adaptation behaviors which might be performed will be identified at offline phase. Actually, a set of adaptation behaviors which can help returning the KPI employed in the event to a desirable state will be selected. For example, for the event “KpiBelow (k.x)”, adaptation behaviors which can increase the value of KPI k will be added to its related adaptation behaviors set. Eventually a corresponding adaptation behaviors set will be established for every event. An adaptation behavior could affect more than one KPI and accordingly it will be added to behaviors sets of more than one divergence event.

However, not all adaptation behaviors in the predefined set will be employed when this event occur at run-time. At the online-phase of a self-adaptive system, several divergence events might happen at the same time. It means that more than one goal is violated and
and probabilistic models are more suitable for complex systems. On the other hand, models like LOREM based models, KPI based models are embed in low uncertainty environment. On the other hand, models like LOREM based models, KPI based models and probabilistic models are more suitable for complex systems.

Probabilistic Models
The adaptation will be performed when NFRs are not well satisfied. For example, a non-functional requirement is stated as \( P >= 0.95 [F(state=Success)] \) but the value of probability calculated through parametric formula is far less than this value. When changes take place in the operational environment, the values of parameters in the formulae vary accordingly. This might lead to violation of non-functional requirements. The violation states that the alternative of variation points which is selected does not adapt to current environment and another alternative should be chosen instead.

The selection process will be implemented using an algorithm named hill climbing. A formula stating the total utility of a selected configuration will be determined at first. This formula is specified as a sum of all identified non-functional requirements multiplied by their weight. For cost and time issue, not all possible configurations of variation points will be calculated. So hill climbing algorithm will be implemented to find a sub optimal solution. Then the system can configure itself according to the chosen solution.

Aspect Oriented Models
AOM (aspect oriented modeling) [38] aims to derive a wide range of modes by weaving aspects into an explicit model. A mode means a possible runtime system configuration and transition may occur between different modes. Because of the high variability of self-adaptive systems, the quantity of modes and modes transition are too large. AOM for self-adaptive system [17, 37] considers to dynamically weave the aspect into explicit model at run-time. The weave can be triggered by human choice or be triggered dynamically by context. To discard changes happen in a cache period, an aspect called “Event Filter Aspect” will be woven into explicit model. When it is required to adapt system from a source model to target model, the validation of target model might be implemented by a simulation method. If the target model is valid, the adaptation toward the target model is automatically computed using model comparison.

4. CONCLUSION
We have identified three categories of challenges for requirements modeling methods to tackle with. And in the previous section, we have introduced how those seven models address key concepts and features of the three categories. Now table 2 gives a conclusion of the results of this preliminary study.

Table 2 Comparison between requirements modeling method

<table>
<thead>
<tr>
<th>Modeling method</th>
<th>Representation of Requirements</th>
<th>Verification of Requirements</th>
<th>Evolution of requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Specification of FR</td>
<td>Specification of NFR</td>
<td>Requirements Relationship</td>
</tr>
<tr>
<td>Goal-oriented Modeling</td>
<td>Goal</td>
<td>Soft goal</td>
<td>Refinement, contribution, constraints, priority</td>
</tr>
<tr>
<td>LOREM</td>
<td>Goal</td>
<td>Soft goal</td>
<td>Refinement, contribution, constraints</td>
</tr>
<tr>
<td>Probabilistic Modeling</td>
<td>Formulae</td>
<td>Priority</td>
<td>System property, Total utility</td>
</tr>
<tr>
<td>KPI Based Modeling</td>
<td>Formulae</td>
<td>Formulae</td>
<td>Priority</td>
</tr>
<tr>
<td>RELAX</td>
<td>Textual language</td>
<td>Textual language</td>
<td>Priority</td>
</tr>
<tr>
<td>Monitor Framework</td>
<td>Textual language</td>
<td>Refinement</td>
<td>Variable related to requirements</td>
</tr>
<tr>
<td>AOM</td>
<td>Possible changes</td>
<td>OCI or simulation</td>
<td></td>
</tr>
</tbody>
</table>

The models introduced above are respectively suitable for different scenarios. Models like monitor framework and goal oriented models are relatively simplified and easier for comprehension. They are more suitable to deal with less complex systems which are embed in low uncertainty environment. On the other hand, models like LOREM based models, KPI based models and probabilistic models are more suitable for complex systems embedded in operational context with a high degree of uncertainty. For instance, probabilistic model is suitable for modeling dynamic software product line.

Among those models, Probabilistic model is a quantitative model and others are qualitative models. Requirements are specified in terms of goals as graph elements in goal oriented modeling methods and LOREM method, while they are expressed as...
formulae in probabilistic model and KPI based model. In addition they are expressed as textual language in RELAX and monitor framework model.

As we know, an adaptive system keeps monitoring adherence to requirements, which can be analyzed using gathered information of the system’s run-environment. Pertinent information of a leaf-level requirement could be gathered from a sensor or an agent such as software components or humans in the system’s environment. It is the requirements that induce the greatest need for monitoring and offer the greatest challenge to determine precisely what to monitor for. The models introduced above specify what to monitor for in the form of textural language, formula or elements of a graph model. As for the verification of requirements, dynamic verification techniques and run-time monitor might be employed. Some models combine dynamic verification techniques and static verification techniques to guarantee that a system meets its requirements and expected properties. In goal oriented models, the satisfaction of high-level requirements could be verified by a propagate algorithm. In a KPI based model, the verification of requirements can be performed using the observed value of KPI and the corresponding threshold. In probabilistic models, non-functional requirements are specified in the form of paratactic formula, thus parametric model checking methods could be used for the verification.

When divergence between requirements model and current status occurs, self-adaptive system will perform certain adaptation behavior to guarantee the satisfaction of high-level goals. When a specified event occurs, corresponding adaptation behaviors to be employed are predefined in monitor framework model and LOREM methods. However, when divergence events are observed, models like goal oriented models and probabilistic models will perform certain algorithms to select optimal combination of adaptation behaviors among possible alternatives. In a goal oriented model, this selection will be implemented according to alternatives’ contribution to high priority soft goals. In a probabilistic model, the selection of optimal adaptation behaviors will be implemented with the target to maximize total utility. Moreover, in a KPI based model, the adaptation which can help restore abnormal KPIs will be selected.

A lot of current researches focus on extending goal oriented model to tame uncertainty in self-adaptive system. To date goal-oriented modeling method is relatively complete and it also has a extensive application range. So we suggest that goal-oriented modeling method may be suitable for internetware. Moreover the textual language RELAX can be integrated with modeling approaches to relax requirements for self-adaptive systems. In addition, KPI based model, which is mainly applied for distributed systems, might also be appropriate to be used in engineering process of Internetware.

These seven approaches can well address some challenges in the three aspects above. However, there are still a few more challenges in requirements engineering of self-adaptive systems which we need to tackle with in the future. Firstly, how to map design model of self-adaptive systems to implementation model is still lack of research. Secondly, it is necessary to study how to integrate design-time and run-time verification in order to provide better assurances. Moreover, aspects like the trace of requirements evolution, decentralization of control-loop and formalization of process for self-adaptive systems are all possible research fields in the future. Technologies such as dynamic software product line and aspect-oriented programming might provide new opportunities in the engineering of self-adaptive systems[8].

5. ACKNOWLEDGMENTS
This work is sponsored by the National Basic Research Program of China (973) under Grant No. 2009CB320701; the National Natural Science Foundation of China under Grant No. 61272163, 91318301.

6. REFERENCES


[34] Subramaniam, N., Drager, S., &McKeever, W. Engineering a Trustworthy Software System Using the NFR Approach.


