Towards a Goal-Driven Approach to Action Selection in Self-Adaptive Software

Mazeiar Salehie, Ladan Tahvildari†

Software Technologies Applied Research Group
Dept. of Electrical and Computer Engineering, University of Waterloo, ON, Canada, N2L 3G1

SUMMARY

Self-adaptive software is a closed-loop system, since it continuously monitors its context (i.e., environment) and/or self (i.e., software entities) in order to adapt itself properly to changes. We believe that representing adaptation goals explicitly and tracing them at run time are helpful in decision-making for adaptation. While goal-driven models are used in requirements engineering, they have not been utilized systematically yet for run-time adaptation. To address this research gap, this article focuses on the deciding process in self-adaptive software, and proposes the Goal-Action-Attribute Model (GAAM). An action selection mechanism, based on cooperative decision-making, is also proposed, which uses GAAM to select the appropriate adaptation action(s). The emphasis is on building a light-weight and scalable run-time model which needs less design and tuning effort comparing with a typical rule-based approach. The GAAM and action selection mechanism are evaluated using a set of experiments on a simulated multi-tier enterprise application, and two sample ordinal and cardinal preference lists. The evaluation is accomplished based on a systematic design of experiment and a detailed statistical analysis with ANOVA in order to investigate several research questions. The findings are promising, considering the obtained results, and other impacts of the approach on engineering self-adaptive software. Although, one case study is not enough to generalize the findings, and the proposed mechanism does not always outperform a typical rule-based approach, less effort, scalability and flexibility of GAAM are remarkable.

KEY WORDS: Self-Adaptive Software, Goal-Driven Model, Run-Time Action Selection

1. Introduction

Regarding the scale of existing software applications, the dynamic environments in which they function, as well as variable system requirements, software operation management is often costly,

*This paper is an extended version of a short paper, titled as “A Weighted Voting Mechanism for Action Selection Problem in Self-Adaptive Software”, published in proceeding of Self-Adaptive and Self-Organizing systems conference (SASO’07).
†E-mail: {msalehie, ltahvild}@uwaterloo.ca

Received May 2009
Copyright © 2010 John Wiley & Sons, Ltd.
Revised June 2010
time-consuming, and likely to be error-prone. This problem can be attributed to the open-loop structure of many existing software systems, and the need for continuous human supervision. In fact, a broad range of applications from large-scale enterprise applications to small stand-alone software systems suffer from the lack of a closed-loop mechanism to adapt to changing conditions. Therefore, there is an increasing demand for adaptivity of software systems at run-time. Self-adaptive software aims at addressing this need in a way which “evaluates its own behavior and changes behavior when the evaluation indicates that it is not accomplishing what the software is intended to do, or when better functionality or performance is possible.” [26]. Although this definition addresses run-time changes regarding functional and non-functional concerns, this article and a remarkable part of researchers in software engineering community focus on non-functional aspects.

Adaptivity is normally described on the basis of a set of properties called self-* properties [24]. Among numerous self-* properties in the literature, the set of self-configuring, self-healing, self-optimizing, and self-protecting properties is well-known (for example see [24]). These properties are mainly aligned with non-functional requirements and software quality factors [46, 50]. For example, self-optimizing mainly addresses the system performance. Therefore, as in goal-driven requirements models (e.g. SIG [10]), these properties can be mapped to quality goals, which in turn can be decomposed into smaller sub-goals. These links to quality factors form the basis of this work, in order to model adaptation goals, and to design decision-making mechanisms. The notable point is that although in some research works functional requirements and emergent behaviors have been also covered by adaptation goals, these goals are not addressed in this article.

Self-adaptive software requires planning or selecting proper adaptation actions, and performing the actions to change the adaptable software. Generally, the adaptation mechanism is decomposed into several processes for monitoring software entities (self-awareness) and the environment (context-awareness), analyzing significant changes, planning how to react, and executing for the decisions to take effect [24]. In this article, the more general term “deciding” process, is used instead of “planning” process, in order to distinguish this process from a merely automated planning approach. Most of the existing solutions, including autonomic computing [18], realize the adaptation processes in an external adaptation manager, which is separated from the application logic [50]. An adaptation manager realizes the four mentioned processes to control the behavior of adaptable software. The application logic is implemented in adaptable software that exposes the required sensors and effectors for adaptation [50]. Adaptable software can also be called managed resource(s) [18] or adapt-ready program [35].

Some researchers in this area (among others [25, 27]) subscribe to the view that it is essential to explicitly represent adaptation goals, and to incorporate these goals in the adaptation processes. The belief is that a goal-based adaptation mechanism could not only be effective, but also be traceable and trustable. Analyzing goals at run-time provides requirements traceability, or “requirements reflection” [12]. This could also lead to trustability. Salehie et al. discuss that trust have several meaning in the domain of self-adaptive software systems, and one of those meanings comes from transparency and visibility [50]. McCann et al. point out that trust can be built via revealing significant information about the system status and the visibility of adaptation processes [34, 17]. This explains why predictability can be considered as a major factor for placing trust upon self-adaptive software.

The main contribution of this article is building a goal-driven model and its associated decision-making mechanism for run-time action selection in the adaptation manager. The main idea is to keep adaptation goals, related to quality goals, “alive” at run-time to choose among alternative actions.
Among the four adaptation processes, this article deals mostly with the deciding process. Of course, tracing goals for identifying their satisfaction or denial status is also related to the analyzing process. Among different forms of decision-making denoted by Roy [43]—selecting, sorting, ranking and description—the deciding process in self-adaptive software is closer to the selecting format. The deciding process basically addresses an action selection problem in order to select a proper action from a finite set of alternatives. A novel model, called Goal-Action-Attribute Model (GAAM), is proposed for tackling the action selection problem.

Goal-based approaches to adaptation are not widely addressed in many research efforts in the self-adaptive software area. The well-established goal models in Requirements Engineering (RE) are designed for development time. They aim at analyzing different design decisions for software engineers instead of focusing on run-time evaluation of goals for automated decision-making. GAAM is inspired by the available goal models in RE, but the emphasis is building a run-time model and preparing a decision-making mechanism to reason upon for adaptation. Therefore, although GAAM has some similarities with these goal models, it has at least two differences: first, monitored attributes activate adaptation goals in GAAM, while in design-time goal models, such as SIG in NFR framework [10], operationalizations and design decisions change statuses of goals. Second, in design-time goal models, the objective is to analyze the impact of operationalizations on goals (i.e. specify their satisfaction levels), while in GAAM the objective is to select an action to change the current status.

Two points are notable about the evaluation of GAAM and the action selection problem. First, one of the shortcomings that software engineering for self-adaptive applications face is the lack of case studies. Some people use systems which are not open source or accessible by the others, and other cases are only very limited scenarios for a single quality aspect in toy applications. Although, we had some limited experiments on real applications, we decided to evaluate GAAM and its action selection mechanism with a simulated model. Second, many case studies on self-adaptive software do not deal with application-level adaptation. The actions in those works change something in middleware, server, network, or even operating system. Therefore, in this article adaptation actions are limited to application level. This does not mean that GAAM cannot include other actions. There is no assumption about the type of actions and we selected the application-level adaptation only because it has not been addressed extensively so far.

The rest of this article is organized as follows. Section 2 specifies the problem statement, and Section 3 elaborates GAAM and proposes a goal-based action selection mechanism for self-adaptive software. For evaluation, the proposed model is applied to an experimental model of a news web application in Section 4. Section 5 reviews the obtained results and the statistical analysis. Section 6 discusses notable related work. Section 7 draws some conclusions, and finally Section 8 provides several ideas for future enhancements.
2. Problem Statement

The focused problem in this article is run-time goal-based action selection in self-adaptive software, which includes three key elements. First, \( G = \{ g_i, i = 1..m \} \) are the objectives\(^*\) of the adaptation; for example, maximizing performance or more specifically guaranteeing a certain level of response time. As noted before, these goals are related to non-functional requirements. Second, \( AT = \{ at_i, i = 1..n \} \) are the measurable properties of adaptable software, that are required to trace goals in \( G \). For example, the model needs to capture average end-to-end response time to evaluate the goal of achieving a possible maximum performance. Finally, adaptation actions (briefly called actions in the rest of the article) are denoted as \( AC = \{ ac_i, i = 1..p \} \). These actions are possible run-time changes provided by effectors in the adaptable software. These effectors are instrumented at development time, or added at deployment or run-time. For instance, an action would be disabling a service to provide resources for more important services. The problem is defined as:

*Given an adaptation goal set \( G \), an adaptation action set \( AC \), and an attribute set \( AT \) from a software system, the problem is how to build a goal-action-attribute model, and to select the appropriate action \( ac_i \) at run-time to satisfy goals under different conditions.*

In other words, the problem is first, how to define goals, actions, attributes and their relationships in a goal-action-attribute model, and second, how to design a decision-making mechanism based on this model. The desired model defines a graph linking the set of actions \( AC \), goals \( G \), and attributes \( AT \). Then, the mechanism selects an action based on the model, particularly the goals. Because goals play a key role in this problem, the problem can be viewed as a goal-driven coordination problem, which means the decision-making mechanism should coordinate the preferred actions of these goals.

In this article, we specifically focus on the following desired characteristics for a goal-driven deciding process:

- **Explicit goals at run-time**: The process is required to use explicit goals navigating the run-time decision-making. In other words, adaptation goals are “alive” at run-time in contrast to goals in common goal-driven RE models. Based on these goals, policies and priorities can be defined and tuned for achieving the desired behavior. This characteristic improves traceability of the system for administrators, which in turn leads to build trust and better evaluation. A notable point is that the way goals are represented explicitly is also important. A more expressive goal model enables a more precise goal tracing and ultimately provides more information for action selection mechanism.

- **Multi-objective decision-making**: In most cases, several goals are involved in adaptation requirements specifications (e.g. performance and security). The deciding process needs to take into account all of these goals, and it also needs to coordinate and resolve conflicts among them.

- **Flexibility**: Goals and their priorities can be changed at run-time. Of course, these changes can be applied to the extent that satisfying and evaluating the new goals are still possible using provided sensors and effectors in adaptable software.

\(^*\)Objective and goal are different in some contexts. For instance, Keeney et al. use objectives in a higher level of abstraction [23]. In this article, without loss of generality, we assume goals and objectives are the same. Therefore, we use them interchangeably hereafter.
The mentioned problem has some similarities with decision-making problems in Robotics and agent-based systems. The related work Section addresses these similarities and point out how this work was inspired by the solutions given for those problems. It is notable that for this article, the focus is on single action selection in each round of decision-making instead of automated planning for a sequence of actions. Of course, as will be discussed later, the proposed approach can be extended to selecting multiple actions, or a course of actions but not chained together through pre and post-conditions, as we see in AI planning. Another point is that the proposed approach does not involve an on-line learning method to build GAAM at run-time. The model is built when we design the adaptation manager, even though the enclosed entities and their relationships can be changed at run-time by human agents or the adaptation manager.

3. Proposed Approach

This section details the proposed approach for tackling the action selection problem. The first part sets out to define the model relating to goals, actions, and attributes. The next part elaborates the decision-making mechanism, which utilizes the goal-action-attribute model to select an action. In the rest of this article, this mechanism is called the action selection mechanism.

3.1. Goal-Action-Attribute Model (GAAM)

The main purpose of the Goal-Action-Attribute Model (GAAM) is to represent the key entities of the action selection problem (goals, attributes, and actions), and to relate them in a well-defined structure. The information about these entities is collected from the adaptable software and stakeholders, as is schematically depicted in Figure 1.

![Figure 1. Building Decision Model, GAAM, in Adaptation Manager](image-url)

Adaptable software is a system which exposes sensors and effectors for adaptation, similar to a controllable plant in a control system. The sensors are used to collect information about attributes (e.g., measuring resource utilization). Effectors facilitate applying changes to software artifacts (e.g.,
redeploying a component) to attain the goals. The stakeholder is defined as “a person or organization
who influences a system’s requirements or who is impacted by that system” [15]. In this situation, roles
are used instead of individuals. While in requirements engineering, the whole set of roles which impacts
the development and operation of software is taken into account (e.g., tester, and project manager), in
this article only the operation time is considered. Thus, the typical roles in this context are end-user,
business owner, and administrator.

Attributes $AT = \{\text{at}_i, i = 1..n\}$ represent measurable properties of adaptable software. Attributes
can have different types like integer, time, etc. Moreover, each $\text{at}_i$ belongs to a specific entity (e.g.,
method or component) or a level (e.g., application or a specific subsystem). Other details about sensors
may be required. For example the period of updating an attribute in case of synchronous monitoring,
which may be different for each $\text{at}_i$. Each $\text{at}_i$ in this set is either directly measurable from the adaptable
software or can be evaluated through a proxy variable. In GAAM, we deal with measurable attributes
that in fact are related to goals, as we will see in the rest of the article. However, some of these attributes
may need pre-processing before feeding into GAAM. For example, applying a moving average function
on the stream of data (e.g., average end-to-end response time) may be required. Since these operations
are related to monitoring and analyzing processes, more details are out of the scope of this article.

Goals are represented using a hierarchy, from high-level to low-level goals. Stakeholders often
start to articulate high-level goals by specifying the desired behaviors of the system. These goals are
decomposed into low-level goals, which are more likely to be related directly to measurable attributes.
The leaf goals are directly related to attributes, such as “achieve minimum possible response time” and
“achieve 80% resource utilization”. Goals $G = \{g_i, i = 1..m\}$ in the GAAM can be any of the high
or low-level goals. The way we present goals in GAAM depends on a trade off between performance
and expressiveness we target for the action selection mechanism. For example, a rich goal structure
generally needs a more complex goal tracing method. Of course some heuristics can help us improve
the performance. To be a light-weight model, GAAM may include only leaf goals which are traceable
directly from attributes. However, it is possible to incorporate the entire goal hierarchy in GAAM. In
this case, tracing starts from leaf goals, but propagates upward to determine the satisfaction levels of
high-level goals.

The notable question is how to specify the adaptation goals. Generally, due to the fact that the main
objective of a software application is fulfilling its requirements, the requirements specification is the
starting point when thinking about modeling adaptation goals [28]. Since, the expected behavior of
self-adaptive and autonomic software is normally specified by self-* properties, these properties also
play an important role in determining the target goals. Self-* properties are related to software quality
factors and non-functional requirements [46]. Therefore, the well-established models and methods
in requirements engineering (e.g. SIG [10]), and quality modeling and measurement techniques (e.g.
in performance engineering [20]) can be employed. Moreover, goals can be structured based on their
relationships with the corresponding adaptable software’s architecture. System-level goals are assumed
to be high level, and may be decomposed into goals at the subsystem, component, method, or parameter
levels.

Goals in GAAM have the two following important properties:

- **Activation Function**: Goals are stimulated by attributes, and the activation function $f_i(.)$
determines whether $g_i$ is activated or not. Activated goals are eligible to participate in the action
selection mechanism. An ACtivation Function (ACF) vector is defined as $ACF = \{f_i(\cdot), i =$
Another parameter for activating goals is to set a rate by which goals participate in decision-making. This option is suitable for synchronous events (e.g., the expected number of user requests in specific hours) or for goals that do not need to be satisfied frequently.

- **Priority**: The priority $p_i$ determines the weight of the goal $g_i$ in the GAAM model, and impacts its influence in the action selection mechanism. Priorities come from stakeholders’ opinions, and are represented by a Priority Vector $PV = \{p_i, i = 1..m\}$. In fact, $PV$ quantifies the order of goals in $g_i \leq g_j \leq ... g_k$. It is assumed that $\sum_{i=1}^{m} p_i = C$, where $C$ is a constant value (e.g., 1 or 100). The process of extracting priorities from requirements has been investigated in requirements engineering in different ways. A suggested method is to use the Analytic Hierarchy Process (AHP) by pairwise comparison of goals [21, 22, 44]. In this way, it is possible to check the consistency ratio of both the comparisons and priorities.

Adaptation actions $AC = \{ac_i, i = 1..p\}$ are changes applicable to adaptable software entities using the provided effectors. These actions usually include some preconditions. Before considering an action eligible to be selected, its preconditions should be satisfied. The precondition set is denoted as $PC = \{pc_i, i = 1..p\}$, where each $pc_i$ can be a vector of preconditions for each action. $PC$ often includes conditions on adaptable software resources, architecture, or attributes. Some actions may be composite, like a workflow. In this way, preconditions of an action are prepared by the previous action in the workflow. For example, redeploying a component in J2EE applications needs storing the state, undeploying the component, deploying a new component, and restoring the previous state in a chain of actions. In GAAM, each $ac_i$ can be a single or composite action, and the precondition set refers to the first set of conditions in the chain of actions.

Actions also have postconditions because of their impact on goals. GAAM does not define these conditions in the action entities, but instead, they are considered in the connections between goals and actions. In fact, postconditions are the basis of setting action preferences for goals in $G$. The goal preferences will be discussed further later. Actions, similar to goals, can be related to different entities in different levels of adaptable software. For example, an action may change the composition of components, while another action tunes a parameter inside a component.

In order to define the GAAM, the aforementioned entities need to be linked together. Therefore, the following two sets of parameters are required:

- **Relating goals to attributes**
  - **Activation Matrix**: The matrix $AM = \{\omega_{ij}, i = 1..m, j = 1..n\}$ shows the relationships among $m$ goals and $n$ attributes. The values show how much each attribute affects any of the goals. Activation values can be boolean, symbolic like in SIG (‘++’ or ‘-’), cardinal values, or even fuzzy terms (e.g. high positive).
  - **Aspiration Level Matrix**: The matrix $ASL = \{\tau_{ij}, i = 1..m, j = 1..n\}$ specifies the aspiration levels for each attribute $at_j$ of each goal $g_i$. In fact, the values determine what the desired levels of attributes of each goal should be.

- **Relating goals to actions**
  - **Preference Matrix**: The matrix $PM = \{\rho_{ij}, i = 1..m, j = 1..p\}$ shows action preferences of each goal $g_i$. As noted before, the postconditions of actions are defined
as goal preferences. In fact, each goal \( g_i \) determines its preferred actions based on their impacts. The preferences are in the form of three relationships \( \succ, \prec \) and \( \sim \). Element \( \rho_{ij} \) in \( PM \) can be defined using ordinal or cardinal utility functions. These two forms of defining preferences will be discussed more in Section 3.2.

![Figure 2. Representing GAAM as a Graph](image)

The GAAM specifies a graph \( G = \{V, E\} \), where vertices are \( V = \{G \cup AT \cup AC\} \) and edges are \( E = \{AM \cup PM \cup ASL\} \), as depicted in Figure 2. The model is built and used in the adaptation manager to trace the goals and keep track of their action preferences in different conditions. As noted before, details of the GAAM design depend on the action selection mechanism and tracing method. The key point is that the GAAM is able to fulfill the desired requirements of a goal-based deciding process, described in Section 2. It is capable of presenting multiple explicit goals, the first and second conditions for the desired goal-driven deciding process (discussed in Section 2). Moreover, depending on how the GAAM is implemented, it can be flexible enough to allow adding/removing goals, actions, and attributes at run-time. The goal preferences can also be dynamically changed at run-time.

### 3.2. Action Selection Mechanism

The GAAM is generic enough to allow different decision-making mechanisms to work with it. In this article, our proposed action selection mechanism is a cooperative method based on the weighted voting of activated goals (i.e. denied goals). In fact, the action selection problem is modelled as a game between goals, in which each goal wants to select its preferred action(s). The action selection mechanism coordinates goals to find satisfactory and good enough actions for goals at run-time.

Here, we assume that the GAAM is continuously being traversed from attributes to actions by a specific adaptation period \( T_{ad} \). This implicitly means that a polling method is used for monitoring; although an event-based method can be used as well. Using the event-based method does not drastically change the mechanism: attributes stimulate goals by sending event notifications, and activated goals select their preferred actions. A notable point is that the analyzing process is also involved in the traverse. Tracing goals is a part of analyzing the adaptable software and detecting any anomalies (i.e., denied or activated goals). Therefore, the action selection mechanism utilizes a combination of information from the analyzing and deciding processes.

The following intermediate parameters are used by the action selection mechanism:
• **Activated goals \( \hat{G} \):** This set shows which goals have been activated regarding their activation function in \( ACF \). For the following action selection mechanism, we assume a goal is activated or not. However, if we consider a satisfaction level for each goal, it is possible to incorporate this level in the action selection.

• **Selected attributes \( \hat{AT} \):** This matrix shows two pieces of information: first which attributes denied the aspiration level specified by \( ASL \), and second how much these attributes stimulate the corresponding goals regarding the values in \( AM \). If we present all the parameters as cardinal values, \( \omega_{ij} \) are weights that indicate how severe each attribute stimulates each goal. For example, \( a_{ti} \) can be activated regarding its aspiration levels for \( g_j \) and \( g_k \), but impacts higher (e.g., 10 times more) on \( g_k \). In short, the outcome matrix \( \hat{AT} \) indicates which attribute stimulate which goal and how much in the current situation.

• **Doable actions \( \hat{AC} \):** This set shows which actions are possible to apply in the each situation.

• **Votes \( \bar{AC} \):** The ordered list of actions for goals is represented in a vote matrix \( \bar{AC} \). For each goal, actions can be only ordered (ordinal preference) or have a scale or severity of preference (cardinal preference). These two types of preferences are discussed in the rest of this section.

Figure 3 illustrates the flow of preparing voters and candidates, and the action selection mechanism. The steps are as following:

**Step 1- Filtering attributes:** In this step, the selected attributes \( \hat{AT} \) are specified. The mechanism initially checks each attribute status regarding the \( \tau_{ij} \) values in \( ASL \). Then for those who failed the check (i.e., not as expected by aspiration levels), the mechanism applies related \( \omega_{ij} \) values in \( AM \). Note that since the monitoring process is not a part of GAAM, we assume any pre-processing on sensed data is accomplished before this step.

**Step 2- Tracing goals:** In this step, the mechanism evaluates goals to determine which one is activated (denied and need to be satisfied). The activation functions receive the latest situation from \( \hat{AT} \) and decide each goal’s activation status based on \( ACF \). The outcome is the set of activated goals \( \hat{G} \). As mentioned before the goal activation function can be implemented using a qualitative or quantitative approach depending on how \( AT \) and \( ACF \) are defined. A simple and straightforward method is to activate a goal when the sum of associated selected attributes in \( \hat{AT} \) for a goal passes a certain level (i.e., activation level).

Figure 3. Action Selection Mechanism

---

Copyright © 2010 John Wiley & Sons, Ltd.

*Softw. Pract. Exper.* 2010; **00**:1–28

Prepared using speauth.cls
Step 3- **Specifying doable actions:** Doable actions are identified by checking the set of preconditions $PC$ for the actions. The set of $AC$ are the set of candidates that can be selected by activated goals in $G$.

Step 4- **Casting votes:** In this step, activated goals generate their preferred lists of actions (votes). This can be different in each goal, for example, based on a fixed set of preferences or by dynamically-generated preference lists using the history of previously performed actions. In fact, due to the differences between adaptation goals, in many cases it is essential to have such a capability. The default preferences are specified in $PM$, but can be changed dynamically. The outcome of this step is $\bar{AC}$.

Step 5- **Aggregating votes:** The last step, is aggregating the votes casted by goals in $\bar{AC}$ based on a voting schema. Selecting the voting schema depends on the structure of the preference lists and how those preferences are used. In this article, ordinal and cardinal preference lists are used. The following subsections discuss these two preference structures and some options for the voting schema.

### 3.2.1. Preferences with Ordinal Utility

One way to define action preferences is using the *ordinal utility* form. In this way, each goal $g_i$ presents its preference as a relationship among actions using the operators $\succ$, $\prec$ and $\sim$. For example, a goal $g_i$ may define $ac_i \prec ac_j \sim ac_k$, to express that it prefers $ac_j$ and $ac_k$ over $ac_i$, if it is eligible to decide. The ordinal utility keeps the preference structure simple, although it does not clearly show how much an action is preferred to another.

The postconditions of each action play a key role on its impact on goals, and its rank in the preference lists. For some goals, it is impossible or difficult to quantify the impact of actions on goals, but it is feasible to judge (by stakeholders or administrators) their order. For example, assume we have maximum user satisfaction in the adaptation specification. Evaluating this goal depends on the end-users working in each period of time, and it is not easy to say exactly which actions improve or deteriorate the satisfaction level. However, it is possible to say that the users prefer actions providing more security to those guaranteeing a good response time.

For example, suppose there are four goals $g_1$ to $g_4$, and four alternative actions $\{ac_1, ac_2, ac_3, ac_4\}$. Goals have priority values $PV = \{3, 2, 2, 1\}$ for $g_1$ to $g_4$ respectively. The following preference structure may be defined by these goals in $PM$:

<table>
<thead>
<tr>
<th></th>
<th>$g_1$</th>
<th>$g_2$</th>
<th>$g_3$</th>
<th>$g_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ac_1$</td>
<td>$ac_4$</td>
<td>$ac_2$</td>
<td>$ac_3$</td>
<td></td>
</tr>
<tr>
<td>$ac_2$</td>
<td>$ac_1$</td>
<td>$ac_4$</td>
<td>$ac_4$</td>
<td></td>
</tr>
<tr>
<td>$ac_3$</td>
<td>$ac_2$</td>
<td>$ac_1$</td>
<td>$ac_1$</td>
<td></td>
</tr>
<tr>
<td>$ac_4$</td>
<td>$ac_3$</td>
<td>$ac_3$</td>
<td>$ac_2$</td>
<td></td>
</tr>
</tbody>
</table>

For this form of preference structure, various voting mechanisms are applicable. The most common ones are the plurality and Borda count mechanisms [54]. In plurality voting, the winner is simply the candidate that the majority of voters preferred to the others. Borda count extends this method to involve the entire preference list by assigning an index number to each element in the list [54]. The winner is
the candidate that has the highest sum for all voters. The advantage of this method is that it is possible to know the rank of each candidate, and its difference with the others in the voting results.

The fairness of the voting schemes has been investigated by researchers, and is out of the scope of this article. We consider only two characteristics called monotonicity and Pareto optimality. The former means that if a goal raises the rank of a winning action, it remains the winner, and when a goal lowers the rank of a loser action, it remains the loser. The Pareto criterion means that when every goal prefers \( ac_1 \) to \( ac_j \), \( ac_j \) is not selected [38].

For the above example, assume that the outcome of this weighted voting system is a single selected action. The following table shows the social choice for this case using three different voting schemes. Pairwise comparison is another applicable scheme which compares the candidates pairwise to find the winner. The order of comparison (called agenda) is important in this scheme, and for this reason it may have different outcomes.

<table>
<thead>
<tr>
<th>Voting Schema</th>
<th>Winner(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plurality</td>
<td>( ac_1 )</td>
</tr>
<tr>
<td>Borda count</td>
<td>( ac_1 )</td>
</tr>
<tr>
<td>Pairwise comparison</td>
<td>( ac_1 ) and ( ac_2 )</td>
</tr>
</tbody>
</table>

The Plurality and Borda count voting methods, which have Pareto and monotonicity properties, choose \( ac_1 \) as the winner. Pairwise comparison also has these properties, but sometimes chooses \( ac_1 \) (depending on the agenda). Among these voting schemes, the Borda count method uses all of the elements in each preference list without using an agenda or any other extra parameter. The Borda count method also produces more reasonable results for different weighting schemas. For instance, for equal weights, the Borda method still chooses \( ac_1 \), while Plurality voting has no choice. For the latter, conflicts exist among all four candidates. We selected the Borda count schema as the default voting mechanism for the GAAM model, although the voting schema can be easily changed.

3.2.2. Preferences with Cardinal Utility

If it is possible to quantify how much an action is preferred (e.g., an absolute value in the range of 0 to 100), then the preference structure can be defined by cardinal utility. In this case, a voting schema can be applied to add the cardinal vote of each action, and select the action with the maximum value. Sometimes, this is called intensity voting [54]. It is also possible to multiply the cardinal utilities of candidates (Nash voting) [38]. Since this voting method needs to normalize the multiplication values, it is not appropriate in some cases. For this article, intensity voting is the default voting schema for cardinal preference lists.

4. Experimental Evaluation

A simulated multi-tier enterprise application is used to evaluate GAAM and the proposed action selection mechanism. The implemented model for simulation is general enough to be used for other enterprise application (e.g., in the e-commerce domain), but we cannot claim that the following experiments evaluate GAAM for any system or any situation. The objective is to assess GAAM for an interesting case study with different settings. The case study has no special assumption about users.
and architecture of the system, and the settings were designed to put the system in different situations to avoid biased assessment. In this experimental evaluation, particularly the following research questions are taken into account:

- **RQ1** - Impact of adaptation: What is the impact of adaptation on the system’s goals? Does it improve the goal satisfaction level (i.e., positive impact) comparing with the non-adaptive case in all situations?
- **RQ2** - Ordinal vs. cardinal utility: Which type of utility is better for determining preferences? While the cardinal utility needs more effort to be extracted and tuned, does it necessarily lead to a more effective adaptation?
- **RQ3** - Goal-driven vs. rule-based adaptation: Does the proposed goal-driven action selection mechanism outperform a rule-based mechanism?

Each of these questions aims at comparing two terms. To make this possible an evaluation function is needed. This function sets out to measure the effectiveness of each method, and the satisfaction levels of the goals during the system’s operation. One option is to measure how many times the goals in GAAM are activated, but were not satisfied for several adaptation periods $T_{ad}$, by considering the priority vector $PV$. This method is not precise enough for comparing different adaptation methods. A more appropriate method is to assess the deviation from the satisfied state of each goal, using a formula similar to the goal programming method [19]. The deviation for each goal is evaluated by $e_j = 1/m \sum_{i=1}^{m_i} (d_i^+ + d_i^-)$. The normalization factor $(1/m)$ has a value between 0 and 1. The objective is to minimize this deviation measure (zero is the ideal value). For each run $k$, the global evaluation function is $E_k = 1/n \sum_{j=1}^{n} p_i * e_j$.

### 4.1. Adaptation Scenario

For the experiments, a sample news web application was used. The motivation for choosing such an application was the problems encountered by news web sites in the US after 9/11. The news web sites’ usage skyrocketed on that day, and continued throughout the week. A Los Angeles Times report offers a few numbers: “MSNBC saw a tenfold increase in traffic, with as many as 400,000 hits at any point. CNN.com surged to 162.4 million page views in 24 hours from a 14 million average.” By the day after, the sites were more prepared, and congestion eased. By early afternoon of the next day, some web sites reported even heavier traffic than the preceding Tuesday. The CNN web site was almost overwhelmed by demands. In this situation, the technical staff redesigned the site in an effort to remove all of the extraneous information, and concentrating on the bare facts. The CNN web site, which normally includes pages with various links and graphics, was reduced to one breaking news web page. While the changes were made to the application by the administrators, we are interested in adding an adaptation manager to accomplish this task. This scenario is appropriate to show how effective the adaptation mechanism would be for satisfying the goals of end-users and administrators.

In fact, during the 9/11 event, administrators and network managers tried to manage the system by tuning the parameters and by applying server and network-level actions. But application-level actions were missing, which were applied manually in the CNN case. Therefore, the interesting question for us is how adaptation actions at the application level would impact the system behavior. Although, other actions are applicable to this case, we focus only on the application-level adaptation actions.
4.2. Experimental Model

The experimental model is a simulated news web application. The model is based on a generic multi-tier enterprise application, which can easily be used for other domains such as e-commerce. Figure 4 illustrates the schematic view of the experimental model, and the data flow. The model is a network of queues (or queueing network [20]), which is built as an open network with infinite population. The component model is a queue-server model, which is implemented in the Simevents toolbox [32] of Matlab/Simulink. The queueing model is one of the common models in software performance engineering [3], and can be applied more easily to component-based applications in particular [9]. In this model, the stakeholders are two groups: i) business owners and administrators, and ii) end-users. The stakeholders define their expectations as goals for the system.

![Figure 4. The Experimental Model of a Multi-tier News Application](image)

Each component has a priority queue based on the service time for requests. The inputs and outputs are as follows:

- **Inputs**: Service time, failure duration, failure probability, restart signal, and input traffic
- **Outputs**: Average end-to-end response time, number of requests served per second, component state, average number of requests in queue, and output traffic

For traffic generation, two parameters are taken into account: *inter-arrival time* (IATime) and *service time* (SVTime). For the conducted experiments, exponential and Weibull distribution functions were used for IATime. For SVTime, an exponential distribution was used with a changing mean to generate burst traffic, as was the case in 9/11. The three main data formats in news are video, image, and text, which are delivered to end-users with high/low quality or normal/small size. The quality and size factors have some impact on the SVTime. As will be described later, five combinations of these formats with different levels were used in the experiments, which resulted in various SVTime values.

For this model, the assumption is that only the business tier components, deployed on the application server, may fail under the high traffic load. In practice, most of the load is on the application server.
and database server, depending on the nature of the deployed application. Since, most database servers are empowered by recovery mechanisms, we did not consider this case in the experiments. The failure rate is generated by a probability distribution function for each component. For the experiments, the probability function was chosen as exponential with a constant duration time. The restart time was also considered to be constant. However, because the model is required to be evaluated for a no-adaptation case as well, restart should also be available in that case (manual administration). Therefore, a restart action has a longer $T_{ac}$ (it is three times larger than the automated restart). The restart mechanism was designed as a micro-reboot mechanism as discussed by Candea et al. [7]. The micro-reboot was modelled as a queue-server model which is enabled by the restart action signal. The failure, restart and warmup states of the component were modeled by a state-flow diagram in the Matlab StateFlow toolbox [33]. Interested readers can refer to [47] for more details on this state-flow diagram. In a set of experiments, a timeout mechanism is used to drop requests after a certain time. The timeout value is set at the traffic generator, and each component has an output which reports the dropped requests.

Because the focus of the experiments is evaluating GAAM, this model and the action selection mechanism are emphasized in the simulation. GAAM covers deciding and analyzing processes but the entire adaptation manager needs to be implemented for experiments. For the sake of simplicity the implemented acting process is highly coupled to the developed effectors. For example, the restarting effector is controlled by a state machine developed in the state-flow toolbox which is attached to components in the simulated model. The monitoring process is also attached to sensors to collect and pre-process data.

### 4.3. GAAM Specification

Table I lists the attribute set $AT$ used in the experiments for the news application model. Throughput is calculated based on the number of requests served per second for the whole system. While the simulation gives us the response time and throughput for each component separately, only the end-to-end value is used for these attributes (respectively for $at_2$ and $at_5$). End-to-end response time is represented by $at_2$ that is the total time from sending request to receiving the response. The network connection is also modeled by two queue-server components for send and receive on a bandwidth-limited channel, but due to replicating experiments the network delay is mostly similar for different settings. Therefore, in fact $at_2$ measures the round-time between entering and exiting the web tier.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>News quality (part of $at_1$)</td>
<td>Video resolution: {High, Low}, Image size {Normal, Small}</td>
</tr>
<tr>
<td>News data type (part of $at_1$)</td>
<td>{Video, Image, Text}</td>
</tr>
<tr>
<td>End-to-End Response time ($at_2$)</td>
<td>Average round-trip response time in millisecond</td>
</tr>
<tr>
<td>User load ($at_3$)</td>
<td>Number of user requests</td>
</tr>
<tr>
<td>Component state ($at_4$)</td>
<td>{Active, Failed}</td>
</tr>
<tr>
<td>End-to-End Throughput ($at_5$)</td>
<td>Number of requests served in a second</td>
</tr>
</tbody>
</table>
For the user load \((at_3)\), due to a lower service time in the web tier components, the number of requests in the components’ queue at the business logic tier is used. The component state \(at_4\) is the state of a specific component which fetches requested news items from the back-end database, and is either active or failed. In practice, for example in the case of Java EE, this component can be implemented by an EJB which uses entity objects and Java Persistence API (JPA) to work with a database. \(at_5\) denotes the number of requests served in a time unit that is one second here.

For the sake of design simplicity, the data type and data quality attributes were combined into one attribute, \(at_1\). Table II lists different values that \(at_1\) can get during the simulation. Switching to lower values increases the response time and throughput, while it decreases the level of satisfaction for the best data type and quality goals.

Table II. Data Type/Quality Values for \(at_1\) in News Application

<table>
<thead>
<tr>
<th>Data Type/Quality</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>TNH</td>
<td>Text + Normal size Image + High resolution Video</td>
</tr>
<tr>
<td>TNL</td>
<td>Text + Normal size Image + Low resolution Video</td>
</tr>
<tr>
<td>TN</td>
<td>Text + Normal size Image</td>
</tr>
<tr>
<td>TS</td>
<td>Text + Small size Image</td>
</tr>
<tr>
<td>T</td>
<td>Text</td>
</tr>
</tbody>
</table>

Figure 5 depicts the goal hierarchy in the conducted experiments. Low-level goals are the ones related to measurable attributes. Maximum availability is translated to min Mean Time To Repair (MTTR), as stated by Candea et al. [7]. As the authors discussed in [7], reducing MTTR can be as effective as increasing Mean-Time-To-Failure (MTTF) in maximizing availability. As seen in this figure, higher level goals may share sub-goals. For example, end users want the service to be available, quick, and of high quality.

![Goal Hierarchy of the Experimental Model](image)

Figure 5. Goal Hierarchy of the Experimental Model (Arc means logical AND)

Six adaptation actions, shown in Table III, are used in the experiments. For these experiments, it is assumed that except \(ac_1\) (restart), all other actions are always applicable. The precondition for restart is a “failed” state in a component. In the initial model, preconditions for the other actions were considered to change only from one state to an adjacent state; for example, to change from TNL to only TN or...
TNH. However, for two reasons these preconditions were removed. The first reason is that when the server changes the service level, for instance from TNH to T, a single user may not even notice this change because of the service time and the thinking time for the next request. Therefore, a jump in the service level is not observed except probably in extremely high traffic, which would lead to a crash in the system. The second reason is that these preconditions increase the system’s adaptation time $T_{ad}$, thus taking several actions to go from one extreme service level to the other. In essence, there is a high probability that users would not see the service level switches, and therefore, changes can be made to the service levels to decrease the adaptation time.

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ac_1$</td>
<td>Restart a component</td>
</tr>
<tr>
<td>$ac_2$</td>
<td>Switch to TNH</td>
</tr>
<tr>
<td>$ac_3$</td>
<td>Switch to TNL</td>
</tr>
<tr>
<td>$ac_4$</td>
<td>Switch to TN</td>
</tr>
<tr>
<td>$ac_5$</td>
<td>Switch to TS</td>
</tr>
<tr>
<td>$ac_6$</td>
<td>Switch to T</td>
</tr>
</tbody>
</table>

Figure 6 depicts schematically relationships between entities. Beside the figure, the list of parameters $f_i(.)$, $p_i$ and $\tau_{ij}$ are available for each leaf goal. The value of $\tau_{ij}$ has the measurement unit of its corresponding $at_j$. This is the reason for the different values in the figure for $\tau_{ij}$. The summation of $p_i$ is set to 100. The “max throughput” goal will be activated if throughput has a high level while load is high enough. This setting is because low-level throughput value does not necessarily mean the system is denying the “max throughput” goal. Therefore, when the load is low and the throughput is lower than its $\tau_{ij}$ value, this goal does not need to be activated. Activation functions of these goals are implemented as a threshold logic unit in Matlab. Therefore, Figure 6 shows only the threshold values. In the goal model design, in order to activate the “max throughput” goal in the right situation, the threshold of $f_i(.)$ is set 1 to reflect a higher threshold for activating this goal. This means two attributes load and throughput both should stimulate this goal to be activated.

Two points are notable about the model designed for these experiments. First, although logical relationships exist between goals in the hierarchy presented in Figure 5, in experiments the relationships above leaf goals are not used by the action selection mechanism. We tracked the upper goals based on the activation of subgoals only to evaluate how well the model and mechanism are working. Decision-making is based on goals that vote for actions, which in this case are leaf goals. However, a designer may decide to link higher-level goals to actions, and then those goals will be activated based on statuses of their subgoals.

Second, all goals in Figure 5 except “best news data type” and “best news data quality” are soft goals. However expected quality factors in the service level agreement, which are normally presented by acceptable ranges of system attributes, can define hard goals at least at the leaf level of a goal hierarchy. The upper-level soft goals can be a part of decision-making as well, if those goals are connected to actions. In the evaluation, we treat goals as soft goals that might be activated to a certain level. The deviation function we calculate can be interpreted as a measure for adaptation on this basis. The closer to zero the value is, the more adaptable the system is.

Two types of preference structure (utility function) are used in the experiments, ordinal and cardinal. For comparing ordinal and cardinal types of preferences, we assumed that the order of $ac_i$ in both cases...
at1: News data type and quality in web server
at2: End-to-end response time
at3: User load (user requests)
at4: Component states in app server
at5: End-to-end throughput

ac1: Restart components in app server
ac2: Switch to TNH (in web server)
ac3: Switch to TNL (in web server)
ac4: Switch to TN (in web server)
ac5: Switch to TS (in web server)
ac6: Switch to T (in web server)

Figure 6. Composing GAAM for the news application

are identical. It means that the descending sorted list of cardinal preferences is the same as the ordinal
preferences. As stated in research question RQ2, one of the goals is to compare these utilities in the
simulated model. Table IV shows the ordinal preferences defined for this case study. In the ordinal
case, the Borda count voting method is used [54].

<table>
<thead>
<tr>
<th>Order/Goal</th>
<th>Min MTTR</th>
<th>Best Data Type</th>
<th>Best Quality</th>
<th>Min End2End RT</th>
<th>Max Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>ac1</td>
<td>ac2, ac3</td>
<td>ac2</td>
<td>ac6</td>
<td>ac6</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>ac4, ac5</td>
<td>ac3</td>
<td>ac5</td>
<td>ac5</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>-</td>
<td>ac5</td>
<td>ac4</td>
<td>ac4</td>
</tr>
<tr>
<td>1</td>
<td>-</td>
<td>-</td>
<td>ac6</td>
<td>ac3</td>
<td>ac3</td>
</tr>
<tr>
<td>0</td>
<td>ac2, ac3, ac4, ac5, ac6</td>
<td>ac1</td>
<td>ac1</td>
<td>ac2, ac1</td>
<td>ac2, ac1</td>
</tr>
</tbody>
</table>
Table V shows the values of preferences in the cardinal utility case. The order of actions for each goal is identical to the ordinal case. These values were tuned in several experiments, and this set represents one of the best settings with this order.

<table>
<thead>
<tr>
<th>Action/Goal</th>
<th>Min MTTR</th>
<th>Best Data Type</th>
<th>Best Quality</th>
<th>Min End2End RT</th>
<th>Max Throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ac_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ac_2$</td>
<td>0</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$ac_3$</td>
<td>0</td>
<td>3</td>
<td>1.75</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>$ac_4$</td>
<td>0</td>
<td>1.75</td>
<td>3</td>
<td>2.75</td>
<td>2.5</td>
</tr>
<tr>
<td>$ac_5$</td>
<td>0</td>
<td>1.75</td>
<td>1.25</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>$ac_6$</td>
<td>0</td>
<td>0.5</td>
<td>1</td>
<td>3.75</td>
<td>4</td>
</tr>
</tbody>
</table>

In the experiments with unequal priorities, several settings were evaluated for $p_i$ values. For these cases, the statistical results were not different. As will be mentioned in the future works section, a complete sensitivity analysis for these priorities is planned. In this article, $PV$ is the set with values of $\{10, 2, 2, 4, 1\}$; $p_1$ is set to 10 so that its value is more than the summation all the other priorities. This is to ensure that the failure state will always be handled first. In other words, self-healing is more important than the other goals. $p_2$ and $p_3$ are equal, that means the best quality and best data type goals are equally significant for users. $p_4$ is the priority value of achieving the best response time, and is set to be the second highest value in the $PV$ set.

4.4. Experiment Design

A factorial design was used for the experiments. Regarding research questions RQ1-3, four treatments were considered for the experiments: no adaptation (NoAdapt), goal-based adaptation with ordinal preferences (Ordinal), goal-based adaptation with cardinal preferences (Cardinal), and rule-based adaptation (Rule). In the NoAdapt treatment, except for the manual restart, there are no actions to change the news application. For the Rule treatment, 13 rules that directly bind attributes to actions control the adaptations of the news application. The rule-based treatment was chosen based on the fact that most practical solutions use a form of this approach, often called the policy-based approach (e.g. in [2]). Using this approach, the goal-based treatments, Ordinal and Cardinal, are compared with the Rule treatment in order to determine the effectiveness of each type.

Because workload characteristics impact the system’s quality and its evaluation function, load intensity is used as a blocking factor. The workload factor can accommodate three levels, medium, high, and very high, depending on the average $IATime$ of requests. All of the traffic patterns in the experiments start with a default workload (i.e. a default $IATime$). This then drops after 10 seconds to one of the above levels in order to simulate burst traffic.

In each experiment, there is an adaptation factor identifying treatments, and a load intensity blocking factor. For each load level and treatment, the experiment has three replications. The treatments are evaluated based on three different conditions: i) with requests generated from two probability distribution functions, Exponential and Weibull, ii) with equal and unequal priorities, and iii) with...
requests including a timeout value. Overall, four experiments were conducted under the following conditions:

- Exp1: exponential traffic, equal priorities, and no-timeout
- Exp2: Weibull traffic, equal priorities, and no-timeout
- Exp3: exponential traffic, unequal priorities, and no timeout
- Exp4: exponential traffic, unequal priorities, and with timeout

Because the timed-out requests are dropped by the application components in Exp4, the evaluation function does not result in a comparable value with that of the other three experiments. In fact, it should also take into account a penalty for dropped requests. Designating such a penalty value is not an easy and straightforward task. Although several attempts were made to incorporate the penalty, it was decided to consider a loss ratio (the percentage of dropped requests) along with the evaluation function.

4.5. Threats to Internal and External Validity

A set of experimental studies are used to validate the proposed model for action selection. Two important concerns of such validation are internal and external threats. This section discusses to what extent these concerns threat the validity of conducted experiments, particularly in terms of the classification provided by Shadish et al. [51].

Because the conducted experiments are controlled, replicated, and independent from each other, most of the internal validity threats such as testing, history, maturation, and instrumentation threats do not exist in this work. This means the relationships between dependent and independent variables are not affected by other factors and time. There are some concerns that may be attributed to internal validity threats, which are not believed to be critical, and can be addressed with reasonable effort. For instance, one point that affects the internal validity of the experiment, is the value of $T_{ad}$, the period of adaptation. A fixed value was used for this parameter, but we observed that the value can have a remarkable impact on the evaluation function. One possible solution would be to use a dynamic value, for example as suggested by Menasce et al. [36].

Generally, in the conducted experiments, there are no dependencies on subjects, treatments, measurement methods and the environment which cause external validity threats. However, there are some issues about generalizing the model to other domains. The GAAM and the proposed action selection mechanism were evaluated by a simulated model of a news web application. The experimental model is a generic model for multi-tier enterprise applications, and fundamental changes in the results are not expected for these applications. A remarkable point is that for this specific application we assumed there is no session state for users in the experimental model. Even though the system does not change the service level for the users using any type of service, restarting components can lead to loss of session information. This feature is not essential for a news application, but is required for some other enterprise applications. Therefore, persistency of this information needs to be added at least for the restarting action. Therefore, by considering stateful cases, the approach is generalizable in this domain. A notable point about goals is that not only satisfying goals, but also keeping them satisfied (protecting goals) is important. As mentioned before, this issue is not critical for this experimental model, and for many server-side applications. However, it should be taken into account that changing
the goals’ states rapidly can cause problems for end users. Currently, GAAM is a memoryless model, but for protecting goals, previous states should be kept persistent. Another point related to external validity is the influence of policy maker on quality of GAAM or the rule set. Because in the conducted experiments one single person was in charge of design this issue was not a concern. However in comparing deciding processes built by different people this could be a factor. In the experiment design this factor should be eliminated or be taken into account.

Another concern in the external validity of the evaluation is that the experiments are based on a simulated model. Although a well-defined queue-server model was used in the simulation and a remarkable time was spent on developing and testing the model, still there is a gap between the model and the real application. The authors conducted some experiments on real applications, not part of this article, to figure out the extent of differences in designating and building GAAM. So far, we have not noticed any major unpredicted challenge in this way, but more work needs to be done. The reader can find more information about applying GAAM to real applications in [45].

5. Obtained Results

The results are first checked using the normality and variance homogeneity tests. The variances are homogeneous, but the distribution is not normal based on the QQPlot and the Levene test. Even the data transformations did not lead to the passing of both tests. Therefore, a non-parametric one-way ANOVA method with blocking is used for statistical analysis. The Friedman test [52] performs such a test by running a Chi square on the ranked data. In this test, data in each block is initially ranked and then an analysis is performed on the ranked data. Table VI shows the results for the Chi square test and the GLM ANOVA F-test in Exp1-3. These findings indicate that the treatments are not similar in these three experiments. To scrutinize the results, it is essential to look at the Box plots and pairwise comparison of the treatments.

<table>
<thead>
<tr>
<th></th>
<th>Row mean score difference</th>
<th>GLM F-test</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Value</td>
<td>Probability</td>
</tr>
<tr>
<td>Exp1</td>
<td>23.0171</td>
<td>&lt;.0001</td>
</tr>
<tr>
<td>Exp2</td>
<td>18.7436</td>
<td>0.0003</td>
</tr>
<tr>
<td>Exp3</td>
<td>18.812</td>
<td>0.0003</td>
</tr>
</tbody>
</table>

Figure 7 depicts the Box plots for the evaluation functions in Exp1-3. In Exp1, the difference between the range and median of the goal-based treatments (Ordinal and Cardinal) with the other two treatments (Rule and No-Adapt) is obvious. However, in Exp2 and Exp3, the Rule treatment is closer to the goal-based treatments. It appears that there is no significant difference between the goal and rule-based treatments (addressing $RQ^3$).

In order to statistically compare these treatments, several Dunnett tests were run. In fact, in each run one treatment is the control treatment, which forms the basis for the comparison. Table VII shows the summary of these comparisons in which duplicate tests were removed. In all three experiments, Ordinal and Cardinal are among the best in terms of the evaluation function. In Exp1, Cardinal and
Figure 7. Evaluation function of treatment in Exp1-3 (1: NoAdapt, 2: Ordinal, 3: Cardinal, 4: Rule)

Table VII. Dunnett t-test for pairwise comparison of treatments (*** means significantly different)

<table>
<thead>
<tr>
<th>Treatment Comparison</th>
<th>Mean Differences</th>
<th>Simultaneous 95% Confidence Limits</th>
<th>Significant ($\alpha = 0.05%$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exp1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cardinal-noAdapt</td>
<td>4.7778</td>
<td>(2.9832, 6.5723)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-noAdapt</td>
<td>4.2222</td>
<td>(2.4277, 6.0168)</td>
<td>***</td>
</tr>
<tr>
<td>rule-noAdapt</td>
<td>2.1111</td>
<td>(-0.3143, 4.5365)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-ordinal</td>
<td>0.5556</td>
<td>(-1.239, 2.3501)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-rule</td>
<td>4.7778</td>
<td>(2.3524, 7.2032)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-rule</td>
<td>4.2222</td>
<td>(1.7968, 6.6476)</td>
<td>***</td>
</tr>
<tr>
<td>Exp2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cardinal-noAdapt</td>
<td>4.4444</td>
<td>(2.6291, 6.2598)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-noAdapt</td>
<td>4.5556</td>
<td>(2.7402, 6.3709)</td>
<td>***</td>
</tr>
<tr>
<td>rule-noAdapt</td>
<td>5.778</td>
<td>(2.879, 8.676)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-ordinal</td>
<td>-0.1111</td>
<td>(-1.9265, 1.7043)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-rule</td>
<td>0.333</td>
<td>(-2.565, 3.232)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-rule</td>
<td>0.333</td>
<td>(-2.565, 3.232)</td>
<td>***</td>
</tr>
<tr>
<td>Exp3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cardinal-noAdapt</td>
<td>4.5556</td>
<td>(2.7402, 6.3709)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-noAdapt</td>
<td>4.4444</td>
<td>(2.6291, 6.2598)</td>
<td>***</td>
</tr>
<tr>
<td>rule-noAdapt</td>
<td>6.333</td>
<td>(3.442, 9.225)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-ordinal</td>
<td>0.1111</td>
<td>(-1.7043, 1.9265)</td>
<td>***</td>
</tr>
<tr>
<td>cardinal-rule</td>
<td>-0.556</td>
<td>(-3.447, 2.336)</td>
<td>***</td>
</tr>
<tr>
<td>ordinal-rule</td>
<td>-0.444</td>
<td>(-3.336, 2.447)</td>
<td>***</td>
</tr>
</tbody>
</table>

Ordinal are significantly different from the Rule and NoAdapt treatments. Figure 7 reveals that the Cardinal and Ordinal treatments are separate and distinct from the NoAdapt treatment. However, the Rule treatment falls somewhere in between. Response to RQ1, based on these results, is that adaptation improves satisfaction-level of goals.

The Simultaneous Confidence Level (SCL) in Table VII indicates that even though the limits are completely skewed to the Rule side, these three treatments are not significantly different. Because the evaluation function of the Cardinal, Ordinal, and Rule treatments were so close in Exp2 and Exp3, the test was repeated with $\alpha = 0.01$. However, the differences are still not significant. Therefore, in response to RQ2 and RQ3, statistical results suggest no difference. Overall, it appears that if the treatments need to be sorted using the Box plots and the SCL values, Cardinal is slightly better than
the others, especially in Exp1 and Exp3. The next treatment is Ordinal (in Exp2 it is even better than Cardinal), and then Rule, and finally NoAdapt treatments.

![Box plots of Evaluation Function and Loss Ratio in Exp4](image)

Figure 8. Evaluation function and loss ratio in Exp4 (1: NoAdapt, 2: Ordinal, 3: Cardinal, 4: Rule)

The fourth experiment generated some interesting results, since there is a tradeoff between dropping requests and satisfying the goals. As depicted in Figure 8, the NoAdapt treatment outperforms the others in terms of the evaluation function, but at the cost of dropping much more requests (median 12%). In fact, regarding both Box plots, it appears that the Ordinal treatment outperforms the others, as it results in a relatively low loss ratio and a moderate evaluation function value. Therefore, Exp4 verifies the results of the previous experiments regarding to RQ1 to RQ3.

6. Related Work

Researchers in the self-adaptive software domain subscribe to the view that deciding is a vital process (among others pointed out by Oreizy et al. [39]), and its realization is still a challenge, as noted by Salehie et al. [46] and McKinley et al. [35]. Existing solutions for self-adaptive software, and more broadly for autonomic systems, have still not addressed all of the requirements of a desired deciding process (discussed before). Maes enumerates several specific requirements in [30]. Finding good enough actions, minimizing back and forth switching between actions that contribute to distinct goals, and never getting stuck in a loop or deadlock situation to satisfy an unattainable goal, are some of these requirements. However, no remarkable set of criteria and metrics exist for verifying that a solution complies with these requirements. Gjorven et al. discusses applicability of some quality factors for adaptation [14], but there are notable difficulties for evaluating most of the quality aspects of adaptation.

In decision-making approaches, it is common to build a structure to relate goals, attributes, and actions, such as discussed in multi-attribute utility theory [23]. In software engineering, goals are mostly defined and utilized in requirements specification, such as i* [56] and Soft-goal Interdependency Graph (SIG) [10]. SIG is an AND-OR graph which decomposes and relates goals using AND and OR relationships. However, i* models goals as a structure of connected nodes without
explicitly representing a logical relationship between the parent and offsprings. Both models were used in representing non-functional goals. For example, Non-Functional Requirements framework (NFR) benefits from SIG [10]. SIG decomposes non-functional goals and design solutions (called operationalizations). Design decisions stimulate the model from operationalization nodes, and the reasoning propagates the impacts though goals. By reviewing the goals’ satisfaction degree, designers evaluate their decisions.

The goal-driven solutions for self-adaptive software can be divided into two broad categories, which are not necessarily orthogonal, namely: development-time and run-time approaches. The idea of goal-driven models has been extensively used for decision-making in the development phase, especially for design decisions. In the self-adaptive software domain, Subramanian et al. adopted the NFR Framework to build a knowledge-based system for developing adaptable software architectures based on requirements specifications [55]. Brown et al. also employ the KAOS goal model to represent the adaptation semantics [6]. Among other works in this category, [27], [37] and [16] are also noteworthy.

In the run-time category, based on the literature review, few research efforts utilize a goal-based model in the decision-making process at run-time. For example, Kramer et al. discuss an architecture-based adaptation approach, which utilizes a goal management layer [25]. This layer generates a reactive plan to satisfy the goals. Salehie et al. have also worked on an adaptation approach based on quality goals, in order to trace and satisfy these goals at run-time [48]. In another work, Salehie et al. employed a weighted voting scheme for goal-based decision-making [49]. A notable point about a goal-driven deciding process is that it may not be possible or efficient to use automated planning, in this case, reactive planning. For example, Srivastava et al. [53] argue that planning is an appropriate option for self-healing, but is generally not effective for self-optimizing. Therefore, having a goal-based decision-making mechanism does not necessarily mean that a planning-based approach is employed.

The focused problem in this article is also similar to the Action Selection Problem (ASP) in autonomous agents [29] and robotics, especially in mobile robots [42]. Similarities are due to dynamism and uncertainty in the environment, as well as online decision-making. However, the problem in software is generally more complex than in robotics. Software systems, especially large-scale distributed applications, have much more attributes and effectors, and are often more complex than mobile robots. Some researchers believe that the complexity of software in comparison with most systems in mature engineering disciplines can be attributed to the nature of software systems: the notable difference is that their behaviors do not obey descriptive physical laws [8].

The focus of this article is on the run-time category. Due to the similarity of the focused problem with the ASP in robotics and autonomous agents, several ideas from these disciplines have been adopted and employed. In particular, the idea of behavior-based robotics [1] seems promising for this purpose. The majority of existing architectures for the adaptation manager in self-adaptive software takes benefit from the Sensor-Plan-Act (SPA) model, that is used extensively in building traditional robotic systems. In these systems, events are collected, analyzed, and fused to update the domain model (i.e. world model). The system then plans its strategy in the new situation. However, the idea of behavior-based robotics is to use distributed specialized task-achieving modules, called behaviors, and to apply command fusion instead of sensor fusion [1]. In this way, there is no need to develop, maintain and extend a coherent monolithic model of the adaptable software and its context. Gat argues that this is one of the main problems of the Sense-Plan-Act schema used in most traditional robots [13]. The successful experiences in building behavior-based robots motivated us to apply the idea to self-adaptive software. Behaviors in robotics are mapped and extended to goals in this domain.
A goal-driven action selection mechanism can be realized in two general competitive and cooperative forms. In the former, the goals are competing with each other in order to select the next action, whereas in the latter, the preferences of goals are combined or fused to determine what to do next. Arkin discusses variant forms of cooperative and competitive mechanisms in behavior-based robotics [1]. In the cooperative category, Arkin names superpositioning (vector addition) as the most straightforward method, if this is feasible. In the competitive methods, arbitration is a way to select one goal (winner-takes-all), e.g., based on predefined priorities. The subsumption architecture basically employs this method [5]. A less autocratic method has been utilized by Maes in an activation network [29]. Maes argues that the lack of explicit goals and goal-handling capability in autonomous agents lead to significant limitations in their operation [29]. Notably, when an agent does not have goals, every situation has to carry complete information for deciding about the next most appropriate action. Maes proposes an activation network for actions, in order to facilitate dynamic action selection based on stimulated goals or actions.

The Distributed Architecture for Mobile Navigation (DAMN) uses a more democratic way by adopting a voting mechanism [42]. In fact, this method, and generally the voting-based mechanisms, can arguably be placed into the competitive category by Arkin. The social choice methods and voting games are well-known in cooperative game theory, in order to combine decisions made by agents [11]. DAMN, introduced by Rosenblatt, realizes the first level of behaviors in the subsumption architecture [42]. DAMN uses a voting mechanism for command fusion, regarding the safety behaviors for the turn and speed of the mobile robot. The beauty of DAMN’s design is that the deliberative and reactive components of the architecture can operate at the same level, and it is scalable due to its lack of hierarchy.

A valid question is that what differences exist between GAAM and the Belief-Desire-Intention (BDI) model for intelligent agents [41]. Beliefs represent the system knowledge about the situation (self and context). Desires refer to the objectives and can be instantiated by goals, and intentions are what an agent decided to do and may include plans. The BDI model has some limitations that are discussed in [40]. The three elements of BDI would be roughly mapped to the three key entities of GAAM. The point is that GAAM does not suffer from some limitations of BDI; for instance in representing goals explicitly. Moreover, the relationships between entities in GAAM are not similar to BDI. Nevertheless, BDI and its extensions in multi-agent systems are well-established and built on a strong body of knowledge, that could be useful in designing self-adaptive software systems.

7. Conclusions

This article aims at providing a goal-based model and decision-making mechanism to address runtime action selection in self-adaptive software systems. For this purpose, this work deals with how to keep goals in the adaptation manager and involve them in decision-making at run-time. To trace goals, attributes are monitored continuously, and to change the adaptable software actions need to be selected by activated goals. The proposed goal-based model, GAAM, addresses the desired deciding process, discussed in the problem statement (Section 2), in the following ways:

- Explicit goals - Rule-based methods implicitly utilize the goals to adapt the system. In fact, developers indirectly consider the goals in defining and tuning the rule set. However, GAAM
and the proposed goal-based action selection method explicitly represents the goals, trace them at run-time, and incorporate them in the deciding process. This characteristic leads to better traceability for administrators, and to more trust for stakeholders due to revealing why adaptation changes occur.

- Multiple objective decision-making - The goal-based approach discussed in this article intrinsically supports multiple goals. However, handling these goals in a rule-based method is not an easy task. This can result in a large set of rules with precisely adjusted priorities to address all the desired goals, while in the proposed approach each goal is defined with its own utility for the system’s adaptation actions. These goals are later coordinated in the action selection mechanism.

- Flexibility - Modifying rules at run-time is not always safe, and the system’s behavior is not predictable. In the goal-based action selection approach, this issue can be handled easier comparing with common rule-based mechanisms. Goals, actions, and attributes can be added or removed from GAAM, although we cannot make the claim that the system behavior is immune from any kind of change. The easiest way of changing GAAM is altering goal preferences. As noted before, these preferences can be generated with a dynamic algorithm.

The proposed solution for action selection problem has been validated by a set of experiments on a simulated model. The model has been built using a queue-server network for a news web application, but it can be used for other enterprise applications as well. Three research questions (RQ1 to RQ3, in Section 4), have been set to be investigated in the empirical evaluation. These questions address effectiveness of adaptation mechanisms (RQ1), comparing ordinal and cardinal utility for modeling preferences (RQ2), and contrasting goal-based and rule-based adaptation mechanisms (RQ3) in the conducted experiments.

Obtained results suggest that adaptation impacts positively on the evaluation function formulated based on the satisfaction levels of goals in the experiments. Thus, regarding the RQ1, adaptation improves goal satisfaction during the application operation. Both goal-based mechanisms, using ordinal and cardinal utility functions, perform well in the experiments. The interesting point is that the ordinal utility function in the conducted experiments generated acceptable results, with respect to the effort spent to define and tune the preferences. We cannot conclude that only by considering these results the ordinal utility can always be as well as cardinal utility function. But at least regarding the more effort required to define and tune cardinal preferences, ordinal preferences are easier to elicit and might lead to results as good as the cardinal case. Therefore, in response to RQ2, although using ordinal and cardinal utilities do not show a statistical difference, ordinal utility can be the initial choice for defining preferences. Thus, the answer to this question is more a design guideline not a utility function selection, due to investigating the question in a single case study. A similar argument can be applied to comparing the goal-based and the rule-based methods for RQ3. While they are not significantly different in terms of statistical measures, formulating and refining rules, as well as resolving their conflicts are not always straight-forward. Hence, the less effort for designing and tuning a goal-based deciding process is a plus point in selecting an appropriate approach. Of course representing explicit goals and a more clear multi-objective approach toward decision-making are notable for the goal-based approach.
8. Future Work

In future, several modifications can be applied to the layers of GAAM and the action selection mechanism. The goal layer of GAAM can use a more complex structure. Goal hierarchies and their relations (e.g. logical and/or) can be utilized in tracing goals, if higher goals in the hierarchy are connected to actions. In this case high level goals may be activated and be involved in the deciding process. This can lead to a change in the action selection mechanism. The activation function can generate a dissatisfaction level. This level indicates how much a goal’s vote will affect the outcome of action selection mechanism. This might be presented as a kind of fuzzy inference as well.

Moreover, the relationships between actions in the action layer can be added to GAAM. For example, the precondition for some actions may be to first perform other actions. Therefore, if those secondary actions are desired by goals, there will be another source of stimulating actions. By considering these relationships, those secondary actions would be selected in the next adaptation rounds. This issue can be related to adding memory to GAAM, not only for goals, but also for actions. One idea is to define an energy level for goals and actions, as suggested by Maes [29].

The focused problem in this article aims at selecting a single action that is primitive or composite. However, the problem can be generalized to plan a reactive course of actions or contingency plan. Automated AI planning can be utilized, but it is not often a simple problem to solve, especially in the presence of multiple goals. Another possible way to extend this work is to combine the goal-based and rule-based methods in order to use both the reactive and deliberative decision-making approaches. This idea is inspired from Gat’s three layer architecture [13] and discussed in this context by Jeff Magee and Jeff Kramer [25].

Another direction for extending this work is that this article only considers Borda count and intensity voting for selecting adaptation actions. However, other voting schemes from the literature can also be investigated; for example the schemes discussed in [31]. For cardinal preferences, Nash voting [38] can be used, and for the ordinal case, approval voting [4] is one of the options. Also, two-phase voting can also be implemented to select actions in the first phase with different voting schemes, and to aggregate the results with a secondary voting scheme, such as majority voting.

Although GAAM and the proposed action selection mechanism do not have any assumption about the adaptable software, in evaluating them mission-critical systems and particularly multi-tier enterprise applications have been targeted. The proposed model needs to be evaluated with other systems to identify the challenges. We have already started some experiments on web-based applications and service-based communication systems. The results were promising and the design and tuning of GAAM did not have a remarkable difference comparing with the reported experience in this article.

REFERENCES