A Weighted Voting Mechanism for Action Selection Problem in Self-Adaptive Software

Mazeiar Salehie and Ladan Tahvildari
Software Technologies Applied Research Group
Dept. of Electrical and Computer Engineering
University of Waterloo,ON, Canada, N2L 3G1
{msalehie, ltahvild}@uwaterloo.ca

Abstract

Self-adaptive software is a closed-loop system which aims at adjusting itself in response to changes at runtime. Such a system is required to monitor domain events, detect significant changes, decide how to react, and act in order to execute the decisions. This paper focuses on the deciding process particularly for application-level adaptation actions. For this purpose, a weighted voting mechanism has been proposed which makes decisions based on a Goal-Action-Attribute Model (GAAM). The decision-making algorithm traverses GAAM, determines activated goals and feasible actions for voting, and ultimately selects an action as the social choice. The proposed mechanism and GAAM are evaluated within a simulated model of a news web site.

1. Problem Definition

The main motivation for research in self-adaptive software comes from increasing cost of the management complexity in software due to changing context and variable goals/requirements. The adaptivity aims at giving a degree of variability to software, and navigating it during applying *adaptation actions*. Generally, the *adaptation mechanism* is decomposed to monitoring, detecting (analyzing), deciding (planning), and acting (effecting) processes. Most of the existing solutions realize the adaptation processes in an external *adaptation engine*. This engine manages the adaptation for an *adaptable software*, which is implementing the application logic.

Among adaptation processes, deciding is a vital process, and its realization is still a challenge, as discussed by McKinley *et al.* [6]. Existing solutions for self-adaptive software have not addressed variant requirements for a desired deciding process [8]. The deciding process essentially needs to know about the adaptable software, objectives related to self-* properties and business goals, and articulated preferences by the system's stakeholders. These objectives can be decomposed to low-level objectives and consequently to goals determining the aspiration levels of system attributes. A set of actions is also required to satisfy the goals.

For satisfying the objectives and goals under different conditions of the attributes, it is required to select the most appropriate action(s). Without loss of generality, we can assume the set of goals and objective are one single set. The focused problem can be formally defined as: *Given a goal set $GL = \{gl_i, i = 1..m\}$, an adaptation action set $AC = \{ac_i, i = 1..n\}$ and an attribute set $AT = \{at_i, i = 1..p\}$, the problem is how to select the most appropriate action a_i to satisfy goals in different conditions of attributes.*

Central to our view, in this paper, is to propose a deciding process which is flexible/scalable, timely efficient, and multi-objective. The proposed process is based on a weighted voting mechanism.

2. Related Works

In decision-making approaches, it is convenient to build a structure to relate, goals, attributes and actions, as discussed in [4]. In software engineering, goals are mostly involved in requirements specifications. The Soft-goal Interdependency Graph (SIG) is one of the model used for this purpose in the Non-Functional Requirements framework (NFR) [2]. Subramanian *et al.* adopted the NFR Framework to build a knowledge-based system for developing adaptable software architectures based on the requirements [9]. However, to our best knowledge there is still no remarkable goal-driven approach in software engineering research which contains an explicit representation of goals involved in decision-making at runtime.

The focused problem in this paper is similar to Action Selection Problem (ASP) in autonomous agents [5] and behavior-based robotics [7]. The similarity is mainly because of dynamism and uncertainty of the environment, as well as online decision-making. However, the problem

First International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2007)
0-7695-2906-2/07 $25.00 © 2007 IEEE

in software domain is generally more complex due to not obeying physical rules; in addition of having more control variables and disturbances.

The idea of behavior-based robotics is to use distributed specialized task-achieving modules, called behaviors, and to apply a command selection/fusion mechanism instead of sensor fusion. By this mean, there is no need to develop and maintain a monolithic model for the adaptable software and its context. Distributed Architecture for Mobile Navigation (DAMN), introduced by Rosenblatt, realizes the behavior-based control using a voting mechanism. DAMN is used for command fusion regarding the safety behaviors for turn and speed of the mobile robot [7]. The beauty of DAMN design is that the deliberative and reactive components of the architecture can operate at the same level and also it is scalable due to lack of hierarchy.

On the other hand, Maes argues that the lack of explicit goals and goal-handling capability in autonomous agents lead to significant limitations in their operation [5]. Maes proposes an activation network consisting of several actions with their own preconditions, postconditions and activation levels. The actions are connected to each other and can be activated by environment states, goals as well as successor/predecessor actions. Although this solution has the advantage of distributedness and planning without pitfalls of traditional AI-planning, there are no clear guidelines for choosing network parameters.

3. Proposed Approach

This section sets out to define the goal-action-attribute model, and to elaborate the decision-making algorithm based on this model.

3.1. Goal-Action-Attribute Model

The main purpose of the Goal-Action-Attribute Model (GAAM) is representing the key entities of the deciding problem -goals, attributes and actions- and relating them in a well-defined structure. Goals $GL = \{gl_i, i = 1..m\}$ in GAAM have the following two parameters:

Activation Level: An ACtivation Level (ACL) vector is defined as $ACL = \{acl_i, i = 1..m\}$. An acl_i can be a threshold value or a rule specifying how the gl_i will be activated. Another option for activating the goals is to have a rate by which each goal participates in the decision-making.

Preference Vector: The stakeholders' preferences order the goals, in the form of $g_i \preceq g_j \preceq ... \preceq g_m$, by designating weights or priorities relating to each goal in the vector $PV = \{pv_i, i = 1..m\}$. We also consider the constraint $\sum_{i=1}^{m} pv_i = 100$ or 1.

The system attributes $AT = \{at_i, i = 1..n\}$ are divided into two categories of controllable and non-controllable attributes. Non-controllable attributes may not be directly manageable attributes, such as the response time, or disturbances like the user traffic.

Adaptation actions $AC = \{ac_i, i = 1..p\}$ (e.g. "restart") generally have preconditions and postconditions. Before considering an action for being selected, its preconditions must be checked. These preconditions can be denoted as $PC = \{pc_i, i = 1..p\}$, in which each PC_i can be a vector of preconditions for each action. The postconditions are basically the impacts of the actions on attributes and goals. In this paper for the GAAM, the postconditions are impacts on the goals which indirectly covers the consequences on the attributes.

Moreover, we define the following relationship matrices to relate the aforementioned entities:

Impact Matrix for relationship among m goals and p actions $IM = \{im_{ij}, i = 1..m, j = 1..p\}$.

Activation Matrix for relationship among m goals and n attributes $AM = \{am_{ij}, i = 1..m, j = 1..n\}$.

Aspiration Level Matrix for specifying aspiration levels of each goal in terms of n attributes $ASL = \{asl_{ij}, i = 1..m, j = 1..n\}$.

The values of im_{ij} and am_{ij} can be a boolean, a number or a fuzzy term. The GAAM specifies a graph $G = \{V, E\}$ in which vertices are $V = \{GL \bigcup AT \bigcup AC\}$ and edges are $E = \{IM \bigcup AM \bigcup PV\}$.

3.2. Decision-Making Algorithm

Figure 1 illustrates the flow of the proposed deciding process. Before making decision, it is essential to determine which goals have been activated and which actions are feasible to take effect. The activated goals ($\hat{GL}$) are voters and the feasible actions ($\hat{AC}$) are eligible candidates. As shown in Figure 1, events are detected by the aspiration values of each goal, asl_{ij}. In GAAM, low-level goals are used which are directly related to the attributes.

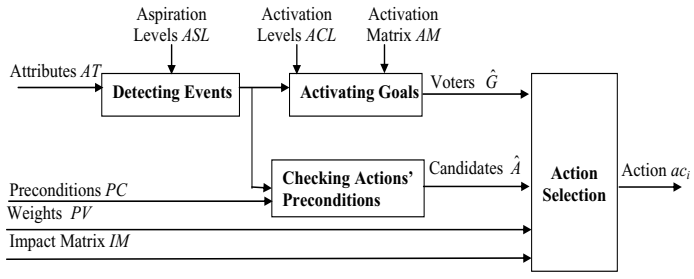

Figure 1. Action Selection Process

Generally, the decision-making algorithm traverses the GAAM graph from attributes to actions in order to select the most appropriate action. The algorithm is summarized in the following steps:

Step 1: Preparing attributes AT- Collect attributes at_i and prepare $at_i * am_{ji}, i = 1..n, j = 1..m$.

Step 2: Selecting voters- First, filter attribute values using $asl_{ij}, \forall gl_i \in GL, \forall at_j \in AT$. Then, aggregate all the processed at_j (by summation) and filter them using $acl_i, \forall g_i \in GL, i = 1..m$ to compile the activated goals in $\hat{GL} \subseteq GL$.

Step 3: Selecting candidates- Check each action's preconditions to compile the feasible set in $\hat{AC} \subseteq AC$ as the voting candidates.

Step 4: Updating preferences- Collect the preferences to update PV as weighting values for voters.

Step 5: Collecting votes- Collect votes of each goal as $gl_i * im_{ij}, i = 1..m, j = 1..p$.

Step 6: Selecting an action- Aggregate all values related to each action ac_i as $\sum_{i=1}^{n} gl_i * im_{ij}, i = 1..m, j = 1..p, \forall ac_j \in \hat{AC}$. Select an action, for instance by selecting the maximum value in the votes.

4. Proof of Concept

As a proof of concept, the action selection problem is used only for the application-level adaptation. One example of need to such an adaptation was news web sites after 9/11. For instance, according to a Los Angeles Times report, CNN.com surged to 162.4 million page views in 24 hours from a 14 million average. The website technical staff did an emergency redesign of the site just to strip out all the extraneous information and downgraded the service to give more significant news content.

The experimental model is a simplified model of a news web site implemented by Simevent toolbox in Simulink/Matlab. The model, illustrated in Figure 2, is a network of queues (or queueing network [3]) which is modeled as an open network with infinite population. The "front-end controller" interprets (preprocesses) requests and passes them to the proper components. In this example, there is only one path for requests. So, all the requests pass to the "news retrieval" component in order to retrieve the content (i.e. text, image or video). The "view-rendering" component prepares the final HTML page for the end-user.

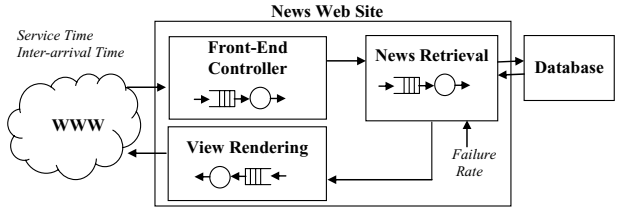

Figure 2. The Experimental Model

Each component has a priority queue based on service time of requests. The inputs are {*service time, failure duration, failure probability, restart signal, in traffic*} and outputs are {*average response time, number of requests served per second, component state, average load, out traffic*}.

For traffic generation *inter-arrival time* and *service time* have to be taken into account. Because there are different types of requests containing text, image and video, for each time interval there are different uniform probability functions for service times. In the experimental model, the service time for the "front-controller" does not depend on the type of requests. But, the randomly generated service time is used for the "news retrieval" and proportionally for the "view rendering" components. The failure rate is also generated by a uniform probability distribution function. The restart action is designed as a micro-reboot mechanism, discussed by Candea *et al.* [1]. Failure, restart and warmup processes of the component are modeled by a state-flow diagram in StateFlow toolbox of Matlab.

4.1. GAAM Specification

In this example, the system stakeholders are administrators and users which define their expectations as objectives of the system. Three objectives have been taken into account, namely: *self-optimizing*, *self-healing* and *max user satisfaction*. Self-optimizing is translated to "best performance" and then to "min end-to-end response time" and "max throughput". Self-healing is translated to "max availability" and then to "min Mean Time To Repair (MTTR)". As Candea *et al.* [1] discuss, reducing MTTR can be as effective as increasing Mean-Time-To-Failure (MTTF) in maximizing availability. Max user satisfaction is decomposed to "best news data type" and "best news data quality". The low-level goals in this goal model are the ones which are directly related to measurable attributes.

For the experimental model the controllable attributes are as follow: i) news quality: video resolution {High, Low}, and image size {Normal, Small}, and ii) news data type: {Video, Image, Text}. Non-controllable attributes are End2End Response time, User load (i.e. average number of waiting requests in a queue) and throughput (i.e. number of requests served per second). The component state is partially controllable (by restart action) and is partially a disturbance (by failure event). For the sake of simplicity of the design, we combine data type and quality in one attribute, at_1. If we denote Text as T, Normal and Small image mode as N and S, High and Low video resolutions as H and L, different feasible states for at_1 in the experiment is from the set $C = \{HNT, LNT, NT, ST, T\}$.

In this model, there are six actions containing restart and switching to each of the entities in set C (enumerated from 1-6). For each action, an action time t_{ac_i} between 200-600msec has been set. The voting module is disabled for t_{ac_i} for each action, in order to prevent the quick oscillation between actions. Preconditions of each action can be identified by the attributes, design constraints and resource limitations. For this example, only the states from set C have been considered.

We also need to designate ACL, PV, IM, AM and ASL entities. The value of asl_{ij} is the aspiration level of each goal regarding its corresponding at_i. For instance, for min end-to-end response time the asl_{ij} value for response time is 400msec. Preference vector PV is set to $\{30, 10, 15, 25, 20\}$ respectively for goals min MTTR, best news data type, best news data quality, min end-to-end response time and max throughput. The last goal, will not be activated unless the throughput is low and the load is high enough to pass the asl_{ij} values.

4.2. Obtained Results

Figures 3-a to 3-d show results in one of the experiments performed with the model in 5×10^5 steps with high load of user requests. Because the "news retrieval" component is the bottleneck of the system, the load and the service time has been shown only for it.

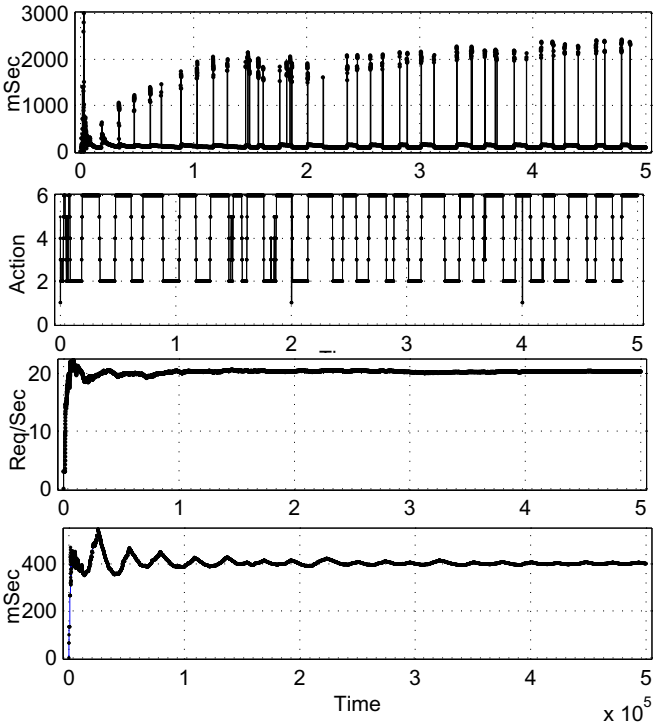

Figure 3. From top: a) Service Time of Requests, b) Selected Actions, c) Throughput (in Second), d) End-to-End Response Time

Figure 3-a depicts the inter-arrival time of web requests for the news web site. Small values of inter-arrival time are likely because of a high load in case of a high service time and remarkable number of requests in components' queues. In addition of the high load, failure can also lead to availability and performance degrade. In this specific experiment three failures occurred for the "news retrieval" component.

Figure 3-b depicts the actions selected by the deciding process. By changing the data type/quality, adaptation engine tries to satisfy both the min end-to-end response time subgoals of the max user satisfaction. Figure 3-c and 3-d show how the end-to-end response time and the throughput are changing after each action selection.

Although, there is no standard set of metrics to evaluate the quality of adaptation, we can discuss on how effective the results are. First, we calculate a utility function $\sum_{i=1}^{5} pv_i \times gl_i$, which at each step shows how much goals are satisfied. It turned out that in %36 of steps all goals are satisfied. We also calculate mean value and standard deviation for system's attributes. Type/Quality values show that on average the system gave NT (normal size image and text) to users with response time 403mSec and throughput 20 when average inter-arrival time was 200mSec.

5. Conclusion

In the proposed model, GAAM, the goal set plays a key role in navigating the deciding process. This matter results in having a better insight for action selection in comparison with the approaches based on state-action rules. The goals and actions can be realized more complex in the GAAM to address different aspects of the domain. The discussed decision-making algorithm is scalable, timely efficient, and not complex. This allows the system managers to extend the GAAM model of an adaptive system during its runtime by changing the goals, attributes or even actions.

The decision-making algorithm is sensitive to the weights (pv_i). We used the values set by stakeholders, but in fact they are required to be adjusted. Moreover, the algorithm uses a cardinal utility (im_{ij} values) for actions, while it can also utilize an ordinal utility. In this case a social choice procedure (e.g. Borda count) can be used.

References

[1] G. Candea, J. Cutler, and A. Fox. Improving availability with recursive microreboots: a soft-state system case study. *Performance Evaluation*, 56(1-4):213–248, 2004.

[2] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos. *Non-Functional Requirements in Software Engineering*. Springer, 2000.

[3] R. Jain. *The art of computer systems performance analysis*. John Wiley and Sons, 1991.

[4] R. L. Keeney and H. Raiffa. *Decisions With Multiple Objectives*. John Wiley and Sons, 1976.

[5] P. Maes. Situated agents can have goals. *Robotics and Autonomous Systems*, 6:49–70, 1990.

[6] P. K. McKinley, M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *IEEE Computer*, pages 56–64, 2004.

[7] J. Rosenblatt. DAMN: a distributed architecture for mobile navigation. *Journal Exp. Theory AI*, 9(2-3):339–360, 1997.

[8] M. Salehie and L. Tahvildari. Autonomic computing: emerging trends and open problems. In *Proc. of ICSE Workshop on Design and Evolution of Autonomic Application Software (DEAS)*, pages 82–88, 2005.

[9] N. Subramanian and L. Chung. Software architecture adaptability: An NFR approach. In *Proc. of Int. Workshop on Principles of Software Evolution*, pages 52–61, 2001.