Towards automation of performance architectural tactics application

R. Champagne, S. Gagné
Department of Software and IT Engineering
École de technologie supérieure (ÉTS)
Montréal, Canada
roger.champagne@etsmtl.ca, sebastien.gagne@gmail.com

Abstract—This paper describes the implementation of performance architectural tactics in ArchE, an expert system designed to help architects elaborate software architectures by automating certain tasks. Our goal was to explore what is involved in converting informally described tactics to a form that can be used by a tool to automate their relevant suggestion and application on software architectures. Adapting the tactics to the analysis tool and elaborating the rules that dictate their suggestion and application are the main challenges. Four performance tactics were successfully implemented in ArchE. Their use is illustrated through an example.

Software architecture; quality attribute scenarios; architectural tactics; performance; automation; ArchE.

I. INTRODUCTION

Software architecture (SA) is an active research field, as illustrated by the recent appearance of many methods, books, courses and specialized conferences on the topic. The work described in this paper falls under the general theme of software architecture.

Two core concepts in SA related work from the Software Engineering Institute (SEI) are quality attribute scenarios (QAS) and architectural tactics, or simply tactics. These concepts appear in many SA-related methods from the SEI. QAS are used to capture quality attribute requirements, while tactics are design decisions that affect some aspect of an architecture. QAS and tactics are both typically expressed informally in natural language.

The question our work seeks to answer is the following: how can we automate the application of architectural tactics to improve software architectures? In other terms, given a set of architectural tactics expressed informally in natural language, can we express them in a tool that automatically proposes only the relevant ones and applies them to an architecture? If so, how do we do this? What information about the requirements (QAS), the architecture, and the tactics themselves is required? This problem is partly solved for modifiability, as the Architecture Expert (ArchE) [3] implements support for some modifiability tactics. Our goal is to explore the automation of tactics for another quality attribute, namely performance. One of our assumptions is that automating tactics support for various quality attributes requires different approaches. We also aim to implement performance tactics support in ArchE.

The rest of the paper is structured as follows. In section II, we briefly summarize relevant software architecture concepts and how they are used in ArchE. In section III, we describe existing performance tactics from the literature and their mapping to ArchE. We also elaborate rules to correctly identify potentially relevant tactics in a given context. Our tactics implementation in ArchE is also briefly described in this section. Section IV describes how we validated our tactics implementation with a small yet nontrivial example. Finally, section V summarizes our main conclusions and identifies areas for future work.

II. ELABORATING AN ARCHITECTURE WITH ARCHE

A. Software architecture concepts

Quality attribute scenarios (QAS) are used to describe quality attribute (often called non-functional) requirements, such as performance, modifiability, availability, reliability and many others. QAS are typically expressed in natural language, and a well-formed QAS is composed of six parts: stimulus, stimulus source, artifact, environment, response, and response measure [1]. An example performance QAS is the following: a position sensor (source) transmits new position values every 130 ms (stimulus) when the system (artifact) is running under normal conditions (environment). Each new sensor value must be acquired, validated and stored in the system (response) before the next sensor update arrives, i.e. within 130 ms (response measure).

A software architecture tactic, or simply tactic, is defined as "a design decision that influences the control of a quality attribute response" [2]. Tactics are discussed in detail in section III.

A responsibility is defined as "an activity undertaken by the software being designed" [2]. It is important to note that in our work, a responsibility represents both a "function", and the architectural element that realizes the function. This is equivalent to having "regular" responsibilities on one hand, some architectural elements (e.g. processes) on the other, and a one-to-one mapping between the two.

B. Architecture Expert (ArchE)

ArchE is "a design assistant that helps architects explore architectural designs driven by quality attributes. ArchE shows the architect proposals for improving the current architecture and allows the architect to decide on the best alternative" [3]. We used the current version (v3) of ArchE in our work. Although ArchE has the ability to manipulate architectural elements such as modules and processes, we did not use this capability in our work. Instead, the "architecture" is expressed in terms of responsibilities, relations between these responsibilities, and relevant responsibility parameters. ArchE is able to perform two main types of operations: analysis, and tactics proposal and application.
Analysis in ArchE has the following meaning: given a software architecture (i.e. a set of responsibilities, the relations between them, and their parameters) and requirements on this architecture expressed in terms of QAS, perform some kind of evaluation on the architecture which results in some value corresponding to the parameter in the QAS response measures. For each scenario, establish if the response resulting from the evaluation satisfies the response measure specified in each QAS. In other terms, establish if the current solution satisfies the requirements. For example, for performance, two parameters are used by ArchE from each QAS: the stimulus value and the response measure. The stimulus value is necessarily the period at which the stimulus repeats (only periodic stimuli are considered), while the response measure is necessarily a deadline, which the "execution" of the scenario must meet. The architectural parameters that are used for the analysis are the relations between the responsibilies, and each responsibility's priority and execution time. Performance analysis in ArchE currently consists in using Rate-Monotonic Analysis (RMA) [4] to determine the worst case latency (WCL) resulting from the execution of responsibilities associated to each QAS. The worst case latency of each QAS is compared to the specified deadline (response measure). If the former is greater than the latter, the scenario is considered as not satisfied; otherwise, the scenario is considered satisfied.

To summarize, the following information must be supplied to ArchE as input for performance analysis: QAS stimulus (period), QAS response measure (deadline), relations between responsibilities, responsibility execution time, and responsibility priority. Figure 1 shows an example of a system for which there are four QAS and nine responsibilities. Each scenario stimulus triggers execution of a responsibility, which in turn either triggers execution of another responsibility, or is the last responsibility executed in a given scenario. Responsibilities shared by multiple scenarios are considered as independent instances of the same responsibility (no blocking considered, all instances of a responsibility have the same priority and execution time).

The second type of operation that ArchE can perform is to propose and apply tactics. In the case where at least one QAS is not satisfied, ArchE identifies and proposes relevant tactics to the user, who must decide which one (if any) to apply. The user might also be asked to supply new tactic-specific parameters so that ArchE can "apply" the selected tactic to the current architecture. Examples of such tactic parameters are given later.

The minimal capability for any quality attribute considered in ArchE is to be able to perform the analysis; support for tactics is not mandatory. ArchE v3 implements the analysis portion for both modifiability and performance, but implements tactics support only for modifiability.

III. PERFORMANCE TACTICS IN ARCHÉ

A. Performance tactics and their mapping to ArchE

Our initial set of candidate performance tactics included the eight tactics for performance described in [1]. We selected a subset of four of these tactics, based on their potential to map to ArchE in its current state, and also based on the fact that some tactics require more detailed information about either hardware or software elements than what we have available. The tactics we chose to implement are "Increase computation efficiency", "Manage event rate", "Increase available resources", and "Scheduling policy". We reemphasize our goal, which is to explore the process of converting informally described tactics to a form that is usable by a tool that partly automates their usage.

Mapping the four tactics selected above to ArchE requires some assumptions. The "Increase computation efficiency" and "Increase available resources" tactics both map to "Reduce responsibility execution time" in our implementation. For computation efficiency, the idea is that some responsibility is reimplemented in a more efficient manner, therefore reducing its execution time (e.g., a more efficient algorithm). This affects a single responsibility execution time. For resources increase, our assumption is that the user supplies a single value (percentage by which resources are increased), and this value translates to a proportional reduction in the execution time of all responsibilities. This is a naïve assumption, but we consider it provides sufficient results, given the information we have available to work with. The "Scheduling policy" tactic maps to "Increase responsibility priority" in our implementation. Since ArchE's current performance analysis imposes the scheduling policy (rate-monotonic scheduling), we considered it would be useful if we could identify responsibilities whose relatively low priority has a large negative impact on the overall latency of one or more scenarios which are not satisfied. Finally, the "Manage event rate" tactic maps to "Increase scenario period" in our work, as the stimulus event rate is in effect the inverse of the stimulus period.

B. Automating identification and application of tactics

In the previous section, we identified a set of candidate performance tactics as they are described in natural language elsewhere. We also mapped these informal descriptions to ArchE, based on some assumptions, within the constraints of ArchE's existing performance analysis capabilities. Our ultimate goal is to have ArchE only propose tactics if they have potential to improve the architecture, i.e. help it meet the requirements expressed in the QAS. In order for the

![Diagram](image)

Figure 1. Sample scenarios and responsibilities for a robot controller.
tactics to be suggested to users when appropriate, rules must be elaborated to determine the right conditions for each tactic to be proposed. The following subsections describe the rules associated to the application of each of the four tactics we selected for implementation in ArchE. The last two subsections are based on RMA, and are described rather briefly here, given the limited space. The design details of these rules are given in [5].

1) Increase available resources

This tactic is always proposed if at least one performance QAS is not satisfied. The rationale here is that this tactic can always potentially apply. The user must decide if this is the case or not.

2) Reduce responsibility execution time

This tactic, like the previous, is also proposed as soon as at least one performance QAS is not satisfied. The challenge here lies in identifying responsibilities whose execution time reduction would have the greatest impact on the overall latency of the scenarios that are not satisfied. To apply this tactic, we simply reduce the execution time of each responsibility one at a time by some percentage, and rerun the analysis to determine if each candidate architecture satisfies all performance QAS. Only those situations that improve the architecture are shown to the user. If no application of the tactic results in a newly satisfied scenario, the tactic application that reduces latency the most is shown to the user. The user then selects which responsibility he wishes to reduce the execution time for, and by how much, and the tactic is applied by ArchE, who changes the appropriate responsibility execution time.

3) Increase scenario period

Given a set of periodic events (scenario stimuli) which each initiate a sequence of actions or tasks (responsibilities in ArchE) which are all executed on the same processor, the period associated to each event has an impact on the worst case latency of potentially all the events. Increasing this period corresponds to a reduction of the frequency at which the periodic event arrives. If this tactic is considered by an architect, one of the challenges is to identify events whose period increase have a positive impact on the worst-case latency of the scenarios that are not satisfied. RMA can help to identify good candidates for period increase. There are two situations of interest for our purpose. The first is that the period of the scenario of interest has an impact on its own WCL. If a scenario is not satisfied, one rule consists in suggesting to the user that the period of that same scenario be increased. The second interesting category of scenarios, based on RMA, is all scenarios in so-called group $H$ ($H$ stands for "always has higher priority"). RMA groups scenarios in various categories when evaluating the WCL of a given scenario. All scenarios in group $H$ are "composed" of responsibilities that all have priorities higher than the responsibilities of the scenario of interest. Scenarios in group $H$ will always preempt execution of the scenario of interest if their stimulus arrives. Therefore, a second rule for this tactic is to point the user to scenarios in group $H$ as good candidates for period increase.

4) Increase responsibility priority

This tactic is similar to the previous one, in the sense that the goal is to identify responsibilities which have an important negative impact on latency. This time, we target low priority responsibilities, who are important contributors to latency because they will be preempted often by higher priority responsibilities. Two priority values are important for the application of this tactic: $P_{min}$, which is the lowest priority of all responsibilities mapped to the scenario of interest (one that is not satisfied), and $P_{sup}$, which is the lowest priority of a responsibility that is just above $P_{min}$, and is associated to a responsibility that is not part of the scenario of interest. The idea here is to change priorities in an incremental fashion. Intuitively, one could decide to change $P_{min}$ to the highest possible priority, but this is drastic, and likely not applicable very often. In short, this rule suggests to the user to change the priority of all responsibilities that are part of the scenario under study and have a priority value $P$ such that $P_{min} \leq P \leq P_{sup}$, to a value that is just above $P_{sup}$.

C. Implementation of performance tactics in ArchE

Implementing our performance tactics in ArchE consisted mainly in three things. First, we needed to change the way priorities are handled by ArchE. Priorities were not associated to responsibilities, so we added a priority parameter to the responsibility data structure. Secondly, implementing tactics in ArchE requires that questions be supplied. These questions are asked to users, who must answer them and in some cases supply extra parameters. We captured our performance tactics related questions and their associated parameters in ArchE’s configuration files. Last but not least, we had to program (in Java) the actual tactics implementation, within the constraints of ArchE’s APIs.

IV. VALIDATION OF THE PROPOSED APPROACH

The main goal of the validation described here is to ensure that the tactics work well together (i.e., no negative interactions). A second goal is to evaluate if this work is useful at all. The example used, a robot controller [6], was therefore selected based on the fact that it is small enough to be easily understood, and complex enough to give us a chance to experiment with the four tactics implemented. Some parameters were modified to accommodate our needs (i.e., some scenarios not satisfied). Architecturally speaking, the example consists of the responsibilities shown in Fig. 1. Table I shows the various properties of both the QAS (period, deadline) and the architecture (i.e., responsibilities, their mapping to scenarios, order of execution, execution times and priorities). With the given values, two scenarios (S2 and S3) are satisfied, and two are not (S1 and S4).

We describe one sequence of tactics proposal and application for this example. Our goal is to illustrate the kind of help a tool can bring in suggesting and applying tactics. We do not justify why values are increased or decreased to new specific values as justifying these decisions in the example is not our objective. Also, when a tool like ArchE suggests that a value be reduced or increased, it lacks information to determine if this is feasible or makes sense.
The tool suggests candidate improvements, but the user decides if the suggestion is applicable or not.

Our implementation of ArchE initially suggested 11 tactic instances (at least one instance of each of the four tactics). The user applies the "increase available resources" tactic, with an increase of 10%. All responsibility execution times are reduced; therefore WCL is reduced for all scenarios, but S1 and S4 are still not satisfied. ArchE then suggests increasing the period for S1 and S3. After consideration, the user chooses to increase the period of S3 from 140 ms to 160 ms. The WCL for S4 is reduced, but S4 is still not satisfied. ArchE then suggests increasing computation efficiency for R6 and R7. The user reduces the execution times of these two responsibilities to 10.5 ms and 7 ms, respectively. This reduces the WCL for all scenarios, but S1 and S4 are still not satisfied. ArchE then suggests increasing the priority of R2 from 3 to 5. The user applies this tactic. S1 is now satisfied. ArchE suggests increasing priority of R5 from 2 to 5. The user applies this tactic. S4 is now satisfied. Table II shows the various parameters associated to the example at this point, which corresponds to our "final" architecture, all QAS being satisfied.

Comparing tables I and II, the most significant difference is the WCL of S4. Intuitively, one might have suspected that its WCL would be high, based on the fact that its only responsibility has the lowest priority of all. However, we did not expect a WCL of over 10000 ms, especially considering an execution time of 0.5 ms. Also, the priority change for R2 (from 3 to 5) had a significant impact on its WCL, which reduced from 254 ms to 108 ms with this change alone. Even for this relatively small example, this improvement would not have been easy to find based on intuition only.

V. CONCLUSIONS AND FUTURE WORK

This paper describes the process involved in mapping a set of informally described performance architectural tactics to a form which can be used by a tool to automate certain tasks during software architecture elaboration. This mapping is constrained by the capabilities of the analysis tool. In our case, ArchE's performance analysis module was deployed some time ago, and we just added tactics support recently. One of our constraints was to use the existing analysis module as is, since we did not have the resources to change it. Ideally, tactics should be considered when the analysis method is designed or chosen, which would likely give more flexibility in terms of the kinds of tactics that can be included. The elaboration of the rules that determine when each tactic should be suggested (or not) is also constrained by the information available in the inputs, namely the requirements (QAS) and responsibilities, and by the nature of and results returned by the analysis method used. Although this paper describes an implementation of performance tactics in ArchE, the overall approach we used could be used elsewhere, as the core concepts are not specific to ArchE.

Future work includes modifying the way ArchE uses the external RMA tool (MAST), in order to gain some flexibility in the choice of performance tactics that can be implemented. In the longer term, we also wish to add support for new quality attributes in ArchE.

REFERENCES


