System Identification for Adaptive Software Systems: a Requirements Engineering Perspective

Vítor E. Silva Souza[1], Alexei Lapouchnian[2], and John Mylopoulos[1]

[1] Dep. of Information Eng. and Computer Science, University of Trento, Italy
`{vitorsouza,jm}@disi.unitn.it`
[2] Department of Computer Science, University of Toronto, Canada
`alexei@cs.toronto.edu`

Abstract. Control Theory and feedback control in particular have been steadily gaining momentum in software engineering for adaptive systems. Feedback controllers work by continuously measuring system outputs, comparing them with reference targets and adjusting control inputs if there is a mismatch. In Control Theory, quantifying the effects of control input on measured output is a process known as *system identification.* This process usually relies either on detailed and complex system models or on system observation. In this paper, we adopt a Requirements Engineering perspective and ideas from Qualitative Reasoning to propose a language and a systematic system identification method for adaptive software systems that can be applied at the requirements level, with the system not yet developed and its behavior not completely known.

1 Introduction

In Control Theory (e.g., [8]), *system identification* is the process of determining the equations that govern the dynamic behavior of a system. White box models describe a system from first principles, e.g., a model for a physical process that consists of Newton equations. In most cases, such models are overly complicated or even impossible to obtain due to the complex nature of many systems and processes (natural or artificial).

A much more common approach is therefore to start from partial knowledge of the behavior of the system and its external influences (inputs), and try to determine a mathematical relation between inputs and outputs without going into the details of what is actually happening inside the system. Two types of models are built using this approach:

1. Gray box models: although the peculiarities of system internals are not entirely known, a certain model based on both insight into the system and experimental data is constructed. This model, however, comes with a number of free parameters (control variables) which can be estimated using system identification. Thus, parameter estimation is an important activity here;
2. Black box models: no prior model is available here, so everything has to be constructed from scratch, through observation and experimentation. Most system identification algorithms are of this type.

We are interested in employing this control-theoretic framework for the design of adaptive software systems. In this paper, we adopt a Requirements Engineering (RE) perspective and assume that a goal-based requirements model is available for the system. At the requirements level, the system is not yet implemented and its behavior is not completely known. With this incomplete information, we are unable to fully identify how system configuration parameters affect outputs. Thus, quantitative approaches cannot be applied. Therefore, we base our approach on ideas from Qualitative Reasoning [10] and propose a systematic way of identifying target outputs and system configuration parameters as well as qualitative relations between these parameters and measured outputs, all using models. Our proposed technique is both qualitative and flexible in the sense that it can accommodate multiple levels of precision in specifications depending on available information.

According to our proposal, the output of system identification for a software system is an extended and parametrized requirements model. Each assignment of parameter values represents a different behavior (configuration) that the system might adapt to fulfill its requirements. Some of the parameters ("variation points") come directly from the model. For instance, for a meeting scheduling system that needs to collect timetables from all participants when a meeting is scheduled, there is a choice of collecting these directly from meeting participants (e.g., through email) or from a central repository of timetables. The behaviors are also determined by a set of control variables that influence system execution, its success rate, performance, or quality of service. For instance, the "Collect timetables" goal is influenced by a parameter "From how Many" (FhM) that determines from what percentage of the participants we need to collect timetables before the goal is deemed to have been fulfilled. If we need to collect from all, i.e., FhM = 100, then the success rate for the goal may be low and its completion time may be high, compared to the FhM = 80 setting.

The main objective of this paper is to propose a systematic process for conducting system identification. This process requires some new concepts, notably the notion of differential relations between control variables and indicators (monitored variables). We illustrate the proposed process with an example and validate the proposal with experiments on it.

The rest of the paper is structured as follows: section 2 summarizes research results used as the baseline in our proposal; section 3 presents a language for the modeling of qualitative information on the relation between system parameters and output; section 4 describes a systematic process for system identification using that language; section 5 discusses the validation of the proposal; section 6 compares it to related work; section 7 describes future research directions; and, finally, section 8 concludes the paper.

2 Research Baseline

The following sub-sections briefly present research results on top of which we build our proposal: Goal-Oriented RE (§2.1) and Qualitative Reasoning (§2.2).

2.1 Goal-Oriented Requirements Engineering (GORE)

Goal-oriented approaches to RE model requirements in terms of goals, softgoals, quality constraints (QCs) and domain assumptions (DAs) [9]. As running example for this paper, figure 1 shows a goal model for a Meeting Scheduler system.

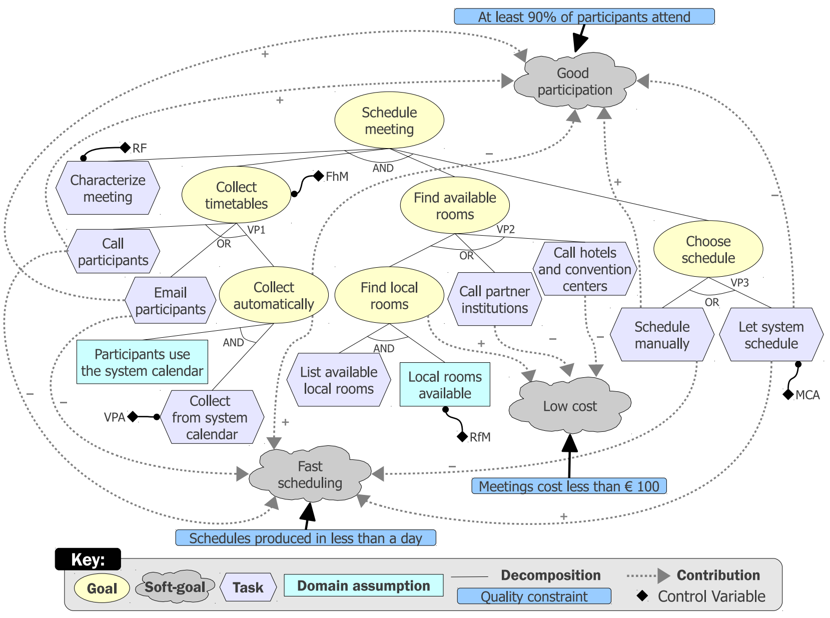

Fig. 1. Goal model for a Meeting Scheduler system.

In our example, the main goal of the system is to *Schedule meeting*. **Goals** can be decomposed using Boolean decompositions with obvious semantics. For instance, to *Schedule meeting*, one has to *Characterize meeting*, *Collect timetables*, *Find available rooms* and *Choose schedule*. On the other hand, to *Collect timetables*, it is enough either to *Email participants*, *Call participants* or to *Collect from system calendar*. Goals are decomposed until they reach a level of granularity where there are **tasks** an actor (human or system) can perform to fulfill them.

Softgoals are special types of goals that represent non-functional requirements (qualities) that do not have clear-cut satisfaction criteria. Goals and tasks contribute to the satisfaction of softgoals through positive or negative contribution links. Softgoals need to be refined into **quality constraints** (QCs) which offer concrete metrics for measuring how well the system is fulfilling a softgoal [9]. For example, *Good participation* is a desired quality for our system, receiving positive contribution from *Schedule manually* and negative from *Let system schedule*. A clear-cut satisfaction criteria for this softgoal is specified by the QC *At least 90% of participants attend*.

Goal models may also contain **domain assumptions** (DAs), which are statements that we assume to be true in order for the system to work. In the example, we assume there are *Local rooms available* in order to *Find local rooms.* If the assumption turns out to be false, its parent goal will not be satisfied.

Finally, figure 1 also illustrates **system parameters** that were identified for the meeting scheduler example. Monitored and controlled parameters have long been proposed as a way to implement reconciliation for adaptive systems at runtime [5]. However, in our proposal these are intentional parameters which are introduced much earlier in the development process, at the level of requirements. The example shows five *control variables* as black diamonds connected to other elements of the model.

OR-decompositions in goal models also represent intentional variability in the system. Choosing a different path at such *variation points* has been proposed as a way to configure systems [13] or to reconcile the behavior of adaptive systems at runtime in previous works such as [19]. In figure 1 we label the three existing OR-decompositions as `VP1`, `VP2` and `VP3` in order to be able to reference them in our language.

In section 3.1 we discuss in more depth the role of such parameters in our proposal.

2.2 Qualitative Reasoning

The key feature of qualitative reasoning (QR) methods (e.g., [10]) is that while frequently there is not enough information to construct quantitative models, qualitative models can cope with uncertain and incomplete knowledge about systems. They do not require assumptions beyond what is known. Most QR approaches can be seen as having two types of abstraction.

Domain abstraction abstracts the real domain values of variables into a finite number of ordered symbols that describe qualitative values, *landmarks*, that are behaviorally significant. Landmarks can be numeric or symbolic and can include the values such as 0 and $\pm\infty$. A qualitative variable value is either a landmark or an interval between adjacent landmarks. The finite, totally ordered set of all the possible qualitative values of a variable is called its *quantity space.*

Qualitative *functional abstraction*, which gives the ability to represent incompletely known functional relationships between quantities, complements domain abstraction in QR. E.g., signs (+,−,0) can be used to describe and reason about the direction of change in variables — one can state that there exists some monotonically increasing function relating two quantities, without elaborating further. Merging qualitative information frequently results in ambiguity, such as when combining positive and negative influences without knowing their magnitudes. Ranges of techniques and notations are available within QR, their applicability depending on the precision of the available information. E.g., one can reason about orders of magnitude, if they are known, possibly resolving said ambiguity.

3 Parameters and Qualitative Differential Relations

In this section, we further discuss system parameters and indicators of system output, as well as propose a language based on qualitative modeling [10] to augment our (goal-oriented) requirements model with information that captures the relationships among the these parameters in a qualitative way.

3.1 System Parameters and Indicators

As previously discussed, our proposal consists of a language and a systematic process to identify and model qualitative relations between *configuration parameters* and *measured outputs of the system*. Given our Requirements Engineering perspective, we propose to augment goal models of system requirements by recognizing *variation points* and *control variables* (collectively called *parameters*) and identifying *indicators* (of *system output*).

Variation points (VPs) are the OR-decompositions already present in the goal model. As we have mentioned in §2.1, selecting a different path at a VP at runtime is one way of reconfiguring the system in order to adapt to failures. Our proposal adds labels to VPs in the goal model (e.g., `VP1`, `VP2` and `VP3` in figure 1) in order to refer to them when modeling qualitative relations (see §3.3).

In this paper we introduce *control variables* (CVs), which represent another powerful mechanism for system (re)configuration. CVs are part of the system input. They can be applied to goals, tasks, and domain assumptions (DAs) and are used as abstractions over goal/domain model fragments. In particular, CVs are derived from families of related, but slightly different goal/task or DA alternatives, as in figure 2, where the goals *Collect timetables from 10% of participants*, *Collect timetables from 20% of participants*, etc. are shown as alternative ways to achieve the parent *Collect timetables* goal.

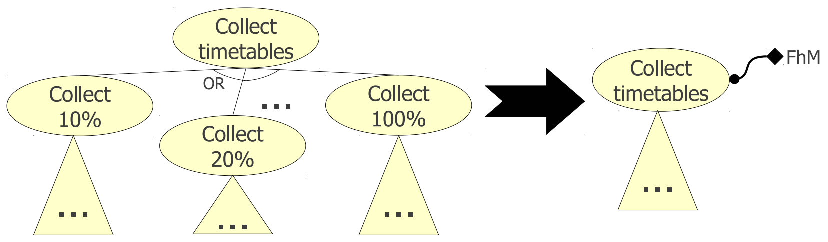

Fig. 2. Using a CV as an abstraction over families of subtrees.

Here, we identify variations that differ in some value (usually, but not necessarily numeric) and abstract that value as a parameter to be attached to the appropriate goal model element as a CV (e.g., the `FhM`, *From how Many* variable in figure 2). Figure 1 shows more examples of CVs, such as: `RF` (*required fields* when characterizing a meeting), `RfM` (number of *rooms for meeting* available — note that this CV applies to a DA), etc.

The benefits of having CVs include the ability to represent large number of model variations in a compact way as well as the ability to concisely analyze how changes in CV values affect the system's success rate and/or quality of service when, e.g., scheduling meetings. As any parameter in software design, a CV needs to be taken into consideration (i.e., propagated) when refining the goal model element that it applies to and later when designing and implementing the system. In this proposal, we are interested in analyzing the effect of values of CVs on system output and thus omit the details of CV refinement and implementation.

Finally, *indicators* are essential to control systems as these are monitored system output values that feedback loops need to compare to the output targets in order to calculate the control error and to determine how the system's control input needs to be adjusted. Indicators are similar to *gauge variables*, proposed by van Lamsweerde in [11].

Indicators need to be measurable quantities. In goal models, quality constraints (QCs) as well as the success rates for hard goals and tasks can be used as indicators. Since the number of potential indicators is large, we need to select as indicators the important values that the adaptive system should strive to achieve. *Awareness Requirements* (*AwReqs*) [18] are requirements that talk about the success or failure of other requirements, e.g., "*Find available rooms* should never fail" or "*Schedules produced in less than a day* should succeed 75% of the time". *AwReqs* are formalized and come with a monitoring infrastructure. They can be attached to QCs, hard goals, etc. (i.e., potential indicators) and capture the *reference input* of the system as well as specify the target success rates or other requirements about them. In our system identification approach (§4), we use *AwReqs* as the indicators in goal models. In the next sub-sections, however, we use *cost* and *speed* to refer to the QCs attached to softgoals *Low cost* and *Fast scheduling*, respectively.

Given the above definitions for system parameters and indicators and taking the *Find local room* goal of figure 1 as an example, we would like to model information such as: "upon increasing the value of `RfM`, the success rate of *Find local room* also increases" and "at `VP2`, when choosing *Call hotels and convention centers* over *Call partner institutions*, your cost will increase". This kind of information is very important for a feedback controller in its task of deciding how to adapt the system to fulfill its requirements.

In the remainder of the section we describe the qualitative approach for capturing and analyzing this information. Our approach does not differentiate between *control variables* and *variation points* and, thus, we hereafter refer to them generally as ***system parameters*** or simply ***parameters***.

3.2 Numeric Parameters

Numeric parameters, such as *Rooms for Meetings* (`RfM`), *From how Many* (`FhM`) and *Maximum Conflicts Allowed* (`MCA`) (see figure 1), can assume any integer or real value at runtime. There could be, however, some domain-related constraints, e.g., `RfM` can obviously assume only positive integer values, `FhM` ranges between 0% and 100%, etc.

Changing the value of a numeric parameter affects many aspects of system performance, which, as explained in the previous sub-section, are measured through *indicators*. Taking the parameter `RfM` as an example, and assuming the success rate of *Find local room* is affected by changes in `RfM`, we could define this indicator as a function of the parameter (clearly a simplification):

$$success\ rate\ of\ Find\ local\ room = f(RfM) \tag{1}$$

We could then say how changes in `RfM` affect the success rate of the goal by declaring if the derivative of f is positive or negative. Using Leibniz's notation:

$$\frac{\Delta\langle success\ rate\ of\ Find\ local\ room\rangle}{\Delta RfM} > 0 \tag{2}$$

Relation 2 tells us that if we increase the value of `RfM`, the success rate of *Find local room* also increases. Of course, the analogous decrease-decrease relation is also inferred. The $\Delta y/\Delta x$ notation is used instead of dy/dx because `RfM`, as previously mentioned, assumes only discrete values. Furthermore, in practice we use a simplified linearized notation to improve writability:

$$\Delta\left(\langle success\ rate\ of\ Find\ local\ room\rangle/RfM\right) > 0 \tag{3}$$

Suppose there is a limit to which this relation holds: after a given number, adding more rooms will not help with the success rate of *Find local room*. For this case, we use the concept of *landmark values* (see §2.2) and specify an interval in which the relation between the parameter and the indicator holds. Since we are dealing with qualitative information, we might not know exactly how many rooms are enough, so we define a landmark value called `enoughRooms`: $\Delta\left(\langle success\ rate\ of\ Find\ local\ room\rangle/RfM\right)[0, enoughRooms] > 0$. Although specifying this interval intuitively tells us that adding extra rooms after there are already enough of them available does not change the success rate of the goal, one could formalize this information, making it explicit: $\Delta\left(\langle success\ rate\ of\ Find\ local\ room\rangle/RfM\right)[enoughRooms, \infty] = 0$.

This gives us the general form for differential relations in our proposal, shown in (4), where Δ can be replaced with d in case of a continuous parameter, the interval $[a, b]$ is optional, with default value $[-\infty, \infty]$, $\langle op\rangle$ should be substituted by a comparison operator ($>$, $\geq$, $<$, $\leq$, $=$ or $\neq$) and C is any constant, not just zero as in previous examples.

$$\Delta\left(indicator/parameter\right)[a, b]\ \ \langle op\rangle\ C \tag{4}$$

Non-zero values for C are useful for expressing different rates of change. When facing a decision on how to improve an indicator I, given the information $\Delta(I/P_1) > 0$ and $\Delta(I/P_2) > 0$ the controller will arbitrarily choose to either increase P_1 or P_2; on the other hand, $\Delta(I/P_1) > 2$ and $\Delta(I/P_2) > 7$ could help it choose P_2 in case I needs to be increased by a larger factor.

If we replace the constant C by a function $g(parameter)$, we will be able to represent nonlinear relations between indicators and parameters, for instance, $\Delta\,(cost/RfM) = 2 \times RfM$ (cost increases by the square of the increase of `RfM`). However, linear approximations greatly simplify the kind of modeling we are proposing and are enough for our objectives. Moreover, it is very hard to obtain such precise qualitative values before the system is in operation.

3.3 Enumerated Parameters

In addition to numeric parameters, parameters that constrain their possible values to specific enumerated sets are also possible. *Variation points* are clear examples of this type of parameter, as their possible values are constrained to the set of paths in the OR-decomposition. *Control variables*, however, can also be of enumerated type (in effect, as discussed in section 3.1, *control variables* are abstractions over families of goal models in an OR-decomposition).

Figure 1 shows five enumerated parameters elicited for the meeting scheduler, two enumerated *control variables* and three *variation points*:

- *Required fields* (`RF`) in the task *Characterize meeting* can assume the values: *participants list only*, *short description required* or *full description required*;
- *View private appointments* (`VPA`) in the task *Collect from system calendar* can be either *yes* or *no.*
- At *Collect timetables*, `VP1` can assume values *Email participants*, *Call participants* or *Collect automatically*;
- At *Find available rooms*, `VP2` can assume values *Find local rooms*, *Call partner institution* or *Call hotels and convention centers*;
- At *Choose schedule*, `VP3` can assume values *Schedule manually* or *Let system schedule.*

Unlike numeric parameters, the meaning of "increase" and "decrease" is not defined for enumerated types. However, we use a similar syntax to specify how changing from one value (α) to another (β) affects a system indicator:

$$\Delta\,(indicator/parameter)\,\{\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n\}\ \langle op \rangle\ C \qquad (5)$$

By performing pair-wise comparisons of enumerated values, stakeholders can specify how changes in an enumerated parameter affect the system. For example, the relations below show how changes in `VP2` affect, respectively, the indicators *cost* and *speed* (both increase if you do the changes listed between curly brackets).

$$\Delta\,(cost/VP2)\,\{local \rightarrow partner, local \rightarrow hotel, partner \rightarrow hotel\} > 0 \qquad (6)$$
$$\Delta\,(speed/VP2)\,\{partner \rightarrow local, hotel \rightarrow local, partner \rightarrow hotel\} > 0 \qquad (7)$$

Often, however, an order among enumerated values w.r.t. different indicators can be established. For instance, analyzing the pair-wise comparisons shown in relations 6 and 7, we conclude that w.r.t. *cost*, *local* $\preceq$ *partner* $\preceq$ *hotel*, while for *speed partner* $\preceq$ *hotel* $\preceq$ *local*. Depending on the size of the set of values for an enumerated parameter, listing all pair-wise comparisons using the syntax specified in (5) may be tedious and verbose. If it is possible to specify a total order for the set, doing so and using the general syntax presented for numeric parameters in equation (4) can simplify elicitation and modeling.

3.4 Extrapolations

Differential relations always involve one indicator, but may involve more than one parameter. For example, "increasing" `VP1` and `VP3` (considering the order of the alternatives in ***variation points*** to be based on their position in the model, ascending left-to-right) contributes positively to indicator I_{FS} = *Fast scheduling* both separately — $\Delta\left(I_{FS}/VP1\right) > 0$ and $\Delta\left(I_{FS}/VP3\right) > 0$ — and in combination — $\Delta\left(I_{FS}/\left\{VP1, VP3\right\}\right) > 0$.

When we are not given any relation that differentially relate two parameters P_1 and P_2 to a single indicator I, we may still be able to extrapolate such a relation on the basis of simple linearity assumptions. E.g., if we know that $\Delta(I/P_1) > 0$ and $\Delta(I/P_2) > 0$, it would be reasonable to extrapolate the relation $\Delta(I/\left\{P_1, P_2\right\}) > 0$. More generally, our extrapolation rule assumes that homogeneous impact is additive (figure 3). Note that in cases where P_1 and P_2 have opposite effects on I, nothing can be extrapolated because of the qualitative nature of our relations.

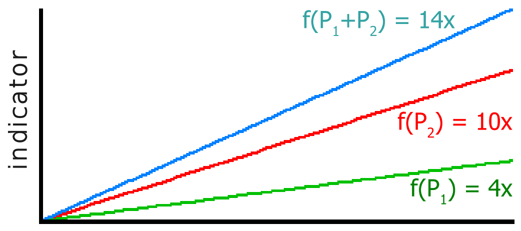

Fig. 3. Combining the effects of different CVs on the same indicator.

Generalizing, given a set of parameters $\{P_1, P_2, \ldots, P_n\}$, if $\forall i \in \{1, \ldots, n\}$, $\Delta\left(I/P_i\right)\left[a_i, b_i\right]\ \langle op\rangle\ C_i$, our extrapolation rule has as follows:

$$\Delta\left(I/\left\{P_1, P_2, \ldots, P_n\right\}\right) \bigcap_{i=0}^{n} \left[a_i, b_i\right]\ \langle op\rangle\ \sum_{i=0}^{n} C_i \tag{8}$$

If it is known that two parameters cannot be assumed to have such a combined effect, this should be explicitly stated, e.g., $\Delta\left(I/\left\{P_1, P_2\right\}\right) < 0$.

From differential calculus we extrapolate on the concept of the second derivative. If $y = f(x)$, we can say that y grows linearly if $f'(x) > 0$ and $f''(x) = 0$ (it "has constant speed"). However, if we have $f''(x) > 0$, then y's rate of growth also increases with the value of x (it "accelerates"). Qualitative information on second derivatives can be modeled in our language using the following notation: $\Delta^2(I/P)[a,b]\ \langle op\rangle\ C$. Thus, if we say that $\Delta^2(I/P_1) > 0$ and $\Delta^2(I/P_2) = 0$, the controller may conclude that P_1 is probably a better choice than P_2 for large values. Other concepts, such as *inflection* and *saddle points*, *maxima* and *minima*, etc. could also be borrowed, although we believe that knowing information on such points in a $I = f(P)$ relation without knowing the exact function $f(P)$ is very unlikely.

4 System Identification Process

In this section, we describe a systematic process for system identification. **Process Input:** a requirements model G (such as the one in figure 1). **Process Output:** a parametrized specification of the system behavior $S = \{G, I, P, R(I,P)\}$, where G is the goal model, I is the set of indicators identified by *AwReqs* in the model, P is the set of parameters, and $R(I,P)$ is the set of relations between indicators and parameters. At runtime, a feedback-loop controller receives S as input in order to adapt the system pro-actively or in case of failures.

The following are the steps of the process. They can be applied iteratively, gradually enriching the model with each iteration.

Step 1. Identify indicators: Introduce *AwReqs* into the goal model G specifying target success rates for QCs, hard goals or tasks. **Output:** the set of indicators I.

Step 2. Identify parameters: Identify possible variations in the goal model affecting the indicators, which, therefore, can be manipulated to adjust the performance of the system. These are captured by *control variables* and *variation points* (see §3.1). **Output:** the set of parameters P.

Step 3. Identify differential relations: For each indicator from the set I the requirements engineer asks: which parameters from P does this indicator depend on? Alternatively, iterate through set P and ask, for each parameter, which indicator in I is affected by it. Either way, one should end up with a many-to-many association between the sets. There are heuristics that help in answering these questions:

Heuristic 1: if provided, softgoal contribution links capture these dependencies for *variation points*. E.g., in figure 1, the choices in `VP1` contribute to the softgoal *Fast scheduling* and thus `VP1` affects the success rate of *Schedules produced in less than a day*, a QC derived from that softgoal. Any *AwReq*-derived indicator involving that QC is therefore also affected.

Heuristic 2: another heuristic for deriving *potential* parameter-indicator relations is to link indicators to parameters that appear in the subtrees of the nodes the indicators are associated with. The rationale for this is the fact that parameters in a subtree rooted at some goal `G`, which models how `G` is achieved, change the subtree, thus potentially affecting the indicators associated with the goal. E.g., the parameter RfM is below the goal *Find available rooms* in the tree and thus can be (and actually *is*) affecting its success rate, an indicator.

Heuristic 3: yet another way to identify potential parameter-indicator relations is to look at the non-functional concerns that these parameters/indicators address and to match the ones with the same concern. [18] describes how NFRs such as robustness, criticality, etc. lead to the introduction of *AwReqs* into goal models. The already-mentioned softgoal contributions explicitly link *variation points* with NFRs. Similar analysis should be done for *control variables*.

The modeling of the parameter-indicator relations is done using the language of section 3. **Output:** $R(I, P)$, the initial set of relations between indicators and parameters.

Step 4. Refine relations: The initial set of parameter-indicator relations produced in Step 3 should be refined by comparing and combining those that refer to the same indicator. When comparing two relations, say $\Delta(I_1/RfM) > 0$ and $\Delta(I_1/VP2) > 0$ (where I_1 might represent $\langle$*success rate of Find local room*$\rangle$), the modeler can investigate whether either of these adaptation strategies is better than the other and by how much. This may result in the model being refined into, e.g., $\Delta(I_1/RfM) > \Delta(I_1/VP2)$, which would help the controller facing the choice between these alternatives. The analysis of whether selecting an alternative makes the value of an indicator match its reference input is to be addressed in future work.

Combining relations also refers to what has been discussed in section 3.4: if a positive change in both parameters results in a positive change in the indicator, should we expect the default behavior in which $\Delta(I_1/\{RfM, VP2\}) > 0$ or should we explicitly specify that this is not the case? Such questions should be asked for any set of relations that refer to the same indicator.

Note that when combining relations to analyze alternatives, care must be taken to only look at the parameters/indicators relevant in the current system configuration. E.g., in figure 1, the parameter *View Private Appointments* (VPA) cannot affect any indicator if the value of VP1 is not *Collect automatically.* **Output:** $R(I, P)$, the updated set of relations between indicators and parameters.

5 Validation

To validate our proposal, we applied the system identification process described in section 4 to the meeting scheduler example presented throughout this paper, identifying 9 indicators (in the form of *AwReqs*), 8 system parameters (5 *control variables* and 3 *variation points* as shown in figure 1) and a total of 24 differential relations among the identified indicators and parameters.

For instance, one of the identified indicators refers to the goal *Find available rooms* as a critical requirement that should never fail, which is modeled in *AwReq* AR5: `NeverFail(G-FindAvailRooms)`. During parameter identification, *Rooms for Meeting* (RfM) and VP2 were identified, along with other parameters that are not relevant to AR5. In the next phase, two relations were identified: $\Delta(AR5/RfM) > 0$ (increasing the number of local rooms helps),

$\Delta\,(AR5/VP2) > 0$ (changing from *local* $\rightarrow$ *partner* $\rightarrow$ *hotel* helps). During refinement, analyzing RfM and VP2 in combination provided $\Delta\,(AR5/\{RfM, VP2\}) = \Delta\,(AR5/VP2)$ (increasing the number of local rooms and then not using them does not make sense) and $\Delta\,(AR5/RfM) = \Delta\,(AR5/VP2)$ (changing RfM or VP2 is equally effective).

Then, we developed a simulation that reads the above system information as well as events reporting *AwReq* failures (which could be provided by the framework we have presented in [18]) in order to identify possible adaptivity actions that could be taken by the controller during reconciliation. For example, when an event representing the failure of AR5 is received during the simulation, the program replies with the choices of parameter changes that have positive effect on AR5 based on the above qualitative relations:

```
* AwReq AR5 has failed! To reconcile, the controller could:
  - Current value of VP2 = local. Change it to one of: [partner, hotel]
  - Current value of RfM = 3. Increase it.
  - Note: VP2 and RfM should not be changed in combination.
```

With the information given by the differential relations, the program was able to identify available alternatives to adapt the system in case of failure. More sophisticated algorithms to analyze all the possibilities and select the best course of action (considering also the effect on NFRs, for example) are in our future plans for developing a complete framework for system adaptivity based on feedback loops. We are also currently working on a larger controlled experiment, conducting system identification on the London Ambulance System [6].

6 Related Work

There is growing interest in Control Theory-based approaches for adaptive systems and many of the proposed approaches include some form of system identification stage, in which the adaptive capabilities of the system are elicited and modeled. In [7], modeling is done by representing *system* and *environment actions* as well as *fluents* that express properties of the environment. In GAAM [17], measurable/quantifiable properties of the system are modeled as *attributes*, a *preference matrix* specifies the order of preference of *adaptation actions* towards goals (similarly to what we proposed in section 3.3) and an *aspiration level matrix* determines the desired levels of *attributes* of each *goal*. Our work differs from these by providing qualitative information on the relation between *system parameters* and run-time *indicators*.

In [14], Letier & van Lamsweerde augment KAOS with a probabilistic layer in order to allow for the specification of partial degrees of goal satisfaction, thus quantifying the impact of alternative designs in high-level system goals. In the approach, domain-specific *quality variables* (QVs) associated with goals are modeled and *objective functions* (OFs) define domain-specific, goal-related quantities to be maximized or minimized. Proposed heuristics for identifying QVs and OFs could be useful in the elicitation of *control variables* in our approach.

However, unlike our work, their models do not contain a clear relation between these variables and *indicators* measured in the target system.

Approaches such as *i** [16], the work by Elahi & Yu [4] and other proposals on design-time trade-off analysis can be adapted to provide information for run-time adaptivity (i.e., removing the need for stakeholder intervention in the analysis). For instance, contribution links in *i** can provide qualitative relations between *variation points* and indicators, although they lack the means of differentiating between links with the same label (e.g., see *Call participants* and *Email participants* in figure 1). GRL [1] could be used for this purpose, if cardinal contribution values (1, 2, ...) were changed to ordinal ones (1^{st}, 2^{nd}, ...), thus providing a graphical representation of enumerated value orders (§3.3). Our proposal provides such run-time trade-off information with a syntax that is more concise (*control parameters* abstract what would have to be represented as large goal sub-trees), uniform (can relate any system parameter to indicators) and flexible (the precision of the specification depends on the available information).

The proposal by Brake et al. [2] automates the discovery of software tuning parameters at the code level using reverse engineering techniques. A taxonomy of parameters and patterns to aid in their automatic identification provides some sort of qualitative relation among parameters, which may be "tunable" or just observed. While their work targets existing and legacy software, our proposal takes a Requirements Engineering perspective and, thus, can refer to higher level parameters, such as the success rate of a functional requirement or a quality constraint imposed over a non-functional one.

Finally, our proposal clearly differs from quantitative approaches (e.g., [1, 3, 11, 15]) in that we are using qualitative information, based on the premise that quantitative estimates at requirements time are usually unreliable [4] (assuming a domain with high uncertainty or incomplete knowledge of the behavior of the system-to-be). Our approach allows the modeler to start with minimum information available and add more as further details about the system become available (either by elicitation or through run-time analysis once the system is executing).

7 Discussion and Future Work

In this paper so far, we have overlooked an important modeling dimension, contextual variability. In this section, we sketch how it can be taken into consideration in the system identification process of section 4. We then discuss other research directions that we plan to pursue in the future.

Properties of the environment can affect the requirements for and the operation of a system, but, unlike the parameters we have discussed previously (CVs and VPs), context parameters cannot be directly manipulated, only monitored. *Contexts* are abstractions of such properties [12]. For instance, the type of a meeting can be viewed as a context for the meeting scheduling system, as can be the importance of a meeting organizer within the company. From the point of view of Control Theory, context most closely corresponds to a *disturbance* *input* that cannot be manipulated, but influences the output and thus must be accounted for. Contexts are organized using (possibly many) inheritance hierarchies that refine general contexts (e.g., *Regular meeting*) into more specific ones (*Mandatory meeting* or *Information session*) with descendants inheriting the properties of their ancestors. Each hierarchy structures contexts along a context dimension — some variable aspect of the domain (e.g., meeting importance) — with leaf-level elements directly monitorable. Multiple inheritance is supported.

In [12], (soft)goals and contribution links are identified as context-dependent goal model elements. *Contextual annotations* capture the effects of contexts on these elements and thus on software requirements by stating in which contexts the elements are visible. Unless explicitly overridden, the effects of ancestor contexts are inherited by their descendants. So, by default, the requirements for *Regular meeting* are inherited by *Mandatory meeting.*

As discussed in [12], varying properties of the environment can have significant effect on goal models — namely, goal/task/softgoal addition/removal, changes in VP choices and different evaluations of these choices w.r.t. softgoals. Given a context-parametrized (i.e., with contextual annotations) goal model, the algorithm for producing *context-specific* versions of it for particular sets of active contexts is also described. It removes model elements invisible in the current context. The goal modeling notation presented here is more complex compared to the notation of [12], thus requiring a modified algorithm. The additional elements — DAs, QCs, *AwReqs*, and CVs — are all context-dependent, i.e., can change from context to context. E.g., the success rate for the goal *Find available rooms* can be set to 95% in a *Regular meeting* context and to 70% in a less important *Information session* context by using the appropriate *AwReqs*. Each *AwReq* will be visible in its respective context. Similarly, variations in possible values for VPs/CVs can be represented by different VP/CV variants, each visible in their appropriate context(s).

Clearly, these goal model variations need to be reflected in the system identification process. When we do it in the particular context c, we produce the model $S_c = \{G_c, I_c, P_c, R_c\,(I_c, P_c)\}$, where G_c is context-specific goal model (a subset of the context-parametrized goal model G) generated by the modified algorithm from [12]. Then, $I_c \subseteq I$ and $P_c \subseteq P$ since some of the indicators and parameters may not be visible in c. Moreover, R_c — the set of relations between the relevant parameters and indicators in C — should be restricted to the elements of I_c and P_c (i.e., $r\,(i, p) \in R_c \Rightarrow i \in I_c \wedge p \in P_c$). While being a necessary condition, this expression does not define the relations in R_c. It is up to the modeler to identify which relations exist in the particular contexts and how they are defined using the language of section 3. Once a relationship $r\,(i, p) \in R_c$ is defined for the context c, it also applies for all the descendant contexts of c unless overridden and provided that both i and p exist in the descendant contexts.

A complete analysis of the role of contextual information on the system identification process as well as validating the ideas briefly discussed above is subject of future work. Other possible future work also include investigating: means of estimating during RE whether a particular behavior change will match the desired targets for the system's output; the effect indicators can have on one another and how to model such a qualitative relation during system identification; what other methods and concepts from the Control Theory body of knowledge could be applied in our approach; how does this approach affect traditional Requirements Engineering activities (e.g., stakeholder negotiation during requirements elicitation); how can our proposal contribute to requirements evolution (i.e., changing the goal model because it does not properly represent current stakeholder requirements, despite the system's adaptive capabilities); etc.

Finally, the full potential of the proposal presented in this paper will be realized in the next steps of our research, which includes the development of a framework that implements adaptivity in a target system using feedback loops. With *AwReqs* [18] and qualitative relations in the requirement model, it is now possible to develop such a framework that will provide reconciliation (attempt to satisfy the requirements after failures) and compensation (resolve any inconsistencies that failures might produce) at runtime. Once we have developed such a framework, more experiments are needed to assess to what extent this approach helps in designing adaptive systems as opposed to traditional GORE methods.

In particular, we are currently working on different strategies for reconciliation. With the information that is added to the models by using the approach proposed in this paper, two basic strategies to be executed when a failure is detected are: **parameter tuning** — if there are any parameters that could be modified in order to reconcile, analyze the qualitative information available and select the best course of action w.r.t. other indicators — and **abort** — if there are no parameters or the ones that exist have already been tried, tell the target system to gracefully fail or degrade performance. Other reconciliation strategies can be devised by analyzing existing proposals in the area of adaptive systems and other fields of computer science, such as fault-tolerant computing, artificial intelligence, distributed systems, etc.

8 Conclusion

In this paper, we argue that current requirements models lack an essential information needed by feedback loop controllers in order to adapt their target systems: how changes in parameters affect relevant monitored indicators. We propose a systematic approach for System Identification and, by taking a RE perspective, we use ideas from Qualitative Reasoning to cope with uncertain and incomplete knowledge about systems. Our language allows modeling of parameter-indicator relations varying precision, based on available information. We also briefly discuss the role of contextual information on this process and conduct experiments to validate our ideas.

References

1. Grl website, http://www.cs.toronto.edu/km/grl/.

2. N. Brake, J. R. Cordy, E. Dancy, M. Litoiu, and V. Popescu. Automating discovery of software tuning parameters. In *SEAMS '08: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems*, page 65, Leipzig, Germany, 2008. ACM Press.
3. S. L. Cornford, M. S. Feather, and K. A. Hicks. DDP: a tool for life-cycle risk management. *IEEE Aerospace and Electronic Systems Magazine*, 21(6):13–22, 2006.
4. G. Elahi and E. Yu. Requirements Trade-offs Analysis in the Absence of Quantitative Measures: A Heuristic Method. In *SAC '11: 26th Symposium On Applied Computing (to appear)*, Taichung, Taiwan, 2011. ACM.
5. M. S. Feather, S. Fickas, A. Van Lamsweerde, and C. Ponsard. Reconciling System Requirements and Runtime Behavior. In *IWSSD '98: 9th international workshop on Software specification and design*, page 50, Washington, DC, USA, 1998.
6. A. Finkelstein and J. Dowell. A comedy of errors: the london ambulance service case study. In *IWSSD '96: 8th International Workshop on Software Specification and Design*, pages 2–4, Mar. 1996.
7. W. Heaven, D. Sykes, J. Magee, and J. Kramer. A Case Study in Goal-Driven Architectural Adaptation. In *SEAMS '09: 2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems2*, pages 109–127, Vancouver, Canada, 2009. Springer-Verlag.
8. J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.
9. I. J. Jureta, J. Mylopoulos, and S. Faulkner. Revisiting the Core Ontology and Problem in Requirements Engineering. In *RE '08: 16th IEEE International Requirements Engineering Conference*, pages 71–80, Barcelona, Spain, 2008. IEEE.
10. B. Kuipers. Qualitative reasoning: Modeling and simulation with incomplete knowledge. *Automatica*, 25(4):571–585, July 1989.
11. A. V. Lamsweerde. *Reasoning About Alternative Requirements Options*, chapter 20, pages 380–397. Springer, 2009.
12. A. Lapouchnian and J. Mylopoulos. Modeling domain variability in requirements engineering with contexts. In *ER '09: 28th International Conference on Conceptual Modeling*, pages 115–130, Gramado, Brazil, Nov. 2009. Springer.
13. A. Lapouchnian, Y. Yu, and J. Mylopoulos. Requirements-Driven Design and Configuration Management of Business Processes. *Business Process Management*, 4714:246–261, 2007.
14. E. Letier and A. van Lamsweerde. Reasoning about partial goal satisfaction for requirements and design engineering. In *FSE '04: 12th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 53–62, 2004.
15. W. Ma, L. Liu, H. Xie, H. Zhang, and J. Yin. Preference Model Driven Services Selection. In *CAiSE '09: 21st International Conference on Advanced Information Systems Engineering*, pages 216–230, Amsterdam, 2009. Springer.
16. J. Mylopoulos, L. Chung, and E. S. K. Yu. From object-oriented to goal-oriented requirements analysis. *Communications of the ACM*, 42(1):31–37, 1999.
17. M. Salehie and L. Tahvildari. Towards a Goal-Driven Approach to Action Selection in Self-Adaptive Software. *Software Practice and Experience*, 2011.
18. V. E. S. Souza, A. Lapouchnian, W. N. Robinson, and J. Mylopoulos. Awareness Requirements for Adaptive Systems. In *SEAMS '11: 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, Honolulu, USA, 2011. ACM.
19. Y. Wang and J. Mylopoulos. Self-repair Through Reconfiguration: A Requirements Engineering Approach. In *ASE '09: 24th IEEE/ACM International Conference on Automated Software Engineering*, Auckland, New Zealand, 2009.