Soft-Goal Approximation Context Awareness of Goal-driven Self-Adaptive Systems

Aurélien Vialon
National Institute of Informatics/Sokendai
Tokyo, Japan
aurelien-vialon@nii.ac.jp

Kenji Tei - Samir Aknine
National Institute of Informatics - Université Lyon1/CNRS
Tokyo, Japan – Lyon, France
tei@nii.ac.jp –samir.aknine@univ-lyon1.fr

***Abstract*—In goal-driven self-adaptive systems, a goal model is used as a requirement model and is held by the system even at runtime. At this moment, the self-adaptive system, which can change its behaviour at runtime, will be able to reason over the variability within the goal model. It will then be able to find the best behaviour to deal with environment evolutions. However, the uncertain nature of the requirements engineering concepts in a real dynamic world is not always rightfully defined. In particular, quality requirements runtime changes are almost never considered in the literacy. Specifically, a problem we highlight here is the lack of context consideration in quality constraint approximation. Our purpose is to clearly define this problem and to propose a first solution. In this paper, we introduce a flexible version of the quality constraints. This new kind of quality constraints can be rewritten at runtime to tackle the context change induced by the environment change. To stick the constraint definition to the context change, we developed a new algorithm which modifies the specification of the quality constraints at runtime.**

***Index Terms*—Self-Adaptive systems, Requirements Engineering, Goal-driven self-adaptive systems, Quality Constraint, Soft-Goal, Context Awareness.**

I. INTRODUCTION

Self-adaptive systems are very useful systems to deal with the runtime environment evolution [1]. Such systems are indeed able to adapt their behaviour regarding environment evolution. Nonetheless, self-adaptive systems need to get the adaptation process at design time. A field addressing this problem is the "Goal-driven adaptation" [2]. During development process, the engineers elicit all the requirements of a system-to-be. These requirements are represented in a goal model as shown in Fig 1. This goal model shall respect an ontology [3]. Briefly, it is necessary to bear in mind that a goal model contains several components: the goals (represent functional objectives of the system-to-be), the tasks (fixe how the goals will be fulfilled), the domain assumptions about the environment (to ensure the fulfilment of the goals), and finally the soft-goals. These latter ones are prime importance in our method. According to the requirements engineering ontology, soft-goals represent the quality expectations, or quality requirements, of a system-to-be. They are called "soft-goals" because we do not exactly know at development phase what we should expect in terms of quality [3]. There are several reasons for that. The most obvious one is because a goal model contains, most of the time, more than one quality requirement. If several quality requirements are present, their definition will depend on the others quality requirements. They are indeed related across the goal model. The basic connection between them could be "if we take care of one quality, then we do not take care of another one".

In goal-driven adaptation approach, adaptation will be decided according to an expected soft-goals global satisfaction. On the basis of this satisfaction, a self-adaptive system will thus find the best goal model configuration regarding the current environment [4] [5]. In these systems, soft-goals are approximated into quality constraints to give a stronger version of the quality, a measurable version, which will be usable at runtime by the self-adaptive system [3].

What we aim to highlight here is that it is widely accepted that quality requirements cannot be clearly defined at design time [3] [4] [6] [7]. This point creates an uncertainty about the definition of such kind of requirements and about how they can be fulfilled. The unclear concept of "soft-goal" is the representation of the difficulty to well-define the quality requirements [3]. The question is why this uncertainty is not kept when we approximate soft-goals into quality constraints? One part of this uncertainty is the context uncertainty, related to the context within these quality requirements are defined [7]. By reducing soft-goals into approximated quality constraints, this uncertainty disappears and, then, a possible source of adaptation for the self-adaptive system-to-be disappears too. This is the problem we wish to expose in this paper, the loss of adaptation capability induced by the non-consideration of context uncertainty during the quality constraints approximation.

Our contribution through this paper is the introduction of context awareness concept in the quality constraints approximation process [8]. Researchers have already recognised the context top citizen role in the adaptation [7] [9]. Nevertheless, this context is never treated from the point of view of quality constraints of goal-driven self-adaptive systems. We claim that a soft-goal approximation, as any other requirements, can only be pertinent in a particular context. We link the uncertainty related to the quality constraint definition to a context uncertainty [7]. Consequently, context uncertainty should be considered at runtime by the self-adaptive process to redefine the quality constraint approximations. These dynamic contextual approximations represent an alternative to reinforce the adaptation of a self-adaptive system. We then define a Quality Constraint Template to allow a better flexibility about the quality constraints in order to authorise their rewriting at runtime. This rewriting will be performed depending on the context evolution.

This article is split into six main parts. In section 2, we introduce some relevant preliminaries about the soft-goals and about the context variant problem in requirements engineering. These preliminaries are essential to understand our method. In section 3, we illustrate the problem with an E-commerce example. Section 4 sets out our solution, through the use of a constraint template and an algorithm. Both allow to rewrite the definition of quality constraints at runtime. Section 5 shows a panel of existing related proposals to compare them with our. Section 6 presents our conclusion and further developments we expect for our work.

II. Preliminaries

A. *Preliminary Definition of the Soft-Goals*

It is necessary to return to what "soft-goals" mean to well understand the problem. We already explained the difficulty to define them with precision at development phase. This is why they need to be approximated to be used in a proper way by the self-adaptive systems. This can be carried out in different ways. We chose to favour the approach of an approximation in terms of quality constraints. The latter is more relevant in the case of self-adaptive systems, practical systems, we are adressing. This form of quality requirements indeed allows to be used in the case of runtime process [3]. The use of a strict and measurable definition of soft-goals allows that.

The existence of quality constraints perfectly shows that the concept pointed by soft-goals cannot be well defined. To deal with this problem, it is necessary to approximate soft-goals into quality constraints. However, as for all approximations, some information are lost during the process. What we aim to prove in this paper is that the information lost in the course of the approximation could become relevant at runtime. Some changes in the environment could indeed lead to reconsider the way we approximate a soft-goal.

B. *Context Variant Problem*

Salifu, Yu and Nuseibeh [7] defined some important assertions which are at first importance for the purpose of this paper. The first one is the definition of the requirement satisfaction:

$$W,\ S \vdash R$$

where R is the requirements, W is the context that R is concerned with and S is the specifications needed to achieve R. Requirement satisfaction is thus related with the context. A change in the context does not lead to the dissatisfaction of the requirements. It leads to their modification. This is stated in the next assertions:

$$W; W_V,\ S; S\Delta \vdash R \qquad (1)$$

where $W; W_V$ denotes a contextual change which invalidates R requirements (semi-colon represents this change). $S; S\Delta$ is then the change in the specification to restore R.

However, it is sometimes impossible to restore the requirements because the context changed in a too important way. In such case, we have the last statement of the context variant problem:

$$W; W_V,\ S_V \vdash R_V \qquad (2)$$

where R_V denotes the requirements which changed their definition because of the contextual changes $W; W_V$ which invalidates R. Finally, the specification has to change too in order to stick to requirements variation.

C. *Context Variant Problem With Quality Constraints*

It is important to remind that quality constraints are also requirements (for the unique reason they approximate quality requirements). Thus, quality constraints are concerned by the context variant problem as any requirements. Especially because quality requirements are at the origin of three of the four dimensions of the context-awareness variability as Salifu et al. expressed them [7]:

1. Quality requirements that may induce variation in their satisfaction in different contexts.
2. Physical phenomena whose variations determine the satisfaction of the quality requirements.
3. Variation in applying a decision-making process to the quality requirements.

The problem we want to raise here is that soft-goal approximations can only exist in a *well-defined quality space* [3]. This restriction is acceptable insofar the definition of the expected context remains stable enough as the assertion 1 showed it. But what if it is not ? What would happen if a variation in the environment would change the *quality space* of the quality constraints ? We know this variation could have at least three dimensions related to the quality requirements. Is the defined quality constraints would remain relevant regarding the new context ? In assertion 2, the context variant problem claims that the requirements are modified when the context changes. We will now show that in the next example.

III. E-Commerce Example

The e-commerce example is particularly pertinent in our case. It proposes a short goal model which presents the kind of situation we want to address. The main purpose of the e-commerce system is to allow its users to order products sold by the website. As shown in Fig 1, some requirements are refined at development phase in order to allow the realisation of the top goal “Product be Purchased”.

At the goal-tree leaves level, we find the tasks which allow the system to act. Thus, the system can realise the various goals it has to fulfil until it can reach the top goal fulfilment. Some

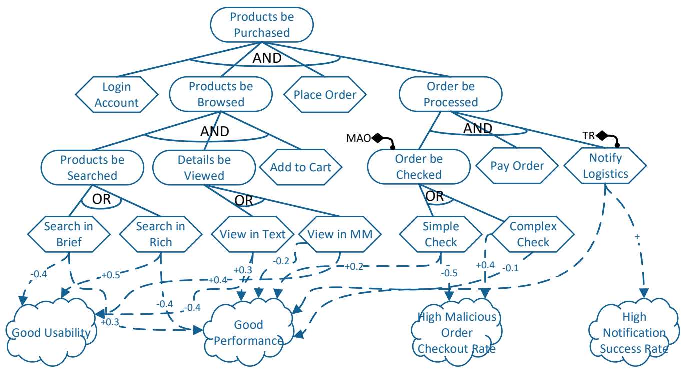

Fig 1 E-commerce goal model

alternatives are available about these tasks. For instance, to fulfil the sub-goal "Products be Searched", the system has two possibilities at runtime. It can use the task "Search in Brief" or the task "Search in Rich".

As it appears within the goal model, the different tasks of the system are linked to some clouds. These ones represent the soft-goals, the quality requirements, we have mentioned earlier. Soft-goals are linked to the system tasks thanks to some "contribution links". It means the realisation of one task will contribute, negatively or positively, to the satisfaction of the linked soft-goal [10]. These contribution links do not only contribute to the realisation of the quality requirements of the system. They are also the basis of the goal reasoning process launched in case of a detected failure.

System failures occur when a requirement violation is detected. In particular when some quality constraints are violated [10]. For this example, the four soft-goals are approximated into two different quality constraints. One of these is approximated from the "Good Performance" soft-goal and another from the four soft-goals. For this latter quality constraint, a combined utility value is calculated by doing the weighted sum of the current metrics of "Good Usability", "High Malicious Order Checkout Rate" and "High Notification Success Rate" soft-goals (with a respective weight of 0.4, 0.4 and 0.2). The different metrics (all between 0 and 1) are calculated in different ways depending on the soft-goal. "Good Usability" represent the user satisfaction through a user feedback process. "High Malicious Order Checkout Rate" has for metric the rate of malicious orders (fake orders) checkout in the total number of malicious orders. The "High Notification Success Rate" metric represents the rate of successfully notified orders to the logistic part of the e-commerce website. Finally, the system will launch an adaptation procedure when at least one of the two following quality constraints is violated:

- Response time is larger than 1000ms
- Combined utility is smaller than 0.45 and the response time is smaller than 600ms

In such case, the system will adapt its behaviour by changing the different variables representing the variability of the goal model. This variability has, here, two representations. First, the system can change its path in the goal model by switching the used branch of OR disjunctions (Variation Points) present in the goal model. This possibility is the most obvious but it is not the only one. Another variability presents in Fig 1 is the variability within each requirement. Here, two examples of such variability is present across the control variables "MAO" and "NR". The first control variable "MAO" represents the value (in dollars) of the orders that should be checked. If MAO is set at 500 dollars, only the orders with a value over 500 dollars will be checked if they are malicious or not. If for instance MAO is set at 250 dollars, much more orders should be checked. The other control variable, NR, sets the maximum number of times that the system can retry to send a notification to the logistic before consider a failure in that.

The switch of OR branches and the modification of the control variables are two possibilities for the self-adaptive system to perform an adaptation at runtime. Thanks to that, it can perform a reconfiguration of its goal model to adapt its behaviour to the environment changes. However, it may happens that the assumption which has been done to trigger the adaptation, through the definition of some quality constraints, is no longer relevant regarding to a change in the context. This is the meaning of the assertion 2 of the context variant problem [7]. In such case, the requirements vary too much to restore it with a specification change. When the requirements vary, it means the previous definition of the requirement is now invalid. In our example, we can glimpse this inconsistency. For instance, let's imagine that the task "Search in Rich" is no longer usable. For an unknown reason, the sub-system which was allowing the system to launch the task of the recommendation engine of the e-commerce website is ineffective. Then, by considering that the system is able to monitor such kinds of requirements states, it will launch an adaptation. This will lead it to switch the OR branch of the "Product be Searched" to the remaining task "Search in Brief". However, because the system can no longer use the task "Search in Rich", the soft-goal "Good Usability" will lose the positive contribution that was given by this task. On another hand, the quality constraint related to this soft-goal checks if the Global Utility (calculated by considering the satisfaction of "Good Usability") is always over 0.45. Hence, a new adaptation process will be launched to catch up the Global Utility decrease induced by the disappearance of the "Search in Rich" task.

The point we would like to arise with the development of this example is the following: is an adaptation really necessary here? Indeed, this question is very relevant in our example because the situation of the system is quite interesting. As we said, with the demise of the task "Search in Rich", the contribution given by this task disappears too. For the soft-goal "Good Utility", the contribution from the task "Search in Rich" represents more than 50% of its total positive contribution. If the system can no longer count on this contribution, some requirements definition previously relevant become irrelevant in this new context. For instance, the minimum Global Utility in the quality constraint related to "Good Usability". If we understand the meaning of the assertion 2 of the Context Variant problem, this requirement becomes irrelevant regarding the current context. How could we still expect this value when the calculation of Global Utility is now distorted? When we chose at development phase the value of this quality constraint, we chose it regarding some domain assumptions we made. And one of these assumptions was the possibility to improve this value with the help of “Search in Rich” contribution link. For what we see of this situation, it is not reasonable to consider in the requirements definition this contribution link anymore. If we would do that, the system would proceed to irrelevant adaptations. With the violation of the quality constraints, it would try to catch up a value which is not matching the current context. For instance by decreasing the MAO control variable value in order to increase the TR control variable value. That will increase the Global Utility rate by increasing the satisfaction of the “High Notification Success Rate” metric. However, we showed that such adaptation, on the based on false assumptions induced by the context change, is not necessary. For this reason, the system should grasp that the context has changed. It should be aware of that to prevent irrelevant adaptations. This can be possible by dynamically changing the definition of the conditions under which adaptation is necessary. Such kind of behaviour would also be an adaptive behaviour.

IV. Context Awareness With the Quality Constraints

We will now explain our solution to deal with the context changes and their implications on the quality constraints approximation. The first part of our solution will be the definition of a flexible template for the quality constraints. And the second part will be concerned with the runtime algorithm which uses this quality constraint template to dynamically deal with the context changes.

A. *Requirements engineering phase*

We assume that the user has already obtained a well-formed goal model. So, all the necessary requirements for the runtime process are present within this goal model. In particular, the quality constraints are already defined. The method to get a well-defined goal model may vary. We assume that the user uses a relevant method of goal-driven self-adaptive systems engineering. This method includes the refinement of the system-to-be main goal into sub-goals, themselves operationalised into tasks or domain assumptions [11].

B. *Constraint Template at Design Time*

Once each soft-goal has an approximated quality constraint after the requirements engineering phase, we need to transform these quality constraints into a more flexible version. However, we want to keep the quality constraints which have been defined for the system start. These quality constraints will actually represent the basic of the constraint template we want to build in order to make more flexible the quality constraint at runtime.

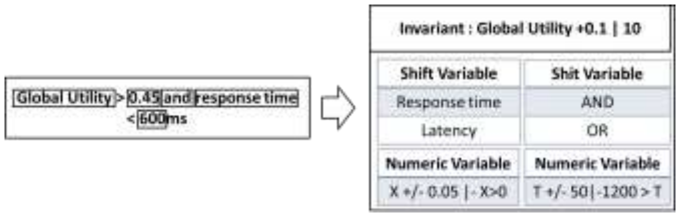

Fig 2 Quality Constraint Template Building

Quality constraints templates will be built by the definition of some flexible template variables. Template variables can only be numeric or set types. Each template variable has a unit change (how much this variable can change at each step) and a direction change (+/-, even for the set type). We will find the following template variables inheritance types: invariants and shift. The invariants are the template parts that will give the frame of the constraint rewriting. In Fig 2, “Global Utility” has been set as an invariant, because it is the only term in the quality constraint which implies a more global judgement (we want to improve its value, but a change of 0.1 is temporary acceptable). Shift variables are variables which can be exchanged with other ones at runtime. For instance in our example, “Response time” can be exchanged with “Latency”.

C. *Algorithm at runtime*

We can use the quality constraint templates at runtime to rewrite one or several constraints affected by a context change. The process describing this is divided into two procedures. The first one, shown in the Algorithm 1, draws the build of context induced by a modification in the system goal model. The second procedure, explains the dynamic rewriting process of a quality constraint.

Algorithm 1 Context Builder

```
procedure ContextBuilder(ContextVariable Cvar)
   for each c in Situation do
      if c.ID == Cvar.ID then
         c.value = Cvar.value
         for each clink in c.links do
            clink.switch()
         end for
         for each cst in c.cst do
            rewrite(cst, c.valueLink)
         end for
         break
      end if
   end for
end procedure
```

In the Algorithm 1, we change the context value of a context variable received in the parameters of the procedure. This variable represents a change in the goal model (for instance one task is no longer available). By considering a more global variable containing all the context variables (which is named “Situation”), we find the corresponding variable and update its value. For instance, if we still consider the same example where the task “Search in Rich” is no longer available, the related context value will switch from “true” to “false”. In the case of a numeric or set context values, we assume that the context variable in procedure parameters contains the new value of the variable. In order to consider the vanishing of the contribution links related to some requirements, Algorithm 1 switches the value of the corresponding links to zero.

Afterwards, the algorithm launches all the necessary procedures to rewrite the definition of the quality constraints concerned by this context change. The link between the context variables and the quality constraints is done at development phase.

Algorithm 2 QC Rewriting

```
procedure rewrite(QCst c, Int lostCLink)
    c.lastVar.weight.update(c.change)
    if c.invValue > c.invLimit then
        tempv = maxWeight(c.template.v)
    else
        tempv = minWeight(c.template.v)
    end if
    if lostCLink<0||c.invValue>c.invLimit then
        tempv.increase()
    else
        tempsv.decrease()
    end if
end procedure
```

To perform the quality constraint rewriting in Algorithm 2, the system first considers the total invariant change which allows to keep an anteriority of the quality constraints rewriting. This value is calculated by adding all the changes (in template variable unit) between the current value and the former one for all the template invariants. That will give the invariant changes. Then, the system modifies the weight of previous unit choice that it did. For example, let's imagine that in a first step, the system changed the X value in Fig 2 by one unit. When it will return to the Algorithm 2 the system will notice that the Invariant Change is 3 units. Then, the new value of the X unit weight will be (1 + 3) / 2 = 2. We do not directly replace the unit weight value by invariant changes value, because an uncertainty exists about the latter one.

After this update process, our system will have two choices. Either its current invariant value not exceed its limit or it is the case. In the second situation, the system will try to immediately return to an acceptable value by using its best positive change choice. In the other situation, the system will find the template variable which has the less weight value. The procedure which performs this task is not shown here. It considers all the template variables of the constraints (excluding the invariants of course) and chooses the ones which have the less weight. If there was only one template variable which has the less weight, then the system returns to the Algorithm 2. On the other hand, if the number of template variables kept from the previous step is more than one, the system will perform a random choice to define which template variable it will keep.

Once the system has got the template variable (with the biggest weight or the less), it will update its value by one unit as it has been defined at design time when the developers built the constraint template. The decision of the update direction will be done by the system on the base of the integer value it received as procedure parameter. In the case this value is positive, it means that the contribution link which has been lost was positive for the respect of the constraint (it's mandatory to build the quality constraints in that way to ensure the good working of the system). So, in this case, the system has to decrease the value of the template variable it selected.

Then, by changing the values of the different template variables, the system will be able, slowly, to modify the quality constraints definition. That will dynamically redefine them in order to connect them to the new context. After one of the template variable has been modified thanks to the Algorithm 2, then the process related to this quality constraint rewriting is paused. The goal-reasoning adaptation process receives the new constraint and run with this modified requirement. In the next cycle, the quality constraint rewriting is reactivated and change again one template variable.

The rewriting process is stopped when the system considers the modified quality constraint is near enough from the new context. That happens when some predefined conditions are fulfilled: if the success rate of the quality constraint is high enough, if the global satisfaction of the system is at its higher value (regarding the results of the performed cycles) and finally if Invariant Change value is stable since a certain number of cycles.

V. Related Works

The literacy about the context awareness problem for the requirements is largely considered. However, the context is not often considered as a first-class citizen in requirements engineering and even less in the self-adaptive systems area. The researchers simply neglect it. Most of the time this is due to the difficulty to grasp the real meaning of the context or to understand the variables that will be implied in the definition of these contexts. Existing works tackle the same problem than us by avoiding the context concept in the soft-goal approximation into quality constraints. They only address this context question from the contribution links point of view [6]. There is thus a lack about this particular point.

Works using relaxation processes for adaptation do not consider the context as first-class citizen. They try to catch the context consequence evolution by some tricks. It's for instance the case of the RELAX language [4]. RELAX addresses to the uncertainty that a system could meet at runtime. This language uses some relaxation keywords (SHALL, SHOULD…) which allow, depending on what we specify at design time, the relaxation of some constraints definition. For instance, by keeping in mind a global satisfaction rate which has to be the highest possible, RELAX will make possible the relaxation of some constraints to allow the system to continue to work even if the context has changed. Zanshin framework works in a very similar relaxation way [5]. Zanshin uses the variation points and the control variables in its adaptation process, but this is not its particularity. Actually, Zanshin proposes two new kinds of requirements: the awareness requirements and the evolution requirements. The awareness requirements basically allow the relaxation of the success rate of the other system requirements. For instance, if one quality constraint fails, it's not necessarily a problem as long as the failure rate of this constraint remains in the accepted failure space defined by the awareness requirements. On their side, evolution requirements permit the modification of some requirements depending on some pre-defined conditions. Whatever we consider RELAX language or Zanshin framework, the context is never made explicit like in our proposal and they basically deal more with the Assertion 1 of the context variant problem.

On another hand, there are some proposals which explicitly consider the context in their process. It's the case of the TROPOS4AS Framework. From the first TROPOS version, context was understandable by an actor distinction during the requirements engineering phase. This distinction allows the consideration of several values of contribution links to the same soft-goal, depending on which actor was contributing. In the following iterations of TROPOS, the context was even made explicit and is conditioning the values of the contribution links [12]. A very similar point of view also exists in the works of Lapouchnian and Mylopoulos [13]. In both cases, some conditions which express the various possible contexts lead to the consideration or not of these contributions links. A related kind of work addresses the same contextual contribution link question by proposing to dynamically change at runtime the values of the contribution links. For instance, a case-based reasoning solution is used in such proposal [6]. Our work does not allow for the moment to modify the contribution links values when a contextual inconsistency is detected (except the ones directly related to the contextual change). Nonetheless, it allows the rewriting of the quality constraints which are impacted by this change in the quality satisfaction possibility. We then propose to adapt the "when" an adaptation is necessary, whereas the works about contribution links contextual update deal with "how" an adaptation will be perform.

VI. Conclusion and future work

We have briefly introduced the necessity to consider the context in a dynamic vision of the soft-goal approximation into quality constraints. Our solution is composed of two parts, the quality constraint template definition and the runtime algorithm, which address the two situations of a goal-directed requirements self-adaptive system: the development phase and the runtime. Now, we need to implement our runtime algorithm in order to prove our vision compared to what our proposal allow in terms of overall adaptation quality. Our first track to evaluate the success of our method is to measure the overall satisfaction of the system quality with our solution and without. Another track could also be to consider in the same time the total number of required adaptations. If we reach a better, or a quite similar, overall satisfaction of the system quality by launching far less adaptation processes, it would be a significant improvement.

Regarding the contributions links, we know that there is an uncertainty about the impact of the disappearance of one contribution link on the others. The value of this impact is impossible to know because it is depending on various parameters. For example, the current values of the related variable have a repercussion on the new value of the contribution links. However, we finally believe that all these contributions links are related through the soft-goal satisfaction process. We will then need to define a system compatible with our rewriting of quality constraints in order to recalibrate the remaining contribution links of the system.

About the definition of the context, we are currently using the same definition than the context variant problem. However, we are aware that we will need to develop our own definition of context, regarding the very particular context point we aim to address with the quality constraints.

We also need to improve our runtime process in order to allow more context cases that we used here. Our current idea about that is to implement a genetic algorithm which will run among the adaptive process to simulate the various possibilities of context evolution regarding all the monitoring values of the system.

VII. References

[1] Rogerio de Lemos et al, "Software Engineering for Self-Adaptive Systems: A Second Research Roadmap" in *Self-Adaptive Systems, LNCS*, pp. 1-32, 2013.

[2] Mirko Morandini , Loris Penserini, and Anna Perini, "Towards Goal-Oriented Development of Self-Adaptive Systems" in *SEAMS'03*, Leipzig, Germany, 2008.

[3] Ivan J. Jureta, John Mylopoulos, and Stéphane Faulkner, "Revisiting the Core Ontology and Problem in Requirements Engineering" in *16th IEEE International Requirements Engineering Conference (RE'08)*, 2008.

[4] Jon Whittle, Pete Sawyer, Nelly Bencomo, Betty H. C. Cheng, and Jean-Michel Bruel, "RELAX : a language to address uncertainty in self-adaptive systems requirement" in *RE*, 2010, pp. 177-196.

[5] Vitor Estêvao Silva Souza, "Requirement-Based Software System Adaptation", Thesis 2012.

[6] Wenyi Qian et al., "Rationalism with a Dose of Empiricism: Case-Based Reasoning for Requirements-Driven Self-Adaptation" in *RE*, Karlskrona, Sweden, 2014.

[7] Mohammed Salifu, Yijun Yu, and Bashar Nuseibeh, "Specifying Monitoring and Switching Problems in Context" in *RE*, 2007.

[8] Joëlle Coutaz, James L Crowley, Simon Dobson, and David Garlan, "Context is key" in *Communications of the ACM*, pp. 49-53, March 2005.

[9] Alistaire Sutcliffe, Stephen Fickas, and McKay Moore Sohlberg, "Personal and Contextual Requirements Engineering" in *RE*, 2005.

[10] Sotirios Liaskos, Rina Jalman, and Jorge Aranda, "On Eliciting Contribution Measures in Goal Models" in *RE*, 2012.

[11] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas, "Goal-directed requirements acquisition" in *Science of Computer Programming*, vol. 20, pp. 3-50, 1993.

[12] Fabiano Dalpiaz, Paolo Giorgini Raian Ali, "A goal-based framework for contextual requirements modeling and analysis" in RE, 2010, pp. 439-458.

[13] Alexei Lapouchnian and John Mylopoulos, "Modeling Domain Variability in Requirements Engineering with Contexts" in *International Conference on Conceptual Modeling*, 2009.