Planning Runtime Software Adaptation through Pragmatic Goal Model

Felipe Pontes Guimarães[a,*], Genaína Nunes Rodrigues[b], Raian Ali[c], Daniel Macêdo Batista[a]

[a] *Department of Computer Science, University of São Paulo (IME-USP), Brazil*
[b] *Department of Computer Science, University of Brasilia, Brazil*
[c] *Faculty of Science and Technology, Bournemouth University, United Kingdom*

Abstract

Adaptivity is a capability that enables a system to choose amongst various alternatives to satisfy or maintain the satisfaction of certain requirements. The criteria of requirements satisfaction could be pragmatic and context-dependent. Contextual Goal Models (CGM) capture the power of context on banning or allowing certain alternatives to reach requirements (goals) and also deciding the quality of those alternatives with regards to certain quality measures (softgoals). It is used to depict facets of the decision making strategy and rationale of an adaptive system at the preliminary level of requirements. In this paper we argue the case for *pragmatic requirements* and extend the CGM with additional constructs to capture them and allow their analysis. We also develop an automated analysis which aids the planning and scheduling of tasks execution to meet pragmatic goals. Moreover, we evaluate our modelling and analysis regarding correctness and performance. Such an evaluation showed the applicability of the approach and its usefulness in aiding sensible decisions. It has also shown its capability to do so in a time short enough to suit run-time adaptation decision making.

Keywords: Requirements Engineering, Adaptive Systems, Context-awareness, Quality of Service

1. Introduction

Adaptive systems are designed to enjoy a degree of flexibility in meeting their requirements and maintaining them. Adaptivity requires variability so that the system can choose amongst alternatives and optimize certain performance indicators. In requirements engineering literature, a main-stream model

*Corresponding author

Email addresses: felipepg@ime.usp.br (Felipe Pontes Guimarães), genaina@cic.unb.br (Genaína Nunes Rodrigues), rali@bournemouth.ac.uk (Raian Ali), batista@ime.usp.br (Daniel Macêdo Batista) to capture and analyse such variability is Goal-Model (GM) [1]. It provides the requirements (goals) for which the system is designed and the various possible ways to reach those goals (alternatives) and their quality (soft-goals). It also allows breaking down the system to a set of autonomous entities (actors) who can also decide adaptively how to depend on each others to reach requirements.

Adaptivity is triggered by contextual factors which could be internal, e.g. errors and availability of computational resources, or external, e.g., newly available services and packages, physical and social environment of the user, their skills and computing device. In Ali et al. [2], traditional goal models [3, 4, 5] were extended to capture the notion of context and its relation with requirements. The proposed Contextual Goal Model (CGM) treats context as an adaptation driver which can help filtering the space of applicable alternatives to reach goals and dependencies between actors and deciding the quality of such alternatives and dependencies with regards to certain quality measures (softgoals). The Runtime Goal Models proposed in Dalpiaz et al. [6] elaborate on specifying the possible valid alternatives to reach goals and the possible sequences for that achievement.

However, we advocate that requirements satisfaction is itself *pragmatic*. Pragmatism is the capability of solving problems in a sensible way that suits the current context rather than obeying fixed rules. This implies that goals refinements are not causal relations and the mere achievement of the sub-goals or execution of tasks does not necessarily imply that the parent goal has been achieved. There may be quality requirements to be achieved in order to consider a goal satisfied. Even more, such requirement can be lightened and strengthened dynamically depending on the context.

As an example, let's consider a scientifical paper's submission process. To have a paper accepted, one must have an idea, write about it, experiment on it and submit a paper. These steps could be seen as the decompositions of "Having a paper accepted" goal. However, performing these tasks would mean little if the submission's deadline is not met. This is a hard and clearcut quality criteria for achieving the root goal. Failing to meet this time constraint means failing the goal altogether, thus rendering it a pragmatic goal.

In Guimarães et al. [7], we have introduced the Pragmatic Goal Models (Pragmatic GM) embracing the concept of pragmatic requirements and pragmatic goals to grasp and model the idea that a goal's interpretation varies according to context. We have also developed an algorithm to compute if a goal is achievable under context-dependent quality of service (QoS) constraints and, if so, it returns a set of tasks that abide by such constraints.

In this paper we extend our model with goal annotations proposed by Dalpiaz et al.[6], which allows the Pragmatic GM to specify the runtime system behaviour. Now, instead of providing a simple *yes/no* goal achievability answer and applicable alternatives sets, through our new analysis and planning algorithm, the Pragmatic GM provides, through the new Pragmatic Planning Algorithm (PPA), a comprehensive goal fulfilment plan which achieves the pragmatic requirements and respects the allowable system behaviour at runtime. On top of that, we have conceived the new PPA algorithm certifying that it is still within a complexity suitable for runtime decision making. By these means, runtime adaptation of a software system benefits from our planning algorithm so as to schedule tasks execution and, therefore, meet the intended pragmatic goal. We evaluate our approach using a Mobile Personal Emergency Response System (MPERS) case study [9] and a scalability analysis. Results show that a plan for the MPERS case study was generated in less than a second. While scalability results point out that even for models with up to 10000 goals and 20 contexts sets the analysis and corresponding planning outcome can be reached within a minute.

The paper is organized as follows. Section 2 presents the CGM and RGM models on which the Pragmatic GM builds and extends. Section 3 presents the pragmatic goal and achievability concepts. Section 4 presents the proposed model, its component parts and the automated reasoning and plan engineering algorithm to find a suitable execution plan for the pragmatic model. Section 5 evaluates the applicability of our modelling and analysis approach. Section 6 presents related work and Section 7 concludes the paper and outlines our future work.

2. Goal Models

Goal-Models (GM) are well established requirements engineering tools to depict and break-down systems using socio-technical concepts [1]. In other words, they provides the goals for which the system should be designed and the various possible ways to reach those goals. However, as pointed out in Ali et al. [2] and Dalpiaz et al. [6] three aspects of real-life cannot be captured in the traditional goal model [3, 4, 5]: the notion of contexts [2], the determination whether a task sequencing is valid within the model and the exploration of alternative system configurations to restore the system to a valid state [6].

2.1. Contextual Goal-Model

Contextual Goal Model, proposed in [2], are meant to capture the relation between requirements and their dynamic environment. Context can guide adaptation and support the decision in the goals to activate and filter the space of alternative strategies - subgoal, task or delegation - which can be applied to achieve activated goals. Context can also have an effect on the quality of those alternatives and this is captured through the notion of contextual contribution from goals and tasks to softgoals.

Context is description that concretize relevant factors in the system's environment, *i.e.*, the surrounding in which it operates [8]. In goal modelling terminology, context is a specification of a partial state of the world relevant to an actor's goals [2]. Actors are social entities, or software representing them, in an organization. Actors exist to have and be responsible of achieving goals and they have degree of flexibility how to achieve them. A context that affects that choice could be the time of the day, a weather condition, patient's chronic cardiac problem, etc.

Fig. 1 present a CGM that depicts the requirements of a Mobile Personal Emergency Response System meant to respond to emergency situations for people in an assisted living environment. The root goal is “respond to emergency", which is performed by the actor `Mobile Personal Emergency Response`. The root goal is divided into 4 subgoals: “emergency is detected", “[p] is notified about emergency", “central receives [p] info" and “medical care reaches [p]” ([p] stands for “patient”). Such goals are then further decomposed, within the boundary of an actor, to finally reach executable tasks or delegations to other actors. A task is a process performed by the actor and a delegation is the act of passing a goal on to another actor that can perform it.

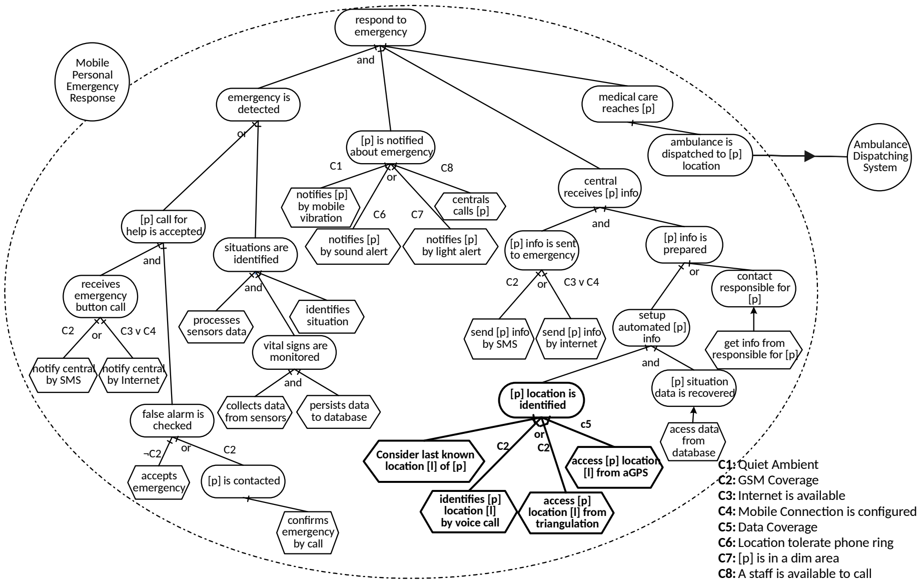

Mobile Personal Emergency Response
respond to emergency
and
emergency is detected
or
[p] is notified about emergency
C1
or
C8
notifies [p] by mobile vibration
C6
C7
centrals calls [p]
notifies [p] by sound alert
notifies [p] by light alert
medical care reaches [p]
ambulance is dispatched to [p] location
Ambulance Dispatching System
central receives [p] info
and
[p] info is sent to emergency
C2
or
C3 v C4
send [p] info by SMS
send [p] info by internet
[p] info is prepared
or
contact responsible for [p]
get info from responsible for [p]
setup automated [p] info
and
[p] situation data is recovered
acess data from database
[p] location is identified
or
C2
C2
c5
Consider last known location [l] of [p]
identifies [p] location [l] by voice call
access [p] location [l] from triangulation
access [p] location [l] from aGPS
[p] call for help is accepted
and
receives emergency button call
C2
or
C3 v C4
notify central by SMS
notify central by Internet
false alarm is checked
¬C2
or
C2
accepts emergency
[p] is contacted
confirms emergency by call
situations are identified
and
processes sensors data
identifies situation
vital signs are monitored
and
collects data from sensors
persists data to database
C1: Quiet Ambient
C2: GSM Coverage
C3: Internet is available
C4: Mobile Connection is configured
C5: Data Coverage
C6: Location tolerate phone ring
C7: [p] is in a dim area
C8: A staff is available to call

Figure 1: A Pragmatic GM for responding to emergencies in an assisted living environment (adapted from Mendonça et al. [9])

It is important to highlight that not all the subgoals, delegations and/or tasks are always applicable. Some of them depend on certain contexts whether they hold.

2.2. Runtime Goal-Model

In [6] Dalpiaz et al. present a framework for bridging the gap between design-time goal-models and runtime behaviour. Starting with an early requirements model representing stakeholder goals (*Design-time Goal Model* or *DGM*) they refine it with additional behavioural detail about how goals are to be achieved. Specifically, they add constraints on valid orderings for pursuing sub-goals thereby creating a *Runtime Goal-Model* (*RGM*).

Annotation	**Meaning**
$R_1; R_2$	Sequential execution of R_1 then R_2
$R_1 \# R_2$	Parallel execution of R_1, R_2 or both
$R^{k\#}$	Parallel execution of k instances of R, with $k > 0$
R^{k+}	Sequential execution of k iterations of R, with $k > 0$
$try(R_1)?R_2 : R_3$	Attempt R_1. If R_1 is fulfilled, R_2; otherwise R_3
$R_1 \| R_2$	Alternative execution of R_1 or R_2, not both
$opt(R)$	R is optional
skip	do nothing;

Table 1: Goal annotations syntax and meaning [6]

Dalpiaz argues that DGMs are in some ways too abstract to enable runtime monitoring of requirements. In this sense, they pose some questions:(1) Is observed behaviour compliant with the system specification?; (2) How does system behaviour relate to the fulfilment of stakeholders (root) goals? and; (3) Can the system switch to an alternative behaviour to restore fulfilment?

Furthermore, they state that such questions may be projected over a time frame, similarly to the Awareness Requirements proposed by Silva Souza et al. [10]. In this sense, one would also be able to question: (4) What is the percentage of success for a given goal during the last month? and ; (5) What is the trend for failure of a given goal in the last week?

Dalpiaz's work on runtime goal models focus on the first two questions: providing algorithms for deciding whether a given execution trace is compliant with the specification and with the relationship of the system behaviour with the stakeholders' goals.

To do so, Dalpiaz proposed the goal annotations depicted in Table 1. In addition to the traditional AND- and OR- decomposition semantics, Dalpiaz's goal annotations guarantee it is fulfilled in accordance with the runtime rules associated with R for sequential and parallel executions.

Suppose a goal R is decomposed into R_1 and R_2. While the AND/OR-decomposition rules define if all of R subgoals must be achieved, the goal annotations define the order in which they may be achieved. As such, an $(R_1; R_2)$ annotation requires the fulfilment of R_1 prior to R_2, while an $(R_1 \# R_2)$ does not impose any order on the fulfilment of R_1 and R_2 allowing for parallel execution of both.

For the other rules, the fulfilment of the node follows runtime rules. If R must be repeated k times, k instances of R must be executed serially (R^{k+}) or in parallel $(R^{k\#})$. The ternary try $(try(R_1)?R_2 : R_3)$ is interpreted as the attempt of R_1 followed by R_2 (if R_1 succeeds) or R_3 (should R_1 fail). Finally, alternative rule $(R_1|R_2)$ is interpreted of either R_1 or R_2, but not both.

In the remainder of this paper, we have tackled the concept brought about by Dalpiaz's third question: the search, within the acceptable system configurations, of some set up to allow it to restore fulfilment of the root goal.

3. Pragmatism in Requirements

In Goal Modelling, the relations between sub-nodes and parent nodes are supposed to be causal. Achieving one (OR-Decomposition) or all (AND- Decomposition) of the subgoals is seen as a satisfactory precondition for achieving the parent goal. However, in [7] we argued that the achievement of goals would in certain cases need to be seen in a pragmatic fashion and not as a direct causation of the achievement of other goals or the execution of certain tasks. The decision whether a goal is achieved could be context-dependent. Our work advocates the need for a more flexible definition of goals to accommodate their contextual interpretation and achievement measures.

It is paramount to note that the representation of the perceived quality of a goal as a softgoal is innately different from the pragmatism of the goal achievement. From a pragmatic point of view, a proposition is true if it works satisfactorily. Thus, a pragmatic goal is achieved if it provides a satisfactory quality level. It is not a matter of achieving it with higher or lower quality, but achieving it at all. Such level depends on the model but may also depend on the system's context at runtime, which may render it more or less strict for distinct situations.

Take the example of Fig. 1: in general, let's consider that the ambulance may take up to ten minutes to arrive. For a patient with a minor discomfort it can take its time and arrive in 20 minutes without suffering any penalty. On the other hand, if the patient is having a heart attack, one cannot say the goal was achieved. In these situations, the delivered level of quality may not be a separate part from the boolean answer of whether a goal is achieved or not, but an integral part of it.

A pragmatic goal model, just like a regular goal-model, describes the means to achieve it but it also describes the interpretation of such achievement. This interpretation, which depicts the goal's pragmatic fulfilment criteria, can be expressed as a set of quality constraints (QCs). Unlike softgoals, which are a special type of goal with no clear-cut satisfaction criteria[10], these QCs are mandatory and crisp, therefore quantifiable, constraints needed for the fulfilment of a goal and an inherent part of its definition. For instance, take goal "[p] location is identified" from Fig. 1($\mathbf{G}_{loc}$ for short): it could be defined as "in order to reach $\mathbf{G}_{loc}$, the location must be identified within an error radius of maximum 500 meters and in less than 2 minutes".

But then again, this would not suffice as a radius of 500m and 2 minutes might be an over-relaxed condition for patients under critical conditions.This brings into light another aspect to be taken into account for the pragmatic requirements: the fact that the interpretation for the achievement of a goal is itself context-dependent. We consider that there is a default condition for achieving a goal. However, for specific contexts, we could relax or further strengthen the condition which interprets whether a goal is achieved. We propose that the contextual QCs on the achievement of a goal should be captured together with the other effects of context in the CGM. One advantage of capturing the pragmatic goals within the CGM is to enable reasoning on the possibility of achieving a

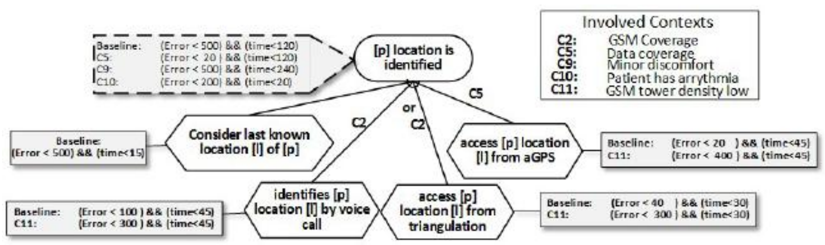

Baseline: (Error < 500) && (time<120)
C5: (Error < 20) && (time<120)
C9: (Error < 500) && (time<240)
C10: (Error < 200) && (time<20)
[p] location is identified
Involved Contexts
C2: GSM Coverage
C5: Data coverage
C9: Minor discomfort
C10: Patient has arrythmia
C11: GSM tower density low
C5
or
C2
C2
Baseline: (Error < 500) && (time<15)
Consider last known location [l] of [p]
access [p] location [l] from aGPS
Baseline: (Error < 20) && (time<45)
C11: (Error < 400) && (time<45)
Baseline: (Error < 100) && (time<45)
C11: (Error < 300) && (time<45)
identifies [p] location [l] by voice call
access [p] location [l] from triangulation
Baseline: (Error < 40) && (time<30)
C11: (Error < 300) && (time<30)

Figure 2: G$_{loc}$ graphical representation as a pragmatic goal

goal under the current context and QCs. We differentiate these interpretations in the sense that a relaxation condition is not mandatory but a requirement that further strengthen the QC must necessarily be considered.

In the previous example, a QC of getting a location within 500m in less than 2 minutes is a default constraint. However, if the user has access to mobile data (context C5) then a much preciser location can be obtained from the GPS. Under these circumstances, a lock within 500m may seem like an over-relaxed constraint. For a patient with cardiac arrhythmia (context C10), a more strict QC is needed. Suppose that the system has to ensure that an ambulance reaches the patient's home within 5 minutes. Possibly, in this case, a faster but less precise location would be better suited. The requirements for a minor discomfort (context C9) are also more flexible than those for an arrhythmia (C10). In the three specific contexts, the interpretation must differ from the original baseline.

Figure 2 sums up $\mathbf{G}_{loc}$'s interpretation criteria and presents it as a box connected to the goal itself. However, not only the goal's interpretation may vary according to the context but also the delivered quality of service (QoS) levels. When executed in different contexts, a task may provide different QoS levels, represented by the boxes linked to the tasks in Figure 2. Furthermore, provided that $\mathbf{G}_{loc}$ is annotated with `G1;G2;G3;G4`, the consequent task sequencing - derived from the goal's annotation - will too impact on the goal's perceived QoS. For instance, for $\mathbf{G}_{loc}$ this means that, although no individual task has a time constraint of over 120 seconds, unless context C9 is active at least one of such tasks will obligatorily not be chosen so that the total time stays under 120 seconds.

The set of QCs that represent the pragmatic aspects of a goal and the variable delivered QoS levels extend the traditional CGM into a Pragmatic CGM and allow for the reasoning over the achievability of a goal under a certain context.

3.1. Achievability of Pragmatic Goals

Pragmatic goals can only be achieved if their provided QoS levels comply with the QCs specified for them, both of which are context-dependent. This means that we extended the basic effect of context on a CGM to cover success and achievement criteria. Such expressiveness enables further analysis for a key adaptation decision: how to, under the current context, pick a task set and arrange them into a valid task sequence in order to reach our goals while respecting its interpretation when the goals' interpretations, the space of applicable alternatives to reach them and the QoS levels provided by the tasks are all context-dependent?

Such adaptation decision may also lead down a dire path. It is inevitable for one to wonder: "what to do when such a sequence cannot be found? ". Situations where it may not be possible to meet the goal's interpretation QoS standards through any of the applicable sub goals, tasks and/or delegations are also considered in our approach. This is a rather likely scenario considering that the tasks deliver not a static but a context-dependent QoS level. Whenever there is no possible task sequencing able to deliver the goal's required QoS we classify the goal as unachievable. This is expected to happen and the proper way of dealing with such goals is explained in the reasoning part.

Let's get back to the example in Figure 1: if we consider the goal $\mathbf{G}_{loc}$ and the contexts impacting on its interpretation, the conclusion is that under a certain context the system may not be able to determine the patient's location with the required precision. This, in practice, does not mean doing nothing. The motivation of doing this analysis and deeming the goal unachievable is simply to have such knowledge beforehand. At design-time, this allows the goal model designer to add new strategies for reaching the goal with that particular context in mind, thus covering a larger range of contexts. At runtime, this early conclusion would lead to search for a better variant at a higher-level goal by choosing another task sequencing, which is able to deliver the required quality standard. Therefore, our analysis is both meant for design-time - reasoning to evaluate and validate the comprehensiveness of the solution - and for runtime - devising the suitable planning strategy to reach goals in a specific context under a certain set of quality constraints.

4. Pragmatic Goal Model

In this section, we take the concepts for pragmatic goals and concretize them as our extension to the CGM. For each of the new constructs we elaborate on its semantics. Mainly, we have enhanced the CGM with the goal annotations for specification of valid decomposition orderings, context-dependent goal interpretations, the expected delivered QoS for tasks - also context-dependent, in order to reason about the achievability of goals and to engineer a suitable execution plan to achieve them.

4.1. Meta-Model

Figure 3 presents a conceptual model of our extension to the CGM. For the focus of this paper, the CGM could be seen as an aggregation of `Requirements`. `Requirements` may be specialized into several types: `Tasks`, `Delegations` and `Goals`. A `Delegation` represents when the `Goal` or `Task` (*dependum*) is pursued not by the current but by an external actor (*delegatee*). `Tasks` are performed by the actor in order to achieve a goal. `Tasks` may report the expected delivered quality for each metric through the `providedQuality` method.

A `Goal` is a useful abstraction to represent stakeholders' needs and expectations and they offer a very intuitive way to elicit and analyse requirements[2].

Goals have a refinements set which define the `Requirements` (subgoals, tasks and/or delegations) that can be used for achieving it as well as a method to distinguish AND- from OR-compositions. Each goal is annotated with a `Goal Annotation` which describes the required behaviour of the goal's requirement instances. Goal annotations are thoroughly discussed in Section 4.2. Context is also strongly related to goals, for it changes the current goals of a stakeholder and the possible ways to satisfy them[2]. In this sense, the goal's decomposition may vary according to the context and this is modelled in the meta-model as the method `getApplicableRefinements`.

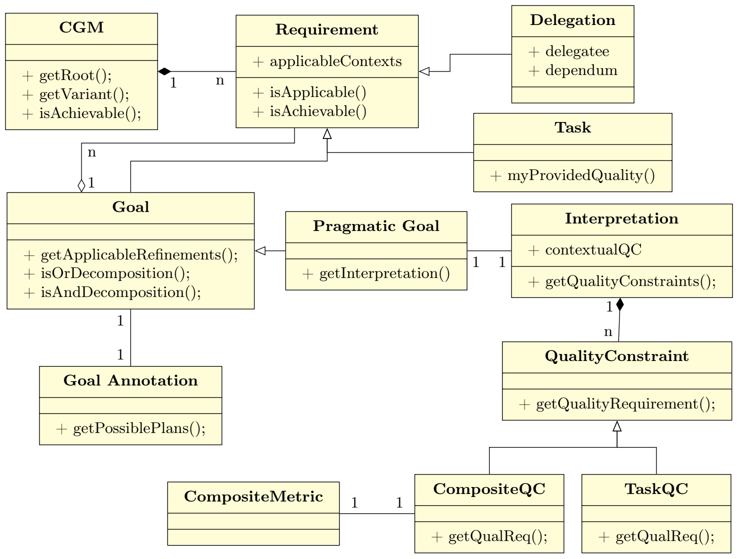


Figure 3: Metamodel for the Pragmatic Goal Model with QoS-constrained planning

`Pragmatic Goals` extend the `Goal` concept with the `Interpretation` of its achievement. A goal's `Interpretation` is an abstract concept that has the function of cross-referencing a context and the appropriate `QualityConstraints` for that given context. The `Pragmatic Goal` is said to be achieved if, and only if, such requirements are met. Otherwise, the goal's delivered QoS is considered inappropriate and the goal is not achieved regardless of achieving one or all of its `Requirements`.

The `Quality Constraints` are expressed in terms of the `applicableContext` in which it holds, the metric that should be considered, the threshold which is a numerical value that represents the scalar value for such metric and the comparison which defines whether the threshold described is the maximum allowed value or the minimum. For instance, to state a quality requirement of at most 250ms for the execution time when context $C1$ holds, the metric would be "ms", `threshold` would be 250, condition would be "Less or Equal" and `applicableContext` would be $C1$. `Quality constraints` may have task (`TaskQC`) or workflow scope (`CompositeQC`). `TaskQCs` impose a restriction on the applicable tasks whereas `CompositeQCs` restrictions apply to the workflow as a whole. The composition rules for the calculation of Composite QCs are expressed via the `CompositeMetric` objects, which are able to estimate the resulting metric depending on the goal annotation being considered (Section 4.2). The difference between these two kinds of quality constraints (QCs) is further discussed in Section 4.3 whereas their usage in the PPA algorithm is explained in Section 4.4.

Every `Requirement` inherits the `isAchievable` method. This method can be used either by the final users or by the higher level goals to define whether a particular goal can be achieved for a given quality requirement under the current context. While this is obviously necessary for the root goal, as the ultimate objective, we also allow certain subgoals to be defined as pragmatic, *i.e.*, they may also have their own predefined interpretation. In principle, actors should be able to impose further constraints on the criteria for achieving any goal within their boundary. The importance of the subgoals quality requirement becomes obvious when dealing with delegation of goals where the external actor may have itself a different, more relaxed or more strict, quality constraint, not necessarily compatible with what the delegator intends.

In this model, the goal annotations, the expectation of delivered quality by the tasks, the quality constraints for the goals, subgoals or delegations and the metrics compositional rules are added to the traditional CGM. This is meant to be done by the requirements expert or the domain experts due to the need for specialized knowledge to define such metrics.

4.2. Goal Annotations

As previously discussed in Section 2.2, goal annotations were first defined in Dalpiaz et al. [6]. They are intended to express the expected behaviour of a goal, regarding the manners in which its refinements may be sequenced. Following the meaning, allowed set of traces for goal annotations and annotation creation rules, the interleaved and sequential annotations are applicable to AND-Goals whereas try and alternative annotations are applicable to OR-Goals. The remaining annotations are unary therefore indifferently applicable to AND and OR decompositions.

In our case, goals are to be fulfilled by the system by taking into account (1) contextual specifications and (2) the possible valid configurations for goals and tasks, following RGM runtime rules, extended from Dalpiaz et al. [6].

In our model, the goal annotations are used to determine, together with the goal decomposition type, the different possible approaches for dealing with the goal's underlying requirements. It is the goal's annotation that first envisions the possible plans for achieving the goal. It is also the goal annotation, together with the Composite Quality metrics' QoS functions, that estimate the overall plan quality measures for each possibility. Only then, does the `Qos Constrained Planning` algorithm (see Section 4.4.1) compare the possible plans and chooses an appropriate one. For the sake of examplification we will deal with time and reliability constraints in the remainder of the paper, although the solution itself

Annotation		**QoS Function Time**	**QoS Function Reliability**
$R_1; R_2$		$t(R_1) + t(R_2)$	$rel(R_1) * rel(R_2)$
$R_1\#R_2$		$max(t(R_1), t(R_2))$	$rel(R_1) * rel(R_2)$
$R^{k\#}$		$t(R)$	$rel(R)^k$
R^{k+}		$t(R) * k$	$rel(R)^k$
$R_1 \| R_2$	R_1 was chosen:	$t(R_1)$	$rel(R_1)$
	R_2 was chosen:	$t(R_2)$	$rel(R_2)$
$try(R)?R_1 : R_2$	R is achievable:	$t(r) + t(R_1)$	$rel(R) * rel(R_1)$
	R is unachievable:	$t(r) + t(R_2)$	$(1 - rel(R) * rel(R_2)$

Table 2: Goal annotations Plan Alternatives and respective evaluated QoS Functions)

can be applied to any quality measure as it will be explained later on Section 4.3.

As depicted in Table 2, the runtime annotations we have used are:

Sequential ($R_1; R_2$) Serial execution of two refinements, with composed time equals to the sum of their individual execution times ($t(R_1) + t(R_2)$) and reliability equals to the chance of successfully executing R_1 and R_2 ($rel(R_1) * rel(R_2)$);

Interleaved ($R_1\#R_2$) Refinement executions performed in parallel, with composed time equals to that of the most time consuming one ($MAX(t(R_1), t(R_2))$)and reliability equals to the chance of successfully executing R_1 and R_2 successfully ($rel(R_1) * rel(R_2)$);

Sequential iterations (R^{k+}) serial execution of k instances of the plan R, with composed time equal to the time spent for a single execution multiplied by the amount of repetitions ($t(R) * k$) and reliability equals to the chance of successfully executing all the k instances ($rel(R)^k$);

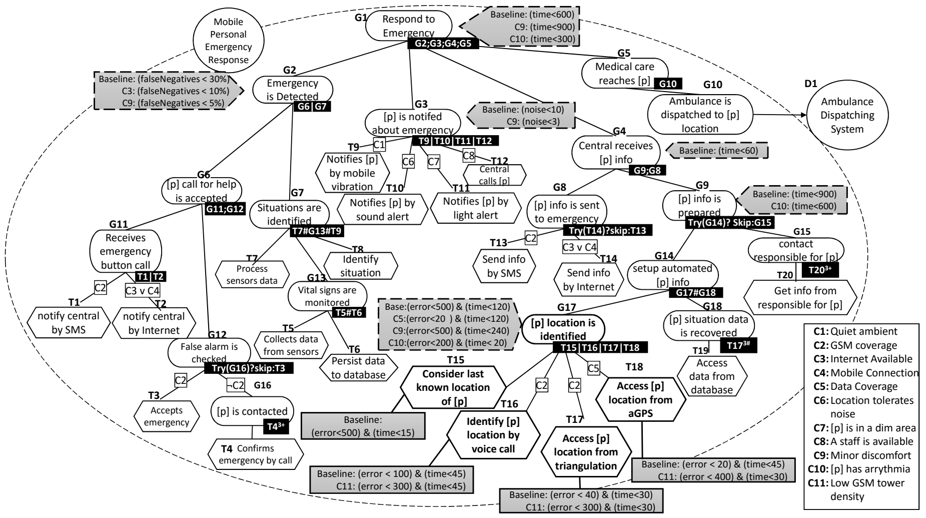
Interleaved iterations ($R^{k\#}$) Parallel execution of k instances of the plan R, with total time estimate equivalent to that of a single task ($t(R)$), since the model allows them all to be executed in parallel and reliability equals to the chance of successfully executing all the k instances ($rel(R)^k$);

Alternative ($R_1|R_2$) Alternative execution of plan R_1 or R_2, with the same time and reliability QoS levels as that of the chosen alternative;

Try ($try(R)?R_1 : R_2$) Annotation that depends on the achievability of R. If it is achievable, performs R and R_1 sequentially, otherwise, performs R - which will fail - and R_2. The time QoS is then evaluated similarly to the sequential annotation and the reliability becomes the chance of successfully executing R and R_1 (R achievable) or just R_2 (since R will fail per design).

In Dalpiaz's original work, there was also the $opt(G)$ annotation which meant G was optional and applicable only under certain circumstances. Accordingly, we map the original annotation as a context-dependency on the goal itself.

Figure 4 presents the Pragmatic GM from Figure 1 but now with annotated goals (the goal annotations are represented as the black boxes on top of the goals). Because of such annotations, the possible behaviours of the system are now thoroughly specified and the dependencies between tasks can now be represented and taken into consideration when planning the system adaptation.



Mobile Personal Emergency Response
G1 Respond to Emergency
G2;G3;G4;G5
Baseline: (time<600) C9: (time<900) C10: (time<300)
G2 Emergency is Detected
G6|G7
Baseline: (falseNegatives < 30%) C3: (falseNegatives < 10%) C9: (falseNegatives < 5%)
G3 [p] is notifed about emergency
T9|T10|T11|T12
Baseline: (noise<10) C9: (noise<3)
G5 Medical care reaches [p]
G10
G10 Ambulance is dispatched to [p] location
D1 Ambulance Dispatching System
G4 Central receives [p] info
G9;G8
Baseline: (time<60)
C1
T9 Notifies [p] by mobile vibration
C6
T10 Notifies [p] by sound alert
C7
T11 Notifies [p] by light alert
C8
T12 Central calls [p]
G6 [p] call for help is accepted
G11;G12
G7 Situations are identified
T7#G13#T9
G11 Receives emergency button call
T1|T2
C2
T1 notify central by SMS
C3 v C4
T2 notify central by Internet
T7 Process sensors data
T8 Identify situation
G13 Vital signs are monitored
T5#T6
T5 Collects data from sensors
T6 Persist data to database
G12 False alarm is checked
Try(G16)?skip:T3
C2
T3 Accepts emergency
¬C2
G16 [p] is contacted
T4³⁺
T4 Confirms emergency by call
G8 [p] info is sent to emergency
Try(T14)?skip:T13
C2
T13 Send info by SMS
C3 v C4
T14 Send info by Internet
G9 [p] info is prepared
Try(G14)? Skip:G15
Baseline: (time<900) C10: (time<600)
G15 contact responsible for [p]
T20³⁺
T20 Get info from responsible for [p]
G14 setup automated [p] info
G17#G18
Base:(error<500) & (time<120) C5:(error<20) & (time<120) C9:(error<500) & (time<240) C10:(error<200) & (time< 20)
G17 [p] location is identified
T15|T16|T17|T18
G18 [p] situation data is recovered
T17³#
T19 Access data from database
T15 Consider last known location of [p]
Baseline: (error<500) & (time<15)
C2
T16 Identify [p] location by voice call
Baseline: (error < 100) & (time<45) C11: (error < 300) & (time<45)
C2
T17 Access [p] location from triangulation
Baseline: (error < 40) & (time<30) C11: (error < 300) & (time<30)
C5
T18 Access [p] location from aGPS
Baseline: (error < 20) & (time<45) C11: (error < 400) & (time<30)
C1: Quiet ambient
C2: GSM coverage
C3: Internet Available
C4: Mobile Connection
C5: Data Coverage
C6: Location tolerates noise
C7: [p] is in a dim area
C8: A staff is available
C9: Minor discomfort
C10: [p] has arrythmia
C11: Low GSM tower density

Figure 4: The goal-annotated Pragmatic Goal for MPERS

4.3. Quality Constraints

To enhance goals with context-dependent interpretation, we must revisit the classical concept of achievability of a goal to fit the nature of Pragmatic CGMs. On top of the basic context effect on a CGM, we enable a higher model expressiveness through the goal interpretation.

The goal interpretation, as stated in Section 3, is a set of mandatory and crisp, therefore quantifiable, quality constraints needed for the fulfilment of a goal. Since we are dealing with a goal annotated model and considering the tasks interdependencies are accurately modelled, we must also further divide the quality constraints into two classes based on their scope: individual task-wise constraints and composite workflow-wise constraints.

Task-wise Constraints: . Many constraints deal with the individual quality provided by the task itself. These constraints can include measures like “level of precision”, “margin of error”, “noise” or “false negatives”. Any task that can isolatedly reach the level of quality stated in such constraints can be used, regardless of the remainder task choices. All constraints from Guimarães et al. [7] were task-wise constraints, given that no sequencing was made. These constraints can be added to the model directly, simply by giving them a unique label. To exemplify a task-wise quality constraint let's consider the Goal $G3$ "Notify [p] about emergency". This goal states that when the patient is in a quiet environment no task can produce noises louder than 3 dB. This is a restriction on the possible tasks but has no relationship with the tasks' interdependencies.

Workflow-wise Constraints:. Differently from the *task-wise constraints*, workflow-wise or composite constraints are applicable to the quality level of the overall chosen tasks sequencing that satisfy the root goal. In this sense, these constraints have a direct dependency on the way the tasks cooperate to fulfil a certain goal. Classic examples of composite constraints may include time expenditure. Such constraints heavily depend on the chosen tasks' and their underlying workflow's topology. For a time constraint, the workflow's critical path will be the decisive aspect for time estimates.

Provided that the composite QoS level calculation is heavily dependent on the actual QoS type being considered, each composite metric must implement a CompositeMetric object (from our meta model in Figure 3) defining the compositional rules for the calculation of CompositeQCs for each goal annotation. In this work, we have implemented CompositeMetrics for `Time` and `Reliability`.

4.4. Pragmatic Planning Algorithm - PPA

The expressiveness power added by the dynamic, context-aware, task- or workflow-wise quality constraints enable richer adaptation decisions which not only consider the static achievability but also the achievability under the dynamic context and its effect on the fulfilment criteria of a goal. The achievability of a goal and the space of adoptable alternatives to achieve it are essential information to plan adaptation, seen as a selection and an enactment of a suitable alternative to reach a goal under a certain context.

However, the enhanced expressiveness allied with the context-dependency also largely affects the model's variability. Given that each OR-Decomposition, each $R_1\#R_2$, $try(R)R_1 : R_2$, $R_1|R_2$ annotation, each context add to such variability, finding a configuration within the expected behaviour considering task and composite quality constraints may become humanly infeasible. Thus we have developed and implemented the PPA algorithm to engineer a plan that abides by the interpretation of the goals within the model, if possible, or to lay out the best effort allowed by the model under the current context.

4.4.1. Task- and Workflow-wise QoS Constrained Planning

In this Section, we present the Pragmatic Planning Algorithm, or simply PPA. In a nutshell, the PPA algorithm initially performs a depth-first filter, aggregating the Task Quality Constraints on each branch of the CGM tree, so that all task-wise constraints respect all the upper level goals' interpretations. Once it finds a leaf-node (task), it uses the collected task quality constraints to decide whether the task at hand is able to fulfil the collected constraints. Then, it returns this task and begins agglutinating, at each goal, the possible plans for its refinements, always choosing the plan that provides the best quality for the composite requirement being considered[1].

We present PPA in Algorithm 1. It implements the `Requirement` entity's *isAchievable* method (Figure 3) and correlates three context-dependent aspects from the model: (1) the applicable requirements; (2) the goals' interpretations and; (3) the delivered QoS level provided by the tasks.

The PPA algorithm decides whether the root goal is achievable and, if so, lays out an execution plan, *i.e.* a workflow, which is most likely to achieve the desired QCs with a complexity of $O(m * n)$, where m is the model size and n the amount of contexts[2].

PPA considers firstly the type of refinement it is dealing with. Should this refinement be a task (line 5) then the algorithm can decide whether this particular task abides by all of the task-wise and workflow-wise constraints (`canFulfil(interp)`) and returns a plan consisting solely of `this` refinement, already indicating whether it is a usable plan or not (lines 7 and 8).

On the more likely case that this is not a leaf-goal and, therefore that this node has refinements, the first step in the algorithm is to define the subgoal's achievement interpretation criteria as the stricter between the parent's QCs and the QCs passed in the *interp* parameter (lines 11-12). Then the algorithm invokes itself recursively for each refinement so that the plans for each of its subgoals are created (lines 13 - 16) and save them. Once it has knowledge of the outputted plans for its refinements, it can begin on the workflow-wise QoS constrained planning. At this point, the PPA algorithm interacts with the `goal annotation` to obtain all the allowed behaviours for the achievement of this refinement (line 18), already annotated with time and reliability QoS metrics calculated through the rules presented in Table 2. Initially, the algorithm accepts any approach to realize the goal at hand available, disregarding whether it is achievable or not (line 22). Then, for each successive possible approach (line 20), the algorithm compares the `candidateApproach` with the previously `chosenPlan` to choose the best alternative in terms of achievability (line 24-25) or QoS (lines 26-27), always striving to return the approach with the best possible QoS for the Composite QC being considered in the given interpretation.

The PPA algorithms validates whether the best candidate plan is still within the interpretation's composite QC limits (line 31) and, if it is not, sets the plan as unachievable.

Either way, the best candidate plan - achievable or the goal's best effort - is finally returned in the end. To exemplify the algorithm, we now present the following scenario: a patient with a heart condition ($C10$) needs medical assistance and needs certainty that it will arrive. He currently has good data coverage ($C4$) and mobile data enabled ($C5$). He is also in a dim location ($C7$)

[1] Currently, only one composite requirement can be considered at each QoS constrained planning.

[2] Demonstration at `https://github.com/felps/PragmaticGoals/blob/master/QosPlanningAlgorithmComplexity.pdf`

Algorithm 1 Pragmatic Planning Algorithm

```
1: procedure QOSCONSTRAINEDPLANNING(Context current, Interpretation
   interp)
2:     if !isApplicable(current) then
3:         return NULL
4:     end if
5:     if (isTask()) then
6:         Plan pTask ← new Plan(this);
7:         pTask.setAchievable(canFulfill(interp));
8:         return pTask
9:     end if
10:    Map<Refinement, Plan> refPlans;
11:    Interpretation stricter =
12:                stricterQC(current, interp,this.getInterpretation());
13:    for each Refinement ref in getApplicableDependencies(current)  do
14:        Plan pRef;
15:        plan ← ref.qosConstrainedPlanning(current, stricter);
16:        refPlans.put(ref, plan);
17:    end for
18:    List<Plan> approaches ← getRuntimeAnnotation()
                                          .getPlans(refPlans);
19:    Plan chosenPlan ← NULL
20:    for each Plan candidateApproach in approaches  do
21:        if chosenPlan == NULL then
22:            chosenPlan ← candidateApproach;
23:        else
24:            if (!chosenPlan.getAchievable() AND
             candidateApproach.getAchievable() ) then
25:                chosenPlan ← candidateApproach;
26:            else if (chosenPlan.getAchievable() AND
                   candidateApproach.getAchievable()) then
27:                chosenPlan ← chooseBetterPlan(candidateApproach,
                                              chosenApproach, interp);
28:            end if
29:        end if
30:    end for
31:    if !interp.withinLimits(chosenPlan) then
32:        chosenPlan.setAchievable(false);
33:    end if
34:    return chosenPlan
35: end procedure
```

which tolerates phone ring ($C6$). However, the area's GSM tower density is low ($C11$).

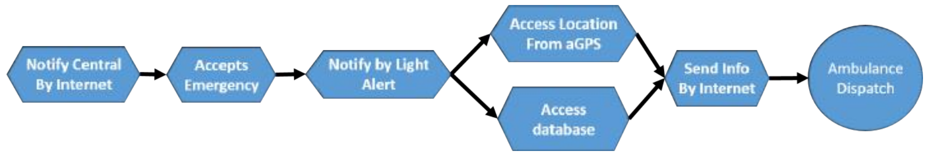


Figure 5: Workflow output for context [$c4$, $c5$, $c6$, $c7$, $c10$, $c11$]

In this situation, the algorithm would analyse the Pragmatic GM prioritizing the reliability quality constraint. It would then suggest the plan depicted in Figure 5. We also present, in Figure 6, the original Pragmatic GM with the chosen tasks highlighted to depict the adherence of the chosen tasks to the current context and goal annotations.

To comprehend the reasoning performed in choosing this approach, we must first identify the alternatives available at such time. To facilitate the reader's understanding of this example, all the tasks are considered to have the same reliability level, let's say 99% reliability.

So first, it considers the root goal and the fact that it is an AND-decomposition with linear sequencing of tasks. Then it invokes the PPA algorithm recursively for goal $G2$, then a new recursive call for $G6$, and yet another for $G11$. At $G11$ the recursive calls made over tasks $T1$ and $T2$ return a plan consisting of task $T2$ and a `null` plan, since $T1$ is inapplicable under the considered context. $G11$ then chooses such plan and returns this to $G6$. $G6$ now invokes the PPA algorithm at $G12$, which checks and sees that $G16$ is unachievable it is not applicable at the current context. Thus it returns a plan consisting of $T3$.

Now that $G6$ has a complete plan for $G11$ ($T2; T3$), it invokes the PPA algorithm on $G7$. $G7$ execution returns a plan consisting of ($T7\#(T5\#T6)\#T8$). Considering the annotations in Table 2, the observed reliability of these two alternative plans for $G2$ would be 0.9801 for ($T2; T3$) and 0.96059601 for plan ($T7\#(T5\#T6)\#T8$). Since the PPA algorithm is prioritizing the reliability QoS constraint, the plan ($T2; T3$) is returned as the best alternative for $G2$ and the algorithm proceeds in similar way for goals $G3$, $G4$ and $G5$, providing the plan ($T2; T3; T11; (T18\#T19); D1$) - depicted in Figure 5 - as a valid alternative for this situation.

From Figure 6, we can see that such workflow respects all the annotations' rules in the model: it contains tasks from $G2$, $G3$, $G4$ and $G5$, sequentially linked between them; it contains tasks from $G6$ but not $G7$ as defined in $G2$ alternative annotation; it contains $T3$ because $G16$ is unachievable given that $C2$ is not active effectively deeming $G16$ unachievable. From the four $G3$ dependencies, the only ones applicable for the patient's context were $T10$ and $T11$, where the algorithm opted for $T11$, which provided better Composite QoS levels. $G4$ annotation required that $G9$ subtree tasks ("Access location from aGPS" and "Access Database") were performed before sending info over the internet. Also, since $G14$ was achievable, the *else* branch for $G9$ was not executed because G14 was achievable. Finally, the $G10$ delegation was performed and the root goal achieved correctly.

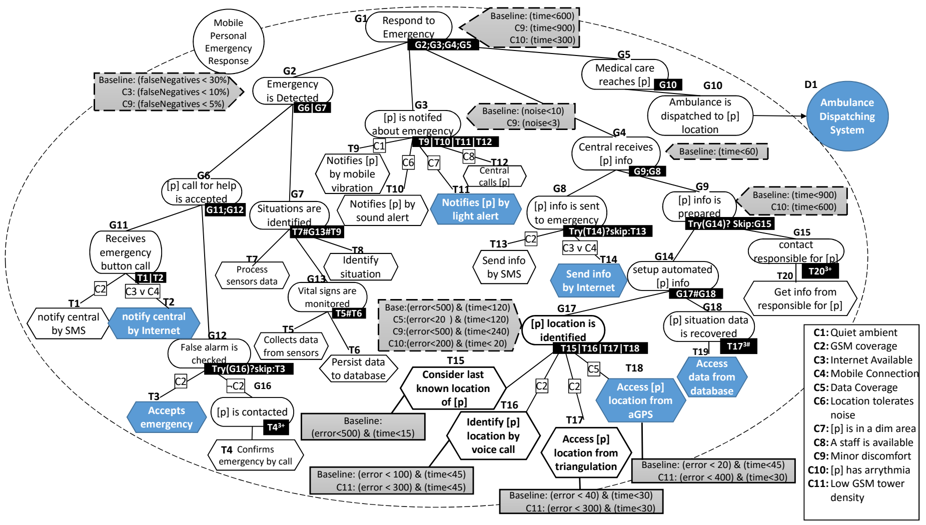

Mobile Personal Emergency Response
G1 Respond to Emergency
G2;G3;G4;G5
Baseline: (time<600) C9: (time<900) C10: (time<300)
Baseline: (falseNegatives < 30%) C3: (falseNegatives < 10%) C9: (falseNegatives < 5%)
G2 Emergency is Detected
G6|G7
G3 [p] is notifed about emergency
T9|T10|T11|T12
Baseline: (noise<10) C9: (noise<3)
G5 Medical care reaches [p]
G10
G10 Ambulance is dispatched to [p] location
D1 Ambulance Dispatching System
G4 Central receives [p] info
G9;G8
Baseline: (time<60)
T9 Notifies [p] by mobile vibration
T10 Notifies [p] by sound alert
T11 Notifies [p] by light alert
T12 Central calls [p]
C1 C6 C7 C8
G6 [p] call for help is accepted
G11;G12
G7 Situations are identified
T7#G13#T9
G8 [p] info is sent to emergency
Try(T14)?skip:T13
G9 [p] info is prepared
Try(G14)? Skip:G15
Baseline: (time<900) C10: (time<600)
G11 Receives emergency button call
T1|T2
C2 C3 v C4
T1 notify central by SMS
T2 notify central by Internet
T7 Process sensors data
T8 Identify situation
G13 Vital signs are monitored
T5#T6
T13 Send info by SMS
T14 Send info by Internet
G14 setup automated [p] info
G17#G18
G15 contact responsible for [p]
T20 Get info from responsible for [p]
T20³⁺
T5 Collects data from sensors
T6 Persist data to database
Base:(error<500) & (time<120) C5:(error<20) & (time<120) C9:(error<500) & (time<240) C10:(error<200) & (time< 20)
G17 [p] location is identified
T15|T16|T17|T18
G18 [p] situation data is recovered
T17³⁺
T19 Access data from database
G12 False alarm is checked
Try(G16)?skip:T3
T3 Accepts emergency
G16 [p] is contacted
¬C2
T4³⁺
T4 Confirms emergency by call
T15 Consider last known location of [p]
Baseline: (error<500) & (time<15)
T16 Identify [p] location by voice call
Baseline: (error < 100) & (time<45) C11: (error < 300) & (time<45)
T17 Access [p] location from triangulation
Baseline: (error < 40) & (time<30) C11: (error < 300) & (time<30)
T18 Access [p] location from aGPS
Baseline: (error < 20) & (time<45) C11: (error < 400) & (time<30)
C1: Quiet ambient
C2: GSM coverage
C3: Internet Available
C4: Mobile Connection
C5: Data Coverage
C6: Location tolerates noise
C7: [p] is in a dim area
C8: A staff is available
C9: Minor discomfort
C10: [p] has arrythmia
C11: Low GSM tower density

Figure 6: MPERS model with tasks chosen by the PPA algorithm highlighted in blue

5. Pragmatic Model and Achievability Algorithm Evaluation

In this section, we experimented on the algorithm to find out its effectiveness and to evaluate its usefulness in finding achievable plans and its scalability as to assert its practical application in large scenarios, both at design-time and runtime using the Goal-Question-Metric evaluation methodology [11]. GQM is a goal-oriented approach used throughout software engineering to evaluate products and software processes. It assumes that any data gathering must be based on an explicitly documented logical foundation which may be either a goal or an objective.

5.1. GQM Validation Plan

GQM's first step is to define high-level evaluation goals. For each goal, a plan consisting of a series of quantifiable questions is devised to specify the necessary measures for duly assessing the evaluation [11]. These questions identify the necessary information to achieve the goals while the metrics define the operational data to be collected to answer each question.

In such a methodology, we started up by setting the three main goals of our evaluation : (I) verifying the usability of the model for the MPERS case study at runtime; (II) verify that algorithm's performance allows it to also be used also at runtime for larger models and; (3) define whether the model can be used at design-time to pinpoint context sets in which the model may be, per design, unachievable. These three goals were the foundation on which the GQM validation plan was based.

Goal 1: PPA's runtime usage capability in the MPERS case study	
Question	Metric
1.1 How long would it take for the PPA algorithm to come up with a plan for the MPERS case study?	Execution time
1.2 How reliable are the plans provided by PPA for the MPERS model?	% of correct answers

Goal 2: PPA's runtime usage capability for larger models	
Question	Metric
2.1 How does the PPA algorithm scale over the amount of goals in the model?	Execution time
2.2 How does the PPA algorithm scale over the amount of contexts in the model?	Execution time

Goal 3: Algorithm's design-time usage capability	
Question	Metric
3.1 How long would it take to cover all context sets for increasingly large models?	Time elapsed

Table 3: GQM devised plan

5.2. Experiment Setup

The experiment setup consisted in evaluating the Pragmatic model and the algorithm's capability to support design-time and runtime usage. These parts and their evaluations were engineered to provide the metrics demanded by the GQM plan (Table 3).

The first evaluation goal is directed towards the MPERS case study and aims at a more comprehensive and detailed evaluation and it is intended as a "proof of concept". On this front we have evaluated the time to produce an execution plan and the reliability of the plans provided evaluation. In this evaluations, we have used the MPERS Pragmatic Goal Model as the input for the evaluation tool and measured the time for producing a plan and its adherence to the model's restrictions.

The second evaluation goal concerns itself with the scalability of the proposed model and its applicability on larger models, both in terms of goals and contexts amount. For this evaluation we have generated models with varying amounts of goals and tasks using each one the possible annotations to evaluate the impact on using such annotations - when compared to one another - and to determine the worst case scenario. This is meant to ascertain the planning overhead on the model. This information also gives designers a baseline on which to base their decision on the applicability of the model to perform runtime decisions.

Finally, the third evaluation front is concerned with the usability of the algorithm as a design-time tool to pinpoint scenarios in the Pragmatic GM which may be unachievable by design, given the goal interpretations, tasks' quality levels provided under specific contexts. For this goal, we have evaluated the time spent to sweep all context combinations in randomized models with 10 contexts and sizes ranging from 500 to 6,000 nodes.

The PPA algorithm from Figure 1 was implemented[3] using Java Oracle JDK 1.8.0_92 and all evaluation tests were performed on a Dell Inspiron 15r SE notebook equipped with a Intel Core i7 processor, 8GB RAM running Ubuntu 16.04, 64 bits and kernel 3.16.0-29-generic.

All experiments to evaluate the correctness and performance of the algorithm were implemented as automated tests under Java's JUnit framework. This guarantees that the evaluation is both effortless and repeatable.

5.3. Goal 1: PPA's runtime usage capability in the MPERS case study

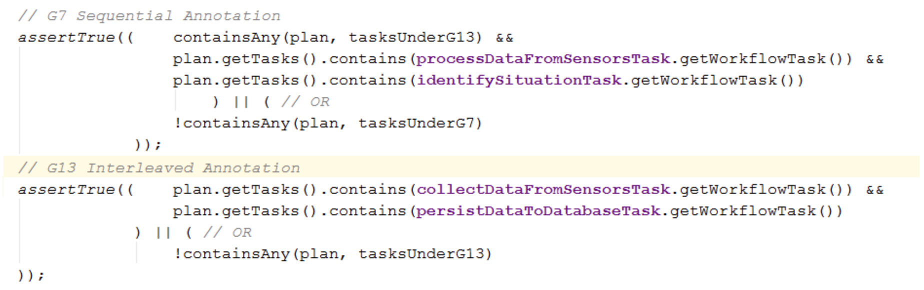
For this goal, we evaluated the average time to produce an answer in the presented MPERS scenario (Figure 1) and the reliability of the engineered plans.

Question 1.1: How long does it take for the PPA algorithm to come up with an execution plan for the MPERS case study?. To evaluate the time for the algorithm execution on the CGM of Figure 1, we executed 100 iterations of the algorithm for each possible context set. The results showed that the algorithm took, in average, less than 1 ms to be executed. This evaluation was performed by executing the algorithm over the MPERS model 100 times for each possible context set, summing up to a total of 409600 repetitions. The average time for coming up with a plan for the MPERS case study was 0.34 ± 0.09 ms. Out of the 409600 measurements, there were however 62 outliers (0.015%) with execution times of more than 10 ms. These outliers points are still under a second and are likely due to background tasks in the machine, which was not dedicated.

Question 1.2: How reliable are the plans provided by PPA for the MPERS model?. To validate the correctness of the plans we have, for each context set for the MPERS, identified all the inapplicable tasks, all restrictions imposed by the goal annotations' and the composite QoS constraints. Then, we formally coded these restrictions through 42 assertion statements, similar to the ones presented in Figure 7, throughout the test so as to guarantee that none of the restrictions were disrespected in any of the outputted plans. Finally this test was performed for every possible context set and none of the coded restrictions were infringed.

Analysis of the results. With regard to the MPERS case study, the proposed algorithm have been shown to be efficient. The planning stage for this scenario took around 3 milliseconds to be computed and none of the 4096 executions, effectively sweeping all the possible context sets, triggered any of the several assertion statements introduced in the tests. It was certainly expected - given

[3]Source code, evaluation mechanisms, and complete result sets are available at `https://github.com/felps/PragmaticGoals/tree/RuntimeGoalModel`. Accessed on 2016/07/30



```
// G7 Sequential Annotation
assertTrue((    containsAny(plan, tasksUnderG13) &&
                plan.getTasks().contains(processDataFromSensorsTask.getWorkflowTask()) &&
                plan.getTasks().contains(identifySituationTask.getWorkflowTask())
                    ) || ( // OR
                !containsAny(plan, tasksUnderG7)
            ));
// G13 Interleaved Annotation
assertTrue((    plan.getTasks().contains(collectDataFromSensorsTask.getWorkflowTask()) &&
                plan.getTasks().contains(persistDataToDatabaseTask.getWorkflowTask())
            ) || ( // OR
                !containsAny(plan, tasksUnderG13)
));
```

Figure 7: Assertions to verify the generated plan's adherence to the annotations' semantics

the deterministic nature of the algorithm - that all the responses were valid. However the level of efficiency in solving this problem was remarkable.

5.4. Goal 2: PPA's runtime usage capability for larger models

The second goal from the GQM validation plan aims at defining if the model and algorithm proposed are suitable for usage in larger models. To uncover some information regarding this objective, we have stated two questions on the scalability of our proposal: how does it scale over the amount of goals in a model and how does it scales over the amount of contexts in a model.

This investigation is important considering the objective of using this model and this algorithm to engineer - at runtime - an execution plan able to achieve the CGM's root goal under the current context and current QoS constraints. To accomplish this goal - and this is a pragmatic goal itself - it is not sufficient to engineer the plan but it needs to done in a reasonable amount of time so that it will not seriously impact the response time.

Question 2.1 and 2.2: How does the PPA algorithm scale over the amount of goals and contexts in the model?. To answer questions 2.1 and 2.2, we have performed a scalability analysis on Pragmatic GMs of different sizes, in terms of goals and contexts amounts, for each one of the possible goal annotations.

For each of the available annotations, we engineered and performed a series of evaluations. A pragmatic model with size ranging from 500 to 10.000 (in steps of 500 nodes) and contexts ranging from 1 to 20 containing goals annotated with the selected annotation would be generated. Then, for each of these models, we executed the `isAchievable` method 50 times and measured the total execution time. Finally, the test outputted the average execution time per method invocation. The resulting average times are presented - one for each annotation - in Figure 8.

In these graphs, the x- and z-axes represent the model-size and the amount of contexts used in each execution respectively, *i.e*, the independent variables. The y-axis represent the time for engineering an execution plan, *i.e*, the experiment's dependent variable. In all six experiments, the average time for engineering the plan did not go over 5 seconds. Also, in the plotted graphs we are able to glimpse into the expected complexity of $O(m * n)$ over the model size m (x-axis) and over context sizes n (z-axis).

Analysis of the results. In general, Figure 8 shows a good behaviour of the algorithm. The growth ratio for the algorithm's execution time grows linearly over the amount of goals and contexts. None of the annotations had issues in elaborating a plan on larger models, although their performance varied in different degrees. The behaviour of the Sequential annotation was the worst of them all with an execution time of a little under 4.5 seconds for models with 10,000 nodes and 10 contexts. Still this was considered an affordable execution time for the planning stage of such large scenario in terms of goals, tasks and contexts.

5.5. Goal 3: Algorithm's pinpointing unachievable context sets capability

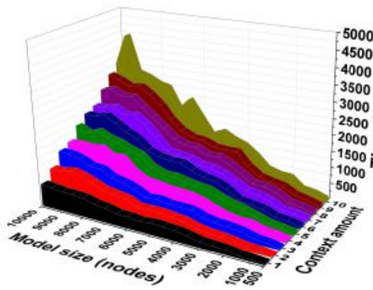
Finally, the last goal of the evaluation is to show the applicability of this algorithm to benefit software designers. More specifically, we want to know if it can answer the following question: "Given that the system has been modelled as a Pragmatic GM, is it possible to use this algorithm pinpoint context sets in which such system would inherently be unable to reach its goals?"

To deal with this we have implemented one last test which would generate 600 random models varying from 100 to 6000 nodes with a fixed set of 10 contexts. Then, the algorithm was executed for every one of the 2^{10} possible context sets and the time spent to do so was measured. The results are now presented in Figure 9, with each point representing the average time to sweep all the context sets in 10 randomized models.

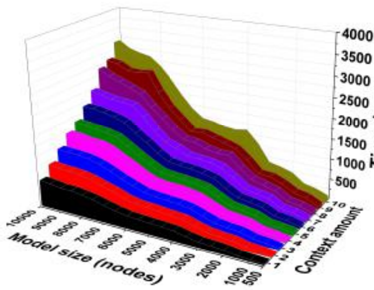
Analysis of the results. As shown in Figure 9, on smaller models - up to 300 goals - the algorithm was able to fully analyze the 32768 context sets within the 10 seconds deadline. On larger models - up to 5000 goals - the algorithm was able to analyse around 40% of the combinations. Even at the limit, with models of 10000 nodes, it was able to cover more than 25% of the possible combinations. This result suggests that even for models with up to 10000 goals and 20 contexts the complete analysis can be performed within a minute.

6. Related Work

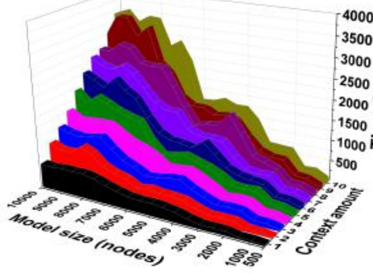
Previous work have tackled similar problems in relation to the dynamic requirements for adaptive systems and system planning for goal achievement but to the best of our knowledge none has dealt with the dynamic context-dependent interpretation of requirements and, in particular, of goals. Relevant approaches include the work of Sebastiani et al., who map the Goal-Model satisfiability problem into a propositional satisfiability (SAT) problem [12]; Souza and Mylopoulos on Awareness Requirement Goals, that define quality objectives for other goals [10, 13]; Baresi and Pasquale on Live Goals: goals whose individual behavior change in order to pursue some qualitative objective and bring the system back to a consistent state [14][15]; Dalpiaz et al. on declarative goals:



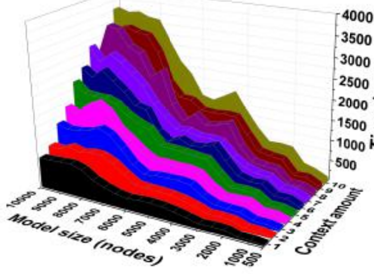
(a) Sequential



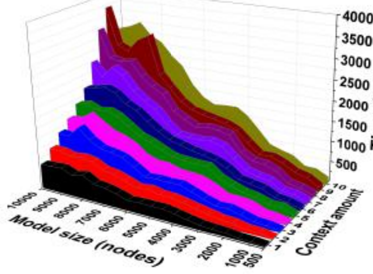
(b) Parallel



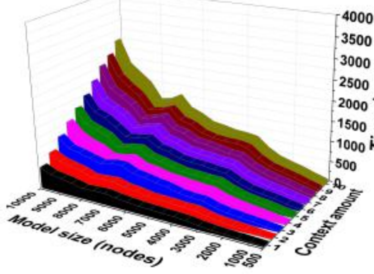
(c) Sequential Iterative



(d) Parallel iterative



(e) Alternative



(f) Try

Figure 8: Algorithm's scalability over the model size, in number of nodes, and context amount

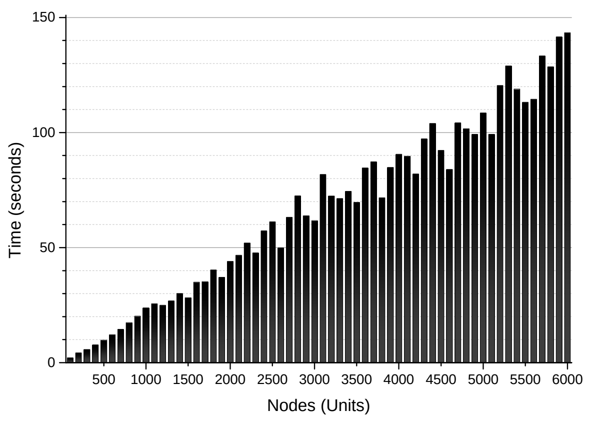


Figure 9: Time spent to run PPA algorithm for all the possible context sets (average execution time over 10 different models)

separate goals whose achievement depends on the effects of its refinements on the environment [16]; and the RELAX framework which provides a more rigorous treatment of requirements explicitly related to self-adaptivity [17]. With regard to the planning stage for achievement of goals, Horkoff and Yu [18] and Letier and Van Lamsweerde [19] have both proposed approaches to dealing with the achievement of non-functional goals. Horkoff and Yu [18] introduced an interactive evaluation procedure that propagates backward from high-level target goals, allowing users to ask “Is this possible?” questions and Letier and Van Lamsweerde [19] presented techniques for specifying partial degrees of goal satisfaction and for quantifying the impact of alternative system designs on the degree of goal satisfaction.

In essence, related work focus on goal-driven adaptation as a system- or model-wise problem. We argue that the notion of pragmatic goals, introduced in our approach, can enrich the rationale of adaptation proposed in those approaches by treating system adaptation in a contextual case-to-case situation.

The Pragmatic Goals’ concept differ from the presented work and from traditional softgoals - which do not have clearcut satisfaction criteria[10] - because, unlike [16], [19], [10] and [13], we consider the pragmatic aspect, *i.e.*, the quality objective as an inseparable part from the goal itself: the mere completion of one or all refinements is not enough to achieve a goal, there may be clients’ expectations/demands which must be met and which, differently from [17] and [18], is itself context-dependent rather than static. We also deal with the identification and reasoning in a fully automated way, without the need for expert judgement [18] during the decision process. This is also done *a priori* and over the CGM as a whole in an effort to keep the system in a consistent state instead of identifying and correcting inconsistent states like [14] and [15]. Regarding the algorithm itself, our approach achieves linear complexity over the amount of goals in the tree, by the means of a simplifying assumption: that there are no contributions or denials between different goals. This enabled us to consider the Pragmatic GM as a tree rather then a generic graph.

Thus, the novelty of our work in comparison to other approaches in requirements-driven adaptation is twofold: (1) The definition of pragmatic goals which means that the satisfaction criteria for goals is context-dependent. (2) The development and implementation of an automated reasoning that can deterministically answer whether the goal is pragmatically achievable and, if it is, point out an execution plan that is likely to achieve it under the current context.

7. Conclusions and Future Work

In this paper we have proposed the utilization of a Pragmatic CGM in which the goals' context-dependent interpretation is an integral part of the model. We have also shown why hard goals and softgoals are not enough to grasp some of the real-world peculiarities and context-dependent goal interpretations. We have also extended the model with Dalpiaz's goal annotations in order to properly specify the allowable system's behaviours at runtime.

We defined the pragmatic goals' achievability property: whether there is any allowable execution plan that fulfils the goal's interpretation under a given context. We also proposed, and implemented the PPA algorithm. This algorithm is able to decide on the achievability of a goal and, if so, engineer an execution plan in compliance with all the goal annotations to do so.

Finally, we thoroughly evaluated the model and the algorithm. The evaluation was performed in three different fronts, all of which rendered very appealing results. On the first front, we evaluated, for the MPERS case study, the algorithm's performance (execution time of less than 4ms in average) and reliability (no errors detected over more than 4000 executions). On the second front we have performed a scalability analysis on the algorithm running with every possible annotation and shown that in all the situations the PPA algorithm scales linearly over the amount of goals and context amount. For the third front, we have evaluated the capability of using the algorithm to pinpoint at design-time scenarios in which the model's root goal is unachievable by design in order to alert the GM designer of this potential flaw. The results for the third evaluation front showed that for models with 6000 goals and 10 contexts, the full scope of context sets could be swept in about 2'30" (two and a half minutes).

For future work, we plan to: (1) compare the performance offered by the PPA algorithm to that achieved by SAT-solvers; (2) study the impact of considering more than one composite QoS constraint and; (3) integrate the PPA algorithm into a service-oriented engine to perform context-aware service compositions

Acknowldegements

This work has been partially funded by CNPq, the EU FP7 Marie Curie Programme through the SOCIAD project and also by Finatec/UnB under call 04/2015, CEPE resolution number 171/2006 and CAPES/PROCAD-grant number 183794.

References

[1] E. Yu, J. Mylopoulos, Why goal-oriented requirements engineering, in: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality, 1998, pp. 15–22.

[2] R. Ali, F. Dalpiaz, P. Giorgini, A goal-based framework for contextual requirements modeling and analysis, Requirements Engineering 15 (2010) 439–458.

[3] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia, J. Mylopoulos, Tropos: An agent-oriented software development methodology, Autonomous Agents and Multi-Agent Systems 8 (2004) 203–236.

[4] J. Castro, M. Kolp, J. Mylopoulos, Towards requirements-driven information systems engineering: the *Tropos* project, Information systems 27 (2002) 365–389.

[5] E. Yu, Modelling strategic relationships for process reengineering, Social Modeling for Requirements Engineering 11 (2011) 2011.

[6] F. Dalpiaz, A. Borgida, J. Horkoff, J. Mylopoulos, Runtime goal models: Keynote, in: IEEE 7th International Conference on Research Challenges in Information Science (RCIS), IEEE, 2013, pp. 1–11.

[7] F. P. Guimarães, G. N. Rodrigues, D. M. Batista, R. Ali, Pragmatic requirements for adaptive systems: A goal-driven modeling and analysis approach, in: P. Johannesson, M. Lee, S. W. Liddle, A. L. Opdahl, O. P. López (Eds.), Conceptual Modeling - 34th International Conference, ER 2015, Stockholm, Sweden, October 19-22, 2015, Proceedings, volume 9381 of *Lecture Notes in Computer Science*, Springer, 2015, pp. 50–64.

[8] A. Finkelstein, A. Savigni, A framework for requirements engineering for context-aware services, STRAW 01 (2001).

[9] D. F. Mendonça, R. Ali, G. N. Rodrigues, Modelling and analysing contextual failures for dependability requirements, in: Proceedings of the 9th SEAMS, ACM, New York, NY, USA, 2014, pp. 55–64.

[10] V. E. Silva Souza, A. Lapouchnian, W. N. Robinson, J. Mylopoulos, Awareness requirements for adaptive systems, in: Proceeding of the 6th SEAMS, ACM Press, New York, New York, USA, 2011, p. 60.

[11] V. R. Basili, G. Caldiera, H. D. Rombach, The goal question metric approach, in: Encyclopedia of Software Engineering, Wiley, 1994.

[12] R. Sebastiani, P. Giorgini, J. Mylopoulos, Simple and minimum-cost satisfiability for goal models, in: Advanced Information Systems Engineering, Springer, 2004, pp. 20–35.

[13] V. E. S. Souza, J. Mylopoulos, From awareness requirements to adaptive systems: A control-theoretic approach, 2011 2nd International Workshop on Requirements@Run.Time (2011) 9–15.

[14] L. Baresi, L. Pasquale, Adaptive Goals for Self-Adaptive Service Compositions, 2010 IEEE International Conference on Web Services (2010) 353–360.

[15] L. Baresi, L. Pasquale, Live goals for adaptive service compositions, in: Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems - SEAMS '10, 2010, pp. 114–123.

[16] F. Dalpiaz, P. Giorgini, J. Mylopoulos, Adaptive socio-technical systems: a requirements-based approach, Requirements Engineering 18 (2011) 1–24.

[17] J. Whittle, P. Sawyer, N. Bencomo, B. H. Cheng, J.-M. Bruel, Relax: Incorporating uncertainty into the specification of self-adaptive systems, in: 17th IEEE RE, IEEE, 2009, pp. 79–88.

[18] J. Horkoff, E. Yu, Finding solutions in goal models: an interactive backward reasoning approach, in: International Conference on Conceptual Modeling, Springer, 2010, pp. 59–75.

[19] E. Letier, A. Van Lamsweerde, Reasoning about partial goal satisfaction for requirements and design engineering, in: ACM SIGSOFT Software Engineering Notes, volume 29, ACM, 2004, pp. 53–62.