Partial and Full Goal Satisfaction in the MUSA Middleware

Massimo Cossentino, Luca Sabatucci, and Salvatore Lopes

National Research Council
Istituto di Calcolo e Reti ad Alte Prestazioni (ICAR-CNR)
Palermo, Italy
{name.surname}@icar.cnr.it

Abstract. Classical goal-based reasoning frameworks for agents suppose goals are either achieved fully or not achieved at all: unless achieved completely, the agents have failed to address them. This behavior is different from how people do and therefore is far from real-world scenarios: in every moment a goal has reached a certain level of satisfaction.
This work proposes to extend the classical boolean definition of goal achievement by adopting a novel approach, the Distance to Goal Satisfaction, a metric to measure the distance to the full satisfaction of a logic formula.
In this paper we defined and implemented this metric; subsequently, we extended MUSA, a self-adaptive middleware used to engineer a heterogeneous range of applications. This extension allows solving real situations in which the full achievement represented a limitation.

Keywords: Partial Goal Satisfaction · Metric · Multi-Agent System.

1 Introduction

Exploring alternative options is at the heart of rational and self-adaptive behavior. In our applications, agents always try to find plans to fully achieve their goals. However, this is different from how people do. A person often adopts plans for partially achieving her/his goals.

Recently, a research direction claims agents should be able of reaching a better approximation of scenarios arising from the real world. Indeed, Zhou and Chen argue that reasoning with a partial satisfaction of goals is an essential issue for achieving this result [19]. For instance, GoalMorph [18] is a framework for context-aware goal transformation to facilitate fault-tolerant service composition. It achieves planning with partial goal satisfaction by reformulating failed goals into problems that can be solved by the planner.

Two main trends exist in dealing with partial goal satisfaction. The former studies partial goal satisfaction by looking at the goal model as a whole (whether it is a goal hierarchy or a goal graph). Conversely, the second trend focuses on the single goal entity, by entering into the details of how it is represented.

Concerning the first direction, Letier and Van Lamsweerde [6] present a technique for quantifying the impact of alternative system designs on the degree of goal satisfaction. The approach consists of enriching goal models with a probabilistic layer for reasoning about partial satisfaction. Within such models, non-functional goals are specified in a precise, probabilistic way; their specification is interpreted in terms of application-specific measures; the impact of alternative goal refinements is evaluated in terms of refinement equations over random variables involved in the system's functional goals.

This paper focuses on the second research direction, i.e., studying the partial satisfaction of a single goal.

An interesting contribution comes from Zhou et al. [20]. They define the partial implication to capture partial satisfaction relationship between two propositional formulas. According to their propositional language, a formula like $x \wedge z$ partially implies $x \wedge y$. Indeed, x, which can be considered as a part of $x \wedge y$, is a logical consequence of $x \wedge z$.

Van Riemsdijk and Yorke-Smith [17] propose a higher-level framework based on metric functions that represent the progress towards achieving a goal. Progress appraisal is the capability of an agent to assess which part of a goal it has already achieved. However, the framework is abstract: authors do not further detail how a partial ordering function may be implemented. According to the authors, it can be based on a wide range of either domain-independent parameters, such as time, utility, number of subgoals, and domain-dependent metrics.

Thangarajah et al. [15,16] adopt an approach base on resource analysis to provide a BDI agent with a quantified measure of effort with respect to the amount of resources consumed in executing a goal with respect to the total resource required. This kind of approach is strictly domain-dependent, and it may be applied only when there is a clear link between goals and resources.

This paper presents a novel approach for enabling agents to reason on partial satisfaction of single goals (where a goal condition is given in predicate logic). The approach is based on the definition of a Distance to Goal Satisfaction that measures the distance to the full satisfaction of a logic formula, given the current state of the world. To practically implement this metric we adopt an analogy between the logic formulas specifying goal satisfaction and an equivalent electric circuit.

The approach has been modeled to be consistent with the classical logic semantics. It is essentially domain-independent but easily extensible for considering other domain-dependent parameters. An interesting side effect of this approach is that the metric allows to calculate the 'cost for the full satisfaction', thus providing a powerful tool for selecting among alternative strategies.

We applied this metric in the context of a self-adaptive middleware (MUSA) in a heterogeneous range of application contexts (from IoT to maritime IT applications).

The remaining of the paper is structured as follows: Section 2 presents the middleware for self-adaptive systems and how we encountered the need for dealing with different degrees of goal satisfaction. Section 3 provides definitions for the core concepts of the proposed approach. Section 4 provides technical details about how MUSA has been extended for supporting the new metric. Section 5 discusses some of the key aspects of the approach, and finally, Section 6 summarizes the approach and reports some future works.

2 Motivation

MUSA (Middleware for User-driven Self-Adaptation) [11,13] is a middleware for composing and orchestrating distributed services according to unanticipated and dynamic user needs. It is a platform in which 1) virtual enterprises can deploy capabilities that wrap real services, completing them with a semantic layer for their automatic composition; 2) analysts and users can inject their goals for requesting a specific outcome. Under the hypothesis that both goals and capabilities refer to the same ontology, agents of the system can compose available services into plans for addressing user's requests.

The enablers of the MUSA core vision are: i) representing *what* and *how* as run-time artifacts the system may reason on (respectively goals and capabilities); ii) a reasoning system for connecting capabilities to goals; iii) finally a common grounding semantic, represented with some formalism.

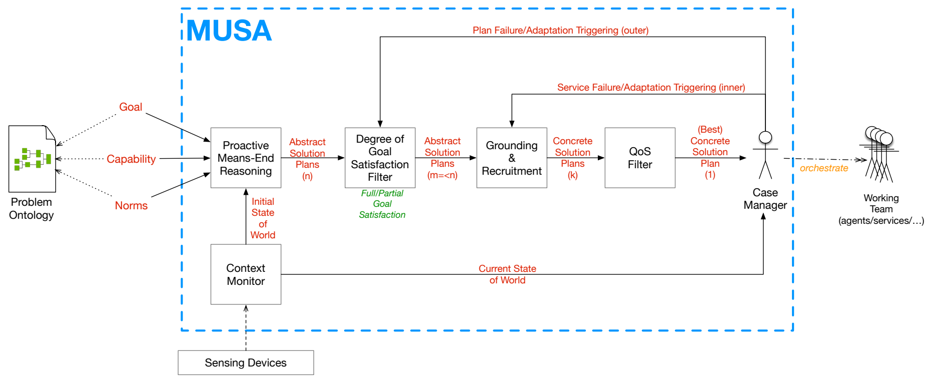

MUSA
Plan Failure/Adaptation Triggering (outer)
Service Failure/Adaptation Triggering (inner)
Goal
Capability
Norms
Problem Ontology
Proactive Means-End Reasoning
Abstract Solution Plans (n)
Degree of Goal Satisfaction Filter
Full/Partial Goal Satisfaction
Abstract Solution Plans (m=<n)
Grounding & Recruitment
Concrete Solution Plans (k)
QoS Filter
(Best) Concrete Solution Plan (1)
Case Manager
orchestrate
Working Team (agents/services/...)
Initial State of World
Context Monitor
Current State of World
Sensing Devices

Fig. 1. Main components of MUSA.

The main abstractions in MUSA are that of *Capability*, which represents the knowledge about what the system can do, and that of *Goals*, representing the user's expectations. In MUSA, a user can describe the desired functionality via a high-level goal-oriented description language [14]. At run-time, once a goal model is specified, it may be injected in the system.

The concept of capability comes from the AI (planning actions [3]) and service-oriented architecture (micro-services [7]). MUSA presents this dual nature by adopting a separation of the abstract capability – a description of the effect of an action that can be performed – and the concrete capability – a small, independent, composable unit of computation that produces some concrete service.

Figure 1 illustrates the logical schema of MUSA with its macro components. The proactive means-end reasoning [11] is the main component, which aggregates abstract capabilities for deducing possible solutions, each one that can guarantee the desired goal state. The set of solutions are, at this stage still abstract (i.e., not bounded to real services), but this allows evaluating to which extent they will satisfy the goals. This way, MUSA can apply a filter to the valid solutions in order to select the best ones of those above a prescribed threshold (according to the application context in which it operates).

After discarding the non-satisfying solutions, MUSA creates a temporary link between abstract capabilities and real services via the Grounding and Recruitment module (the link is temporary because it can change at run-time for self-adaptation purposes), thus generating concrete solutions. These can be finally evaluated from other (non-functional) perspectives, for instance evaluating the fitness toward a QoS value.

MUSA has been employed in a heterogeneous set of application contexts. A selection of these is shortly illustrated in Table 1.

Table 1. Summary of research projects and case studies where the MUSA middleware has been employed between 2013 and 2018.

Achronym	**Type**	**App. Name**	**Description**
IDS	Research Project	Innovative Document Sharing	The aim has been to realize a prototype of a new generation of a digital document solution that overcomes current operating limits of the common market solutions. MUSA has been adopted for managing and balancing human operations for enacting a digital document solution in a SME.
Smart Grid	Research Project	Cloud Mashup	MUSA has been adopted as mashup engine, i.e., for self-configuring ad-hoc orchestration of existing services in order to address run-time business.
Smart Travel	Case Study	Travel Agency System	MUSA provides the planning engine that creates a travel-pack as the composition of several heterogeneous travel services. Traveler's goals drive the planning activity.
UPA4SAR	Research Project	IoT and robotic home assistance	MUSA is the core for implementing the behavior of smart devices deployed in a simulated environment.
SPS Reconfiguration	Case Study	Electric Management	MUSA is the core for the algorithm for run-time reconfiguration of the shipboard power system in case of malfunctioning.

During its usage designers and developers have pointed out the need for extending the flexibility of MUSA goal definition in order to deal with many degrees of goal satisfaction. To clarify this concept, we report some examples.

Example 1: in the Smart Travel application [10], the tourist is able to configure its travel request by injecting a set of user's goals. For instance (s)he can desire "to book a 5-star hotel in the center of *TownX* with a pool". Sometimes too constrained goals produced zero results in the travel configuration so that the tourist has to manually change its goals with no information about what part of the goal was unsatisfiable. To solve this problem, the system should include a mechanism to relax the condition, thus finding solutions that partially solve the problem (for instance either a hotel with a pool but out of town, or in downtown with no pool).

Example 2: the Shipboard Power System (SPS) reconfiguration problem consists in acting on electric switches for changing power flows and ensuring ship's components are powered on when necessary. In the MUSA application [12], the different ship's missions are coded by using a set of goals (with different priorities). This proved to be an interesting case study for system self-adaptation[1]. An example of a goal is "the navigation system, the communication system, and the radar should be powered". In case of severe malfunctioning, when the main generators are off, the auxiliary generators are switched on. However, they can not produce enough power for the three components work properly. For such a reason MUSA should produce plans for partially restoring the energy on board, even if not all goals are fully satisfied.

Example 3: the UPA4SAR project will deliver an IoT based robotic home assistance to help elderly patients to remain at home, thus avoiding hospitalization or admission to long-term care institutions. In MUSA application [8] this high-level goal is decomposed in a set of subgoals for daily monitoring of patient activities and providing services for helping individuals to live with greater independence and promoting the optimal level of well-being. One of the subgoals of the system is something like "in the afternoon, propose some entertainment activity, as long as there is time". Actually, the goal is often unfeasible because there are (frequent) situations in which an elderly patient changes its mood in the meanwhile the system is proposing its activities. In these situations, the system cannot complete its goals, (because the activity does not terminate). In a boolean goal satisfaction approach, the system will declare a failure, whereas reasoning with partial satisfaction would be more appropriate.

Currently, the ability of MUSA to deal with goal satisfaction is inspired to the classical Kripke model, thus allowing associating a logical condition to a true-false interpretation. If W is the current state of the world and $W \models c$, then we can assert c is true; otherwise, it is false. This model is not suitable for calculating a partial degree of satisfaction when a formula is neither fully addressed nor totally unachieved. The next section introduces the background to let agents the ability to evaluate and measure partial goal satisfaction. The implementation and technical details are provided in Section 4.

3 Full and Partial Goal Satisfaction

In this section, we will provide some definitions of the most relevant concepts in the proposed approach. Let us draw a border to define our system; this is one of the most sensitive operations in many different problems. The system should include all the parts of the world that significantly participate in defining the behavior of the system itself. In the following, we will refer to discrete-time systems, and we will address successive time slots as successive steps in time.

Let be ‘k’ an arbitrary moment in time. Let us suppose our system is controlled by a set of inputs $U[k]$, where U is a vector of n elements, and k defines the time step where we consider the values of its elements. This system is supposed to be, at k, in a state $X[k]$ where X is a vector of m elements; the output of the system is designated as $Y[k]$, again a vector, of p elements.

We suppose there is a relationship among these vectors as follows:

$Y(k) = f(X(k), U(k))$

It is relevant to say we suppose to be able to observe the elements of the state vector X and the output vector Y (for instance by measuring them). An interesting consideration comes from M. Jackson, in fact, he studies the impact of environmental phenomena on the system [4], and he states that “All environment phenomena mentioned in the requirement are shared with the machine” [5]. For this reason, we can suppose that the environment is providing a part of the input vector while the remaining part is the result of some software computation.

Referring to such a model of our system, we can define the concept of goal:
Def. 1 - Goal: A goal specifies a desired condition for the output of the system, more specifically, a goal is a condition on the values of a subset of the output vector elements y_i.

This means a goal may aggregate values from different elements of the output vector by means of logic operators or mathematical formulas in order to specify the desired condition on the output of the system.

In order to better clarify that we will refer to two examples related to different systems:

1. An electric system composed of 3 loads (bulb lights), two generators providing input power to the system and three on/off switches to control the flow of power reaching the loads. In this case, the input vector is composed of the two values of voltage for the generators, the state vector of the on/off condition of each switch and the output vector of the current values in the three loads.
2. A production plant composed of two buildings: workshop and office. This plant operates 24/24 hours all days of the week (except for holidays) for the workshop part while the office is open only on working days (Monday to Friday). The system is an advanced ambient intelligence system that can control (and personalize by specific needs and preferences) ambient temperature, humidity, light level (illumination) in the different places of the plant also according to some specific tasks the painting room requires higher temperatures to dry painting quickly. The system can control air conditioning, artificial light and rolling shutters in a continuous (analog) way.

Referring to the first example, goals may specify conditions like:

- $g_1 = [I_1 > 0]$ that means the first load is powered;
- $g_2 = [I_1 > 0 \text{ AND } I_2 = 0]$ that means the first load is powered while the second one is not;
- $g_3 = [I_3 = 4 \text{ Ampere}]$ that is a more detailed specification of the expected outcome allowing for the detection of anomalies in the system such as fail-

ures or unexpected negative influences by the environment (such as a flood causing current dispersion);

Looking at the second example, some goals may be:

- $g_4 = [L_{WRoom01} = 0\ ]$ meaning that light in the workshop room 01 has to be switched off;
- $g_5 = [L_{Office101}{=}500]$ meaning that illumination in the office 101 has to be 500 Lux;
- $g_7 = [L_{WRoom01} = 0$ AND $L_{Office101}{=}0]$ meaning that office 101 and workshop room 01 lights are switched off.

Being goals applied to the values of the output vector elements, it is interesting to note that more than one instances of this vector (i.e., the combination of different values of the output elements) may verify the same goal. Such a situation is represented in Figure 2 where three different value instances of the output vector are shown (instance values of the vector are represented as uppersigned). We can suppose some goal of our system may be verified by the first two instances ($\bar{Y'}$, $\bar{Y''}$). This may happen for instance because the goal requires the element y_1 be greater than zero and the element y_2 be lesser than zero. Now let us suppose for the instance vector $\bar{Y'''}$ only the first one of these two conditions is verified. The goal is not satisfied by this set of output values but it is also obvious that the system is some way near to the desired condition.

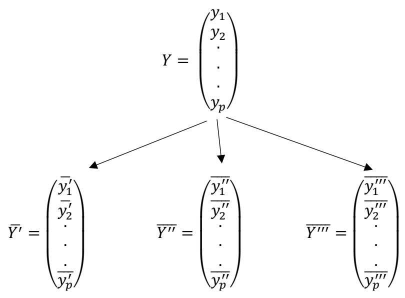

$Y = (y_1, y_2, \ldots, y_p)$
$\bar{Y'} = (\bar{y_1'}, \bar{y_2'}, \ldots, \bar{y_p'})$
$\bar{Y''} = (\bar{y_1''}, \bar{y_2''}, \ldots, \bar{y_p''})$
$\bar{Y'''} = (\bar{y_1'''}, \bar{y_2'''}, \ldots, \bar{y_p'''})$

Fig. 2. Instances of the output vector with different element values.

In order to support the evaluation of such situations, we introduce the concept of distance to goal satisfaction as reported in the definition below.

Def. 2 - Distance to Goal Satisfaction: The Distance to Goal Satisfaction (DGS) is the distance of the current output vector $\bar{Y}$ from the nearest one that satisfies goal conditions.

Many different ways may be conceived to measure this distance, we decided to measure that by considering an electric analogy, and we introduced the measure of Resistance to Satisfaction (R2S) as a metric for DGS. We consider each condition specified in the goal as a piece of an electric circuit, more specifically as a resistor. The value of this resistor will be 0 (short circuit) if the condition is true (it means the corresponding value of the output vector verifies one of the goal conditions) while the value will be R_{max} if the condition is false where is a great value of resistance representing the open circuit condition. Let us consider the $\bar{Y'''}$ vector that does not verify goal conditions. For instance, let us suppose the goal specifies one unique condition: y_1 >0. If the value of $\bar{y_1'''} > 0$ then the value of the corresponding R2S $(\bar{y_1'''}) = 0$ otherwise it will be R2S $(\bar{y_1'''})$ $= R_{max}$ If the goal condition implies the evaluation of more than one element of the output vector we compose them as a growing complexity circuit where the AND condition is represented as a series of the resistors representing the two AND variables and the OR condition is represented as a parallel of the resistors representing the two OR variables. The NOT operator is simply dealt by inverting the measure values: if the result is true, R2S=R_{max}, otherwise R2S=0. A description of the formulas and equivalent circuits is reported in Figure 3.

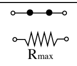
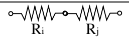
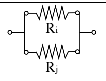
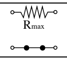
Goal	**R2S**	**Equivalent circuit**
g = φ (y_i)	If $\begin{cases} \varphi(y_i) = True \Rightarrow R2S(\varphi(y_i)) = 0 \\ \varphi(y_i) = False \Rightarrow R2S(\varphi(y_i)) = R_{max} \end{cases}$	
g = $y_i \wedge y_j$	$R2S = R_i + R_j$ where: If $\begin{cases} y_k = True \Rightarrow R_k = 0 \\ y_k = False \Rightarrow R_k = R_{max} \end{cases}$ with k=i,j	
g = $y_i \vee y_j$	$\frac{1}{R2S} = \frac{1}{R_i} + \frac{1}{R_j}$ where: If $\begin{cases} y_k = True \Rightarrow R_k = 0 \\ y_k = False \Rightarrow R_k = R_{max} \end{cases}$ with k=i,j	
g = !y_i	If $\begin{cases} \varphi(y_i) = True \Rightarrow R2S(\varphi(y_i)) = R_{max} \\ \varphi(y_i) = False \Rightarrow R2S(\varphi(y_i)) = 0 \end{cases}$	

Fig. 3. Measures of the distance to goal satisfaction for logic formulas.

Of course, if the distance to goal satisfaction for a specific situation is 0, then the goal is fully satisfied.

4 The Agent and Artifact Architecture

This section presents the current MUSA architecture derived by that illustrated in [13] in 2017. It has been realized as a multi-agent system implemented in the Jason language [2] where the environment is modeled by an independent computation layer that encapsulates functionalities the agents can access and manipulate. These entities are called artifacts and are programmed via CArtAgO [9] as Java classes.

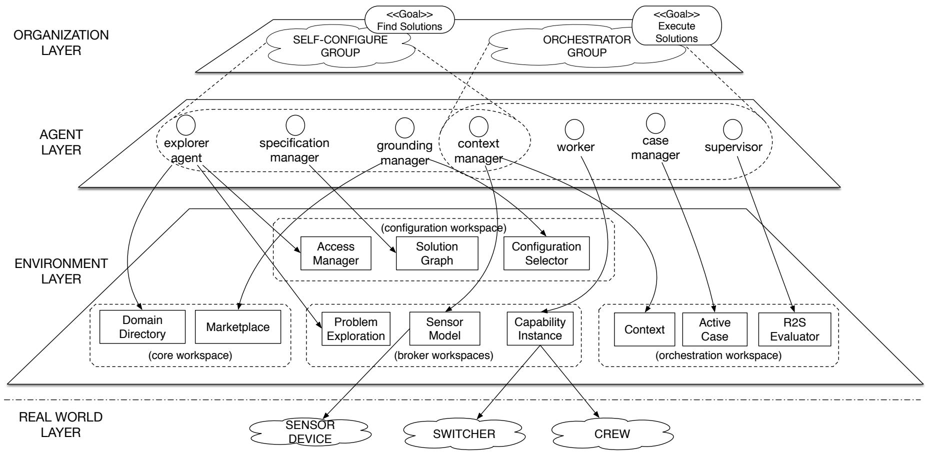


Fig. 4. The Agents and Artifacts view of the MUSA architecture.

The new architecture is shown in Figure 4 and it is decomposed in three layers (organization, agent and environment) whereas the main functionalities are organized in four workspaces: i) core, ii) configuration, iii) broker and iv) orchestration. The *core* workspace provides general facilities for the registration of agents and services. The *configuration* workspace is responsible for the self-configuration ability, providing a distributed algorithm for the discovery of solutions. The *broker* workspace is responsible for interfacing the real services available for creating and executing solutions. The orchestration workspace is responsible for granting the self-adaptive orchestration of the services that are contained in a solution. For more details about all the agents and artifacts of the solution, please refer to [13].

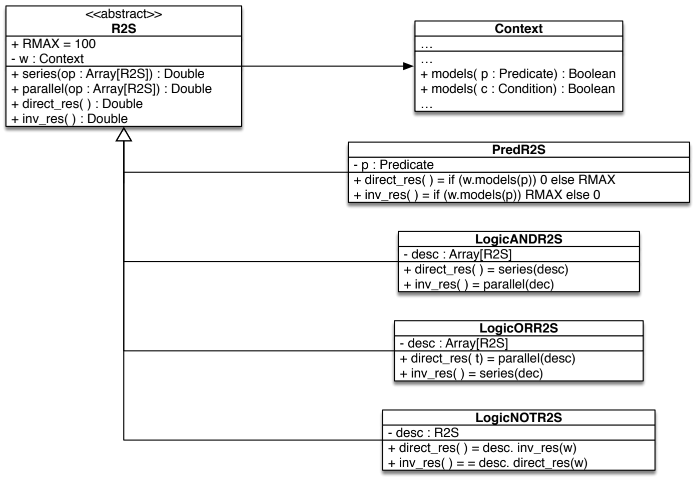
In the following, we provide more details about the supervisor agent of the orchestration workspace, who is responsible for checking at run-time the degree of satisfaction of goals.

4.1 Implementing the Metric for Partial Satisfaction

In order to provide MUSA with this feature, we will refer to the definitions provided in Section 3 where the *ResistanceToFullSatisfaction* (R2S) metric has been introduced.

Given the current state, R2S is calculated as a real number in the range $[0 - \infty]$, with the following interpretation: the lower the value of this variable, the closer the goal is to full satisfaction. Indeed, each variable in a goal condition produces something similar to a resistor for the full goal satisfaction.

The class diagram for implementing this metric is shown in Figure 5 where R2S is the main abstract class that contains R_{Max} and a reference to the current Context object, and two public methods for calculating series and parallel of resistances.



<<abstract>>
R2S
+ RMAX = 100
- w : Context
+ series(op : Array[R2S]) : Double
+ parallel(op : Array[R2S]) : Double
+ direct_res( ) : Double
+ inv_res( ) : Double
Context
...
...
+ models(p : Predicate) : Boolean
+ models(c : Condition) : Boolean
...
PredR2S
- p : Predicate
+ direct_res( ) = if (w.models(p)) 0 else RMAX
+ inv_res( ) = if (w.models(p)) RMAX else 0
LogicANDR2S
- desc : Array[R2S]
+ direct_res( ) = series(desc)
+ inv_res( ) = parallel(dec)
LogicORR2S
- desc : Array[R2S]
+ direct_res(t) = parallel(desc)
+ inv_res( ) = series(dec)
LogicNOTR2S
- desc : R2S
+ direct_res( ) = desc. inv_res(w)
+ inv_res( ) = = desc. direct_res(w)

Fig. 5. Class Diagram for implementing the ResistanceToFullSatisfaction metric. For reasons of conciseness we revealed the body of some of the methods with an inline notation (example: direct_res(w) = parallel(desc)).

To describe the algorithm, we start focusing on the NOT operator that is not directly translated into a formula. Indeed it changes the way how R2S is calculated for its operand:

– the negation of a predicate condition p implies, $R2S_{!p} = R_{Max}$ when $W \models p$ otherwise it is 0;
– the negation of an AND condition between a couple of predicates is calculated as a parallel of resistances: $R2S_{!AND} = \frac{1}{\sum \frac{1}{R2S_i}}$;
– whereas the negation of a OR condition is a series of resistances: $R2S_{!OR} = \sum R2S_i$.

In other words, these formulas are the opposite of those described in Figure 3. To generalize this behavior, we define two modalities for calculating R2S: direct (the normal way - Figure 3), and inverse (the negated way). We extend the notation with $R2S^!$ for indicating the latter modality.

Consequently, we can set: $R2S_{!\phi} = R2S^!_{\phi}$

For coherence, the abstract class (and all its descendants) have a couple of public methods: *direct_res* uses the direct modality, whereas *inv_red* uses the inverse modality.

The whole procedure for calculating R2S is recursive, and it is summarized in the following schema:

formula	class	$R2S(\phi)$	$R2S^!(\phi)$
$\phi = p$ (a predicate)	PredR2S	$0 \mid R_{Max}$	$R_{Max} \mid 0$
$\phi = \psi_1 \wedge \psi_2$	LogicANDR2S	$R2S_{\psi_1} + R2S_{\psi_2}$	$\frac{1}{\frac{1}{R2S_{\psi_1}} + \frac{1}{R2S_{\psi_2}}}$
$\phi = \psi_1 \vee \psi_2$	LogicORR2S	$\frac{1}{\frac{1}{R2S_{\psi_1}} + \frac{1}{R2S_{\psi_2}}}$	$R2S_{\psi_1} + R2S_{\psi_2}$
$\phi = !\psi$	LogicNOTR2S	$R2S^!(\psi)$	$R2S(\psi)$

5 Discussion

To discuss the introduced metrics, we will now propose a few examples. Let us consider the following goals:
$g_1 = (A \vee B) \wedge C$
$g_2 = A \vee (B \wedge C)$
and their corresponding equivalent circuits and resistances to satisfaction as reported in Figure 6.

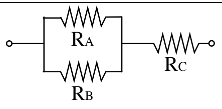
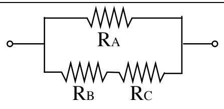
Goal	$g_1 = (A \vee B) \wedge C$	$g_2 = A \vee (B \wedge C)$
Equivalent circuit		
Resistance to satisfaction	$R2S_1 = \frac{R_A \cdot R_B}{R_A + R_B} + R_C$	$R2S_2 = \frac{R_A \cdot (R_B + R_C)}{R_A + R_B + R_C}$

Fig. 6. Examples of goals with related equivalent circuits and distance to satisfaction metrics.

Considering the different combinations of true/false values for the three variables, we can calculate the results as reported in Figure 7 under the hypothesis: $R_A = R_B = R_C = R_{max}$. As it can be seen, regarding goal g_1, we can note three different conditions for full goal satisfaction (combinations # 3,5,7) and five different partial satisfaction conditions (combinations # 0,1,2,4,6). This situations are worth some considerations:

1. The first consideration is straightforward: if we associate some kind of cost evaluation to the satisfaction of each of the logical variables in the goal formula, and we assume each of them has the same cost, we can see that dealing with full goal satisfaction, conditions # 3,5 are less costly than condition # 7. Of course, this could not be the case if the satisfaction of each variable is assumed to have a different cost. This allows for the selection of the most economical plan if more than one is available for goal satisfaction.
2. We may have three different levels of partial satisfaction for goal g_1 according to the different values of R2S. The smallest one (condition #1, value: $\frac{R_{max}}{2}$) represents the situation where there are two alternative paths towards full

#	A	B	C	R2S$_1$
0	0	0	0	$\frac{3}{2}R_{max}$
1	0	0	1	$\frac{R_{max}}{2}$
2	0	1	0	R_{max}
3	0	1	1	0
4	1	0	0	R_{max}
5	1	0	1	0
6	1	1	0	R_{max}
7	1	1	1	0

#	A	B	C	R2S$_2$
0	0	0	0	$\frac{2}{3}R_{max}$
1	0	0	1	$\frac{R_{max}}{2}$
2	0	1	0	$\frac{R_{max}}{2}$
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

Fig. 7. Results for the resistance to satisfaction in the two example goals according to the different combinations true/false (1/0) of the variables (by hypothesis: $R_A = R_B = R_C = R_{max}$).

goal satisfaction (that is making either A or B true). Conditions #2,4,6 report a higher value for $R2S_1 = R_{max}$ and this is an intuitive representation of the fact that there is only one optional path towards full goal satisfaction, making C true. The highest value of the distance to goal satisfaction is represented by condition (#0 value: $\frac{3}{2}R_{max}$) and this correctly represents the fact that at least two variables are to be made true in order to achieve the goal.

Similar considerations may be drawn for the second goal of Figure 6 and corresponding results of Figure 7.

Another advantage of the proposed approach is that it allows the runtime evaluation of the progression towards goal satisfaction. This feature is relevant in the solution space exploration performed by MUSA. In fact, by continuously measuring the distance to goal satisfaction the system can monitor the successful execution of the selected strategy and to estimate the distance from the optimal path even if multiple goals are pursued at the same time.

Besides, the approach proves useful in monitoring goals where some proposition is to be maintained over time (maintenance goals) rather than reached once (achievement goals). While maintaining a goal, the immediate identification of a more than zero distance to satisfaction may trigger some corrective action that could also depend on the measured distance. In fact, in some conditions, this distance may indicate the system is in a stable or unstable situation, and different actions need to be performed. Maintenance goals rise other issues related to the role of time in their conditions. These issues may only partially be solved with the proposed approach: an extension is required, with some temporal logic, to explicitly manage (and measure) clauses where, for instance, the goal is to be maintained for (or after/before) a specific amount of time.

Finally, a very interesting improvement to the proposed approach could come from the introduction of different values of resistance for each of the variables in the goal proposition. This could represent a finer-grained representation of the user preferences in terms of the conditions inside a goal if it could not be fully satisfied. A clear example of that could be provided by considering the goal of finding a five-star category hotel, with pool, in the city center (already proposed in Section 2). Supposing in a specific city there is not such a hotel; the user would prefer a five-star hotel in the city center without a pool or a five-star hotel with a pool far from the city center? The proposed approach, theoretically speaking, fully supports this feature but this is not currently implemented in the MUSA middleware. We are working to introduce that in the next sub-release.

6 Conclusions

This work proposes to extend the classical definition of goal achievement that may represent a limit to the way agents reason and adapt themselves. The presented approach allows reasoning with partial satisfaction of goals. We presented the Distance to Goal Satisfaction, and a metric to measure the distance to the full satisfaction of a logic formula representing the goal.

We applied this metric in the context of a self-adaptive middleware (MUSA) in a heterogeneous range of application contexts (from IoT to maritime IT applications).

We are now working to extend the MUSA implementation with a specific support for different degrees of preference in the satisfaction of specific portions of the goal as discussed in Section 5.

Finally, we are also working on a major improvement of the approach that includes the adoption of Linear Temporal Logic in order to support temporal specifications in goals.

References

1. Agnello, L., Cossentino, M., De Simone, G., Sabatucci, L.: Shipboard power systems reconfiguration: a compared analysis of state-of-the-art approaches. In: Smart Ships Technology 2017, Royal Institution of Naval Architects (RINA). pp. 1–9 (2017)
2. Bordini, R., Hübner, J., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason, vol. 8. Wiley-Interscience (2007)
3. Gelfond, M., Lifschitz, V.: Action languages. Computer and Information Science **3**(16) (1998)
4. Jackson, M.: Problem Frames: Analysing and Structuring Software Development Problems. Addison Wisley (2001)
5. Jackson, M., Zave, P.: Deriving specifications from requirements: An example. In: Proceedings of the 17th International Conference on Software Engineering. pp. 15–24. ICSE '95, ACM, New York, NY, USA (1995). https://doi.org/10.1145/225014.225016
6. Letier, E., Van Lamsweerde, A.: Reasoning about partial goal satisfaction for requirements and design engineering. In: ACM SIGSOFT Software Engineering Notes. vol. 29, pp. 53–62. ACM (2004)
7. Namiot, D., Sneps-Sneppe, M.: On micro-services architecture. International Journal of Open Information Technologies **2**(9) (2014)

8. Napoli, C.D., Valentino, M., Sabatucci, L., Cossentino, M.: Adaptive workflows of home-care services. In: In proceedings of 27th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE-2018) (2018)
9. Ricci, A., Viroli, M., Omicini, A.: Cartago: A framework for prototyping artifact-based environments in mas. In: International Workshop on Environments for Multi-Agent Systems. pp. 67–86. Springer (2006)
10. Sabatucci, L., Cavaleri, A., Cossentino, M.: Adopting a middleware for self-adaptation in the development of a smart travel system. In: Intelligent Interactive Multimedia Systems and Services 2016, pp. 671–681. Springer (2016)
11. Sabatucci, L., Cossentino, M.: From means-end analysis to proactive means-end reasoning. In: Proceedings of the 10th International Symposium on Software Engineering for Adaptive and Self-Managing Systems. pp. 2–12. IEEE Press (2015)
12. Sabatucci, L., Cossentino, M., Simone, G.D., Lopes, S.: Self-reconfiguration of shipboard power systems. In: In Proceedings of the 3rd eCAS Workshop on Engineering Collective Adaptive Systems (2018)
13. Sabatucci, L., Lopes, S., Cossentino, M.: Musa 2.0: A distributed and scalable middleware for user-driven service adaptation. In: International Conference on Intelligent Interactive Multimedia Systems and Services. pp. 492–501. Springer (2017)
14. Sabatucci, L., Ribino, P., Lodato, C., Lopes, S., Cossentino, M.: Goalspec: A goal specification language supporting adaptivity and evolution. In: Engineering Multi-Agent Systems, pp. 235–254. Springer (2013)
15. Thangarajah, J., Harland, J., Morley, D.N., Yorke-Smith, N.: Quantifying the completeness of goals in bdi agent systems. In: Proceedings of the Twenty-first European Conference on Artificial Intelligence. pp. 879–884. IOS Press (2014)
16. Thangarajah, J., Harland, J., Yorke-Smith, N.: Estimating the progress of maintenance goals. In: Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems. pp. 1645–1646. International Foundation for Autonomous Agents and Multiagent Systems (2015)
17. Van Riemsdijk, M.B., Yorke-Smith, N.: Towards reasoning with partial goal satisfaction in intelligent agents. In: International Workshop on Programming Multi-Agent Systems. pp. 41–59. Springer (2010)
18. Vukovic, M., Robinson, P.: Goalmorph: Partial goal satisfaction for flexible service composition. In: Next Generation Web Services Practices, 2005. NWeSP 2005. International Conference on. pp. 6–pp. IEEE (2005)
19. Zhou, Y., Chen, X.: Partial implication semantics for desirable propositions. In: KR. pp. 606–612 (2004)
20. Zhou, Y., Van Der Torre, L., Zhang, Y.: Partial goal satisfaction and goal change: weak and strong partial implication, logical properties, complexity. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 1. pp. 413–420. International Foundation for Autonomous Agents and Multiagent Systems (2008)