Computing (2016) 98:709–749
DOI 10.1007/s00607-014-0405-x

Managing the complex data center environment: an Integrated Energy-aware Framework

Alexandre Mello Ferreira · Barbara Pernici

Received: 31 August 2013 / Accepted: 23 April 2014 / Published online: 28 May 2014

Abstract The problem of Information Technology energy consumption has gained much attention due to the always increasing use of IT both for business and for personal reasons. In particular, data centers are now playing a much more important role in the modern society, where the information is available all the time and everywhere. In this context, the aim of this paper is to study energy efficiency issues within data centers from the Information System perspective. The proposed approach integrates the application and infrastructure capabilities, in which the enactment of adaptation mechanisms is aligned with the business process. Based on both energy and quality dimensions of service-based applications, a model-based approach supports the formulation of new constrained optimization problem that takes into consideration over-constrained solutions where the goal is to obtain the better trade-off between energy and quality requirements. These ideas are combined within a framework where time-based analysis allow the identification of potential system threats and drive the selection of adaptation actions improving overall energy and quality requirements, represented by indicators satisfaction. In addition, the framework includes an evolution mechanism that is able to evaluate past decisions feedback in order to adjust the model according to the current underlying environment. Finally, the benefits of the approach are analyzed in an experimental setting.

Keywords Complex information systems · Green information systems · Adaptive services · Green performance indicators · Green IT · Goal-based model · Identification of system threats · Computational intelligence for IS evolution

A. M. Ferreira (✉) · B. Pernici
Dipartimento di Elettronica e Informazione, Politecnico di Milano, Milan, Italy
e-mail: melloferreira@gmail.com

B. Pernici
e-mail: pernici@elet.polimi.it

Mathematics Subject Classification 68U01 · 68U35 · 68T05

1 Introduction

The pervasive usage of Information and Communication Technology (ICT) in the modern society is emerging as one of the critical environmental challenges to be dealt with. Despite the many facets of the problem, like devices manufacturing and disposal, in this paper we focus on energy aspects. Energy consumption and energy efficiency of ICT centers gained priority due to high computing demand and new environmental regulations. Cloud computing paradigm has been an important contributor for increasing data centers importance, where users are always connected through different devices. The Greenpeace 2012 year report [1] states that "data centers are the factories of the 21st century Information age" which can consume as much electricity as 180,000 homes.

In order to drop the electricity load of ICT equipment, Green IT [2] (Information Technology) in its first wave provided solutions to save energy from hardware (such as processors able to scale up and down their computational capacity) and software (like virtualization) layers. In the second wave, which is called as Green ICT/IS (Information System), the solutions are extended to the entire equipment life-cycle, such as eco-friendly procurement and recycling. To cope with these emerging challenges, data centers have adopted Service Oriented Architectures (SOA) [3], in which the available computing resources are shared by several different users or companies. In such systems, the software is accessed as-a-service [4] and computational capacity is provided on demand to many customers who share the same pool of IT resources.

The data center facility is composed by many components that are interconnected at different levels. One of them is the IT Infrastructure, which is composed by servers, network, and storage devices. Techniques such as virtualization, data deduplication, and energy-efficient Ethernet devices have been adopted to improve infrastructure energy efficiency. Server-specific efforts have concentrated on processor dynamic voltage and frequency scaling and server virtualization. The first technique adapts the CPU power when it is not running during critical periods, saving significant amount of energy with little performance reduction [5]. Due to the high computational capacity of modern servers and to the fact that servers typical utilization rates remain between 10 and 50 % [6], virtualization increases data center computational capacity (by hosting multiple virtual servers) without increasing, at least not proportionally, energy demands and floor space.

The main contribution of the present approach is to demonstrate how we can combine some of these actions together with the application characteristics, where the impact of actions enactment throughout the system is evaluated. It provides an integrated mechanism that is able to identify all system elements relationships and to evaluate positive and negative impact propagation throughout them. We argue that the selection of adaptation actions shall be made with caution since they involve many energy and quality parameters. Moreover, the action's enactment need to be monitored in order to analyze expected results to reinforce selected actions or to suggest a different adaptation path.

The adaptation action enactment starts with the proper identification of the problem to be solved. In general, this is made by triggering mechanisms based on predefined

rules. However, we provide a deeper analysis over these mechanisms with respect to the identification of undesired situations, in which options are considered based on the underlying environment running characteristics. These characteristics are composed by analyzing monitored data of high complex environments such as data centers.

The representation of these characteristics is made using sets of indicators, which are defined in terms of quality and energy parameters. Looking at the indicators values and their defined thresholds, system events are evaluated in order to enable system threats recognition. The classification of these threats guides the adaptation selection phase. With this regard, a goal-based model drives the indicators, threats and adaptation actions relationships representation.

As data centers are highly dynamic environments, the approach contains evolutionary mechanisms that are able to capture the executed adaptation results and to modify future adaptation selection or impact propagation relationships. These modifications are driven by confronting the current environment situation, historical data and context changes. All these issues are integrated within an Integrated Energy-aware Framework (IEF) which puts all these components in a coordinated manner.

This paper is organized as follows: Sect. 2 details how energy and quality requirements are represented by indicators thresholds and their satisfaction levels, Sect. 3 explains how goal-based models are used to support system threats identification, Sect. 4 introduces the integrated framework, which combines different modules in order to support adaptation action selection, Sect. 5 demonstrates the approach using experimental data, and finally Sect. 6 describes some of the most significant research work in this direction.

2 Indicators

IT equipment is the data center's core and it is composed basically by servers, network equipment, and storage devices that are available for service-based applications (SBAs) in a loosely coupled manner. As we concentrate on how these applications make use of this environment, we adopt a 3-layers architecture, which is composed of: infrastructure, middleware, and application layers.

The *infrastructure layer* includes all physical energy-hungry equipment. Focusing on SBAs, we consider the CPU as the main energy consumer component, which is passive of runtime adaptation like frequency change. As higher frequencies consume more power than low frequencies, modern processors are able to dynamically scale their operating frequencies through defined P-states. The *middleware layer* manages virtual resources reservation, workload distribution, and resource monitoring. It includes the virtual environment (i.e., VMs and virtual disks) and the application container, where concrete services are deployed. The *application layer* involves the concrete services, abstract tasks and business processes. Concrete services (usually represented by web-services) implement small operations that, when coordinated, perform the activity described by one or more abstract tasks. Finally, abstract tasks represent the business process objectives described as actions.

By monitoring these layers through selected indicators is possible to identify undesired situations such as energy inefficiencies, which are represented by underachieved

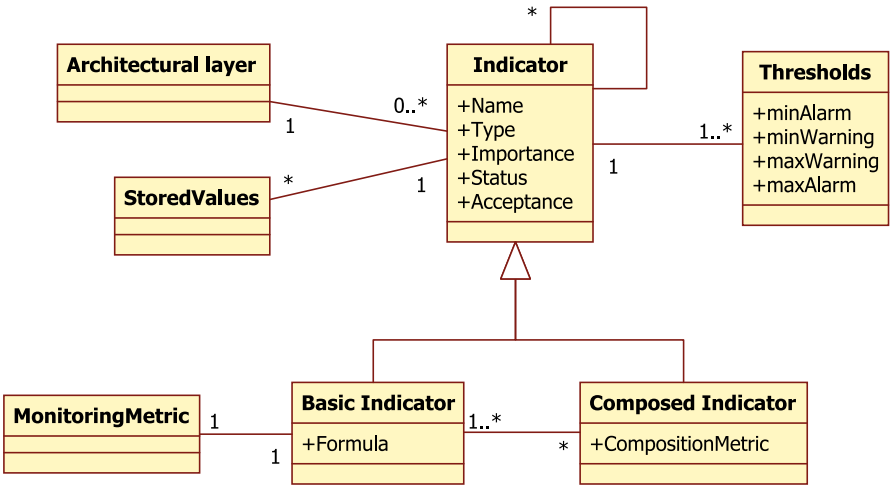

Fig. 1 Indicators meta-model

levels of satisfaction of defined requirements goals. In order to avoid or to recover from these situations, the system shall be able to trigger adaptation actions that improve one or more indicators levels of satisfaction. In this section we introduce indicators in detail and how they can be used to identify system undesired situations, called system threats. To do so we adopt a goal-based model, which is able to represent the diverse relationships between the indicators and the IS.

Figure 1 depicts the indicator meta-model, where the abstract class *Indicator* defines both quality and energy indicators identified by the attribute *type*. The attribute *importance* reflects the user's preferences and dictates the indicator priority in case of multiple thresholds violations.

The indicator threshold violation is defined by *Threshold* class which is composed by two lower-bounds, `minAlarm` (a^{min}) and `minWarning` (w^{min}), and two higher-bounds, `maxAlarm` (a^{max}) and `maxWarning` (w^{max}) such that $a^{max} \geq w^{max} \geq w^{min} \geq a^{min}$. Alarming thresholds represent critical boundaries that should not be violated in order to keep the system soundness. On the other hand, warning thresholds are less critical as their violation can be accepted (although not desired). Thus, warning thresholds can be seem as the system elasticity, which are used to keep the majority of the indicators outside the alarming ranges. An indicator is not violated, i.e., **green**, when its current value is within both warning and alarming boundaries. Otherwise, we can have warning (**yellow**) or alarming (**red**) violation. Alarming indicators have priority to be solved with respect warning ones as they can cause higher damage to the system. The current indicator situation (green, yellow or red) is represented by the attribute *status*.

The indicator last attribute, *Acceptance*, defines the accepted interval in which an indicator can stay in a violated state without requiring adaptation. This attribute is important in order to avoid unnecessary adaptation enactment, where the violation is temporary and might not require adaptation. For instance, let us suppose that the action VM migration may violate the indicator availability of an application. In this

case, the acceptance attribute shall dictate the maximum time interval allowed by the indicator availability to stay in a violated state without triggering a new adaptation action, i.e., the maximum amount of time the VM migration action can take without causing any new side-effect.

In our previous work [7] we argue that adaptation actions enacted at runtime should be aligned with the design characteristics. To do so, an energy-aware controller is responsible to determine the most convenient adaptation strategies, composed by sets of adaptation actions. The selection of the more suitable strategy is based on the definition of a set of adaptation rules that link the violation of an indicator with the set of associated adaptation strategies. However in this paper we provide a more sophisticate method, in which actions are not simply linked to indicators violation, but we consider different threats levels that these violations may represent to the system.

The indicator architectural layer represents where the indicator is placed within our 3-layers architecture. The indicator value is obtained through the indicator formula calculation, that uses the information provided by the monitoring system represented by *MonitoringMetric* class. Indicators are divided in basic and composed. An indicator is called as basic (*BasicIndicator* class) when its calculation formula is either a direct measure from the monitoring system variable or a combination of several monitoring variables within one formula. Instead, composed indicators (*CompositionMetric* class) formulae use other indicators as input.

2.1 Green performance indicators (GPIs)

Focusing on the design of SBAs, energy consumption can be constantly monitored by specific indicators called *GPIs* [8,9]. The aim is to guarantee the satisfaction of energy requirements, specified by GPIs, together with the more traditional functional and non-functional requirements like QoS.

Despite the fact that there is a large number of quality and energy indicators in the literature, most of them either measure the facility as a whole or specific characteristics of one or few physical components. In our approach, an indicator is defined as a metric that provides information about the status of the underlying system. Indicators can be utilized at different levels of granularity. Power consumption, for instance, can be measured from the entire facility to a single software component. In this paper, we focus on indicators that support SBA requirements satisfaction from both energy and quality aspects.

3 System threats identification

Once indicators are defined and their value calculated, the system shall be able to recognize which ones represent system threats, i.e., which ones can harm the system. In this way, an isolated indicator violation does not represent necessarily a system threat. The identification of such system threats introduces new complexity boundaries to the approach, which need to be selective in mining the monitored data in order to identify relevant data to support the decision making mechanisms.

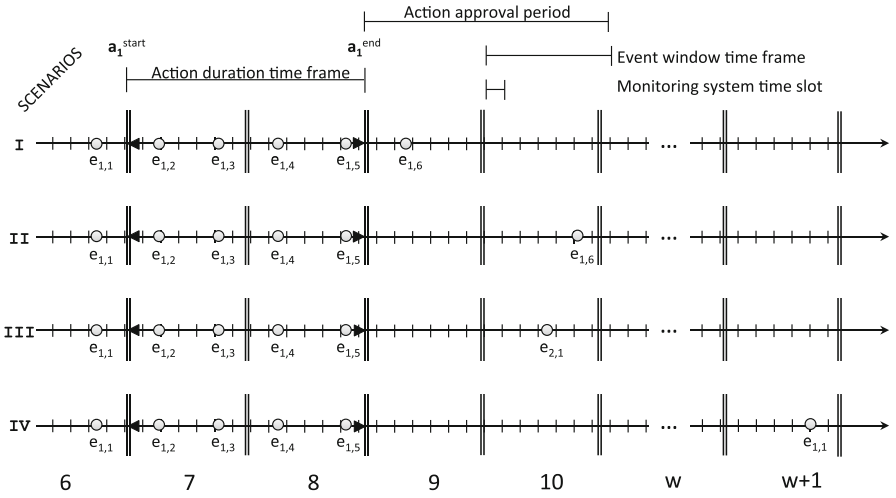

Fig. 2 Timed scenarios

These threats are identified through timed event occurrences, which can appear within four different scenarios depicted in Fig. 2. In these scenarios, an event occurrence is associated with the execution of adaptation actions or indicators violations. Let us assume that an event occurrence $ec_{1,1}$ is created at time window 6. In order to diminish the event impact, which we assume to be negative in this case, adaptation action a_1 is enacted (a_1^{start}). In the meanwhile the event occurrence continues to appear since a_1 is not yet concluded (a_1^{end}). Up to window 8, the event occurrences $e_{1,2}$, $e_{1,3}$, $e_{1,4}$, $e_{1,5}$ are expected since a_1 is not finished. However, occurrences of event e_1 should not appear from window 9. This situation is represented in *scenario IV* and it does not mean that an event occurrence $ec_{1,j}$ will never appear again. However, three other scenarios are also feasible to happen.

In *scenario I*, a_1 is not effective and a new occurrence of e_1 is created, i.e., $e_{1,6}$ and a_1 does not solve the problem. Depending on the next event occurrence window, it is possible to determine if action a_1 was partially or fully failed. *Scenario II* represents the non-effectiveness of a_1, in which the action solves the problem temporally and $e_{1,6}$ appears during the action approval period. This means that a_1 was not suitable to eliminate the threat of event e_1 and a different action should be enacted next time. In *scenario III*, the action a_1 fully solves e_1, but generates a side-effect (expected or not) represented by $e_{2,1}$, i.e., event e_2 was raised. The identification of a raised event as a side-effect of an action enacted to solve another event ($e_x - a_z - e_y$) is determined by two factors: (i) the second event e_y appears within the action a_z approval period and (ii) both events e_x and e_y share at least one monitored variable with action a_z.

In order to make clear the difference between the last three scenarios, let us consider a flat tire example. After the identification of the problem, one possible action to solve the problem is filling the tire with air. If the reason of the problem is a tire hole, this action is not effective since the tire will be flat again in few minutes or hours. This situation is represented in *scenario II*. Instead, if another action is chosen, such as replace the tire, different events can be raised due to the enactment of the action. In

the example, the events car unstable (due to inadequate spare tire) or drive not safe (due to the lack of available spare tire within the car) can be raised. This situation is represented in *scenario III*. Even if the tire replacement effectively solves the problem, it does not prevent that a new hole appears in the future (*scenario IV*).

3.1 Goal-based modeling

The adaptive behavior aims to support the capability of reacting in a (semi) automatic way in case of unexpected situations, which are identified as system threats. Considering that indicators (KPIs and GPIs) represent system objectives, we argue that the adaptation selection phase should take a higher level of abstraction, in which effects and trade-offs of different adaptation options are passive of evaluation. A suitable way to define these goals is using goal-based models presented in Requirements Engineering (RE). These models allow the designers to specify goals of the system and their relationships. Some of the most well-known approaches include Tropos [10], i* [11], and KAOS [12]. In order to support the selection of adaptation actions such that side-effects are minimal, we adopted the goal-based risk model proposed by [13]. The model links system goals (represented by indicators thresholds) to events that might either positively or negatively affect them. The approach also includes the set of actions that can be enacted to intensify or diminish the effects of these events.

As the model and its elements (goals, events and actions) are proposed at design-time, they are based on assumptions that can be verified at runtime. Thus, we believe the model has to support three main features: (i) events occurrences that represent a threat towards the system goals fulfillment, (ii) feedback aware adaptation action selection that analyzes previous executed actions in order to select new ones, and (iii) relationships evolution according to a systematic comparison between the expected and the obtained outcome after the action enactment.

The approach aims to facilitate quality and energy trade-offs such that relationships and their impacts are easily identified while hidden ones are discovered. The initial version of the model was created based on GAMES[1] knowledge base [14], where relevant indicators are identified in the project assessment phase. The project monitoring system was used to provide the required input data for the proposed approach.

Indicators violation might or might not ask for adaptation, but if so, what adaptation should be enacted? Questions similar to that are not easy to answer and, usually, there are many alternative paths to follow. In order to properly represent the impact of an indicator violation within the system and to analyze what are the most suitable adaptation actions within specific contexts, we model the system according to the goal-based modeling approach proposed by [13]. The model is divided in three layers named as *asset*, *event* and *treatment*. Traditionally, the asset layer represents stakeholders' business goals which are taken from the application requirements model. In the present approach however, goals represent indicators targets levels with respect to application non-functional requirements and involve both performance and energy consumption, i.e., KPIs and GPIs thresholds. Differently from the original model, we focus only on

[1] http://www.green-datacenters.eu.

Table 1 Goals evidence calculation according indicator status

Indicator i_h.status	SAT (g_p)	DEN (g_p)	Fulfillment
Green	F/P	N	H
Yellow	F	P	Mh
	F	F	M
	P	P	M
	P	F	Ml
Red	N	F/P/N	L

non-functional requirements (soft-goals) and, for simplicity reasons, we call soft-goals simply goals.

The model nodes are divided into three different types, that are goals G (strategic indicators to be satisfied), events E (circumstances that impact over the achievement of goals), and adaptation actions A (actions used to support the achievement of goals). Thus:

Goals. Goals G are expressed as indicators fulfillment, in which one goal $g_p \in G$ is related to one indicator $i_h \in I$ and I can be a single indicator or an aggregated through a Green Index Function [8]. The goal fulfillment depends on the indicator value with respect to its thresholds. We defined five different levels of goal fulfillment: (H)igh, (M)edium-(h)igh, (M)edium, (M)edium-(l)ow, and (L)ow, where $H > Mh > M > Ml > L$. Based on that, the evidences of satisfaction/denied (SAT/DEN) and the fulfillment of a goal $g_p/g_p \triangleq i_h$ are calculated according the rules based on Table 1.

Events. Events E represent unexpected situations that impact positively or negatively over goals. Although they are defined by the users, events are defined in terms of indicators formula variables. They are expressed by two attributes: *status* and *likelihood*. The event *likelihood* (λ) is calculated according event evidences that support (SAT) or prevents (DEN) an event occurrence, such that $\lambda(e_i) \leftarrow SAT(e_i) \wedge DEN(e_i)/e_i \in E$, where high SAT and low DEN values imply in high event likelihood [13]. However, in order to properly manage the event occurrences we added the attribute *status*. This attribute is responsible to identify the current event with respect to its occurrences and can assume the following values: `ready`, `new`, work in progress (`wip`), `done`, and `renew`. The parameter `ready` states that incoming event occurrences have no relation with past ones. The parameter `new` identifies a new event occurrence that was not yet managed by an adaptation action. The parameters `wip` means that one or more adaptation actions were selected as candidate actions to reduce the event *likelihood* but not yet executed or are currently under execution. If the executed set of actions reduced the event likelihood, the parameter `done` is assumed. This parameter has an expiration time, which represents the actions approval period. The approval period aims to validate the action effectiveness and identify possible solutions that are only temporary. Thus, if a new event occurrence of the same type shows up during this period, the event status is defined as `renew` and a different set of actions should be executed. Otherwise, the event status is returned to `ready` at the end of the approval period.

Adaptation actions. Adaptation actions A are required when a goal fulfillment is *Low*, i.e., there is an indicator alarming violation. An adaptation action represents a single

mechanism able to change one or more monitoring variables such that it supports goals fulfillment by avoiding indicators thresholds violation. This change can be at the infrastructure level (e.g., change the CPU frequency), middleware (e.g., allocate more memory for a VM) or application (e.g., skip the execution of an abstract task within the BP). The proposed approach focuses on design-time adaptation actions, which are divided into two groups: design-time actions and mixed (design-time and runtime) actions. The design-time actions include actions that are selected and executed at design-time, such as initial SLA negotiation. On the other hand, mixed actions are enabled at design-time and executed at runtime, such as skip an abstract task of the BP.

An adaptation action $a_v \in A$ aims to eliminate or diminish a negative event occurrence through an alleviation relation. As described above, this type of relation is used to reduce negative event *likelihood* ($A \xrightarrow[alleviation]{\{-S,--S\}} E$) or *severity* ($A \xrightarrow[alleviation]{\{-,--\}} [E \xrightarrow[impact]{\{-,--\}} G]$), which is based on the event occurrence context rules.

4 Integrated Energy-aware Framework

Following the architecture described in the previous section, indicators violation may represent system threats that have to be eliminated for the system soundness. The threat elimination is represented by the enactment of combined adaptation actions that bring positive and negative impact to the overall system. In order to identify these system threats and to select the best set of adaptation actions to be executed, we based our analysis on the goal-based model. In this section we detail all the elements that are responsible for identifying and eliminating system threats within a framework.

Figure 3 shows the main elements of the framework. First, the `monitoring` system provides raw data about the underlying environment. This information is used to calculate the defined indicators by the `indicator calculation` module. The `event identification` is responsible to verify both monitored variables and indicators values to recognize possible situations that represent a threat to the system. This is done according to the system goal-based model defined in the assessment phase. If a threat is identified, the module creates several event occurrences that are analyzed by the `event analysis` module in order to separate the different types of events and, in particular, the ones that require the enactment of adaptation actions (meaningful events).

The `adaptation selection` module is responsible to select the adaptation actions that eliminate the system threat without creating new ones. Thus, single actions are aggregated as adaptation actions and sent to the `adaptation parameterize` module. In this module the system manager, who is the user responsible for the system environment, shall approve or disapprove the adaptation actions and, if approved, he should provide some necessary actions parameters values. In parallel with the `adaptation selection`, the `history-based analysis` module tries to identify possible misleading relationships within the goal-based model that do not

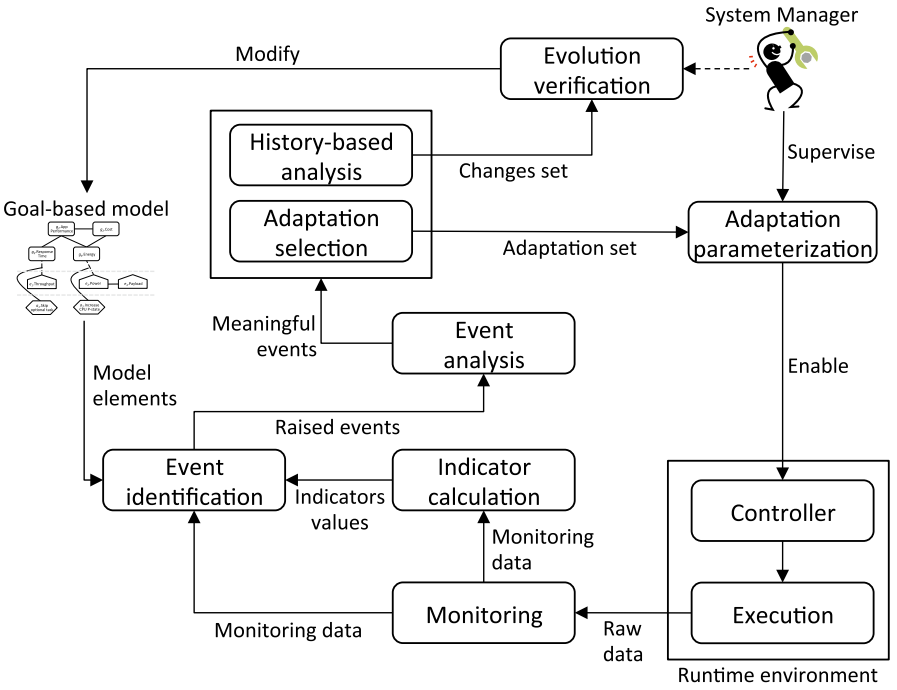

Fig. 3 IEF overview

provide effective adaptation actions. The output of this module implies a new adaptation attempt, a model modification or both. The model modification comprehends: (i) relationships impact labels update, and (ii) revelation of unexpected and not yet modeled relationships. The identified model changes are verified by the system manager in the `evolution verification` module in order to ensure the goal-based model soundness.

4.1 Event identification

The goal-based model described in the previous section supports the identification of different adaptation actions that could be taken in order to mitigate the risk of an indicator violation. However, it does not specify what are the underlying circumstances that may restrict the enactment of an action (or a set of actions). These circumstances are represented by sensible changes in the monitored variables values, which might raise events at the model event layer. We argue that, depending on the gathered context information about the environment, the impact of these raised events towards the asset layer can be negative, positive or null. The identification of possible threats towards goals fulfillment is performed by the `event identification` module from Fig. 3.

Based on the monitored data and the indicator calculated values, the proposed approach recognizes three different levels of threat, which might or might not result

in the creation of an event occurrence. An event occurrence represents that an event is currently harming one or more goals fulfillment and, therefore, adaptation actions shall be enacted in order to restore desired indicators fulfillment levels. The creation of an event occurrence is based the underlying environment, which is represented by the instantiation of the defined set of context rules.

4.1.1 Event context instantiation

In environments such as data centers, the monitoring system variables are likely to vary according to (i) the customer functional and non-functional new requirements; (ii) the outcome of past actions enactment; and (iii) the middleware and hardware environment changes. For this reason, the model should be able to support the identification of different situations, represented by the aggregation of monitored variable values. For instance, the event `server power consumption` can have a positive or negative impact over the goal `energy`. Knowing that energy represents power in time, the increased power consumption may result in application execution time reduction, which makes the server free to be powered-off or used for other users. On the other hand, if the reduction of execution time is not significant, i.e., the gains in performance do not represent gains in execution time, higher power consumption directly implies a higher energy consumption. Thus, it represents a negative impact. This simple example clarifies the event impact polarity (positive or negative), which depends on other monitored variables.

Due to the spread usage of the word 'context' and to avoid misunderstandings, we adopt the context definition provided by [15]: "context is a partial state of the world that is relevant to an actor's goals", where *world* is the underling environment captured by the monitoring system and *actor* is the system itself. In fact, all defined goals are towards the system soundness and effectiveness. However, instead of using statements and facts to represent the context, we define several context rules that are represented as IF-THEN statements where the IF clause is composed by a set of monitored variables aggregated as AND/OR-decomposition. On the other hand, the THEN clause describes the significance and severity parameters of which the IF clause represents with respect to the asset layer. At the event layer, these IF-THEN statements represent positive and negative effects of an event-goal impact relation. In the Event Context Model (ECM) meta-model depicted in Fig. 4, the attribute `CondRule` represents the positive or negative effect of the IF clause and `SigVal` and `SevVal` represents significance and severity of the THEN clause, respectively.

4.1.2 Identifying system threats

Monitored data, from application, middleware and infrastructure layers, is continuously produced by the monitoring system. All this information has to be analyzed so that threats are identified and properly treated. In this module, we create three threat levels sub-modules, which are responsible to recognize situations that might threaten the indicators fulfillment. In order to deal with such a huge bunch of data rapidly, we adopt Stream Reasoning techniques proposed by [16] where streams generated by the

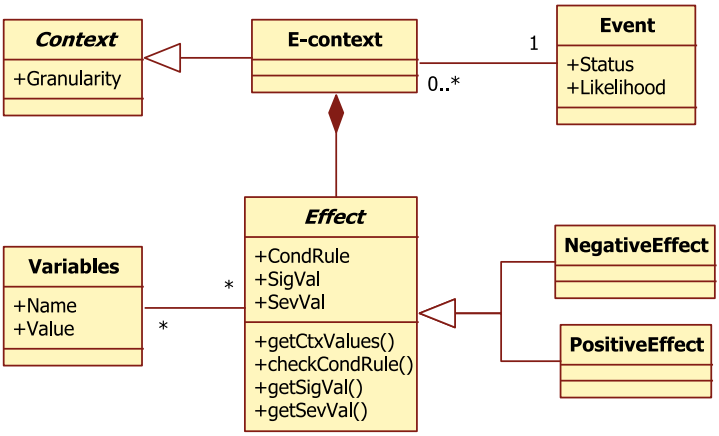

Fig. 4 Event context meta-model

monitoring system are represented as materialized views of RDF[2] triples. These views are based on deductive rules and each triple is associated with an expiration time. Data streams are defined as unbounded sequences of time-varying data elements [18] and the proposed solution manages to inspect "continuous" data streams.

Differently from other types of data, streams are consumed by queries registered into a stream processor, that continuously produce answers. A common and important simplification applied by stream engines is that they process information within windows, defined as periods of time slots in which the flowing information should be considered. Such windows are continuously evolving due to the arrival of new data in the stream, whereas data falling outside the windows is lost, in other words, it expires. The window size defines the stream *expiration time* which represents the triple arrival timestamp plus the window size, assuming the window size to be constant (time-invariant). We also assume time as a discrete and linear ordered variable parameter. Therefore, if the window is defined as three time slots long and a time slot is 5 s long, a triple entering at time τ will expire at time $\tau + 15$ s. The expiration time of derived triples depends on the minimum expiration time of the triples it was derived from. In our approach, derived triples represent different levels of threats that may raise an event.

Considering the current materialized predicates window, represented by $\mathcal{W}$, the `event identification` module derives different levels of threat in order to raise an event based on a set of rules. A threat is identified if one or more indicators are violated with respect to their warning and alarming thresholds. An indicator violation represents the **first level** of a threat, which might or might not raise an event based on the indicator alarming violation duration. This means that, a single indicator violation might not represent an ongoing system problem that requires an adaptation action. For example, during the VM migration action, the application availability indicator may be violated. However, if the migration action is executed normally, i.e., the VM

[2] Resource Description Framework (RDF) is a W3C recommendation for resource description [17].

is successfully migrated without errors, the application availability violation does not require an adaptation action and, therefore, an event occurrence should not be created.

The identification of an indicator violation is represented by the creation of the following entailment:

$$
\begin{aligned}
&i_h.\text{status} = \text{'yellow'} \longleftrightarrow i_h.\text{value} \in Y_h / Y_h = \bigcup_{t=1}^{T} ([a_h^{min}, a_h^{max}](t) \backslash [w_h^{min}, w_h^{max}](t)) \\
&i_h.\text{status} = \text{'red'} \longleftrightarrow i_h.\text{value} \in R_h / R_h = U_h \backslash \bigcup_{t=1}^{T} [a_h^{min}, a_h^{max}](t)
\end{aligned} \tag{1}
$$

where v_h is the calculated value of indicator i_h and U_h is the set of all possible values the indicator can assume and, therefore, $v_h \in U_h$. The set of values that do violate the indicator's warning thresholds (max or min) is defined as Y_h. In the same manner, R_h represents the set of values that violates alarming (max or min) thresholds, in which $(Y_h \cup R_h) \subseteq U_h$. Finally, T represents thresholds sets of i_h where t defines the thresholds dimensions $\{a_h^{min}, w_h^{min}, v_h, w_h^{max}, a_h^{max}\}$.

The *second level* of a threat tries to identify indicators warning violation (*yellow*) that are likely to become alarming (*red*). We represent these indicators as *orange* ones. The identification of an *orange* indicator is similar to the identification of *yellow* ones, but narrowing alarming thresholds. Considering that W_h represents the set of triples of i_h such that warning threshold (min or max) is violated. The alarming thresholds are narrowed based on W_h standard deviation σ (R'_h). The calculation of $\sigma(W_h)$ is $\sqrt{\frac{1}{N}\sum_{n=1}^{N}(x_n - \mu)^2}$ where N is the number of triples in W_h. As we aim only the triples that violate the narrowed alarming thresholds, but do not violate the alarming thresholds, we have $R'_h \cap Y_h$. Thus, an indicator status is defined as *orange* according to the following:

$$
\begin{aligned}
i_h.\text{status} &= \text{'orange'} \longleftrightarrow i_h.\text{value} \in R'_h \cap Y_h / R'_h \\
&= U_h \backslash \bigcup_{t=1}^{T} [a_h^{min} + \sigma(W_h), a_h^{max} - \sigma(W_h)](t)
\end{aligned} \tag{2}
$$

Finally, the **third level** of threat actually raises an event after we have identified indicators with *red* or *orange*. However, differently from the first level of threat, the violation lasts more than it is accepted by the system. As described in the previous section, the indicator alarming violation is linked with a acceptance value acp_h, measured in number of monitored time slots, in which the indicator can stay violated without the enactment of adaptation action. For instance, if the acceptance of an indicator is three time slots, an alarming violation is considered as first level threat until its appearance in the fourth consecutive slot. As each triple is associated with a timestamp, the calculation of the violation duration is obtained by subtracting the timestamp of the last triple by the first one. Considering that V_h is the set of indicators with either *red* or *orange* status, we calculated the violation duration by $MaxTime(V_h) - MinTime(V_h)$, where $MaxTime()$ returns the last violated triple window and $MinTime()$ returns the first

violated triple window. Thus, an event occurrence is created when the following expression hold:

$$MaxTime(V_h) - MinTime(V_h) > acp_h / \forall v \in V_h : v.\text{status} = \text{'red'} \vee v.\text{status} = \text{'orange'} \quad (3)$$

4.1.3 Creation of event occurrences

An event is described by attributes and a set of occurrences that represents raised events over time. Hence, the creation of a new event occurrence $ec_{i,j} \in EC_i$ means that event $e_i \in E$ is raised, where EC_i is the set of occurrences of event e_i. Event occurrences are created by the `event occurrence` module. They are defined by the following attributes: ⟨*timetsamp, value, window, direction, significance, severity*⟩, where:

- *Timestamp*: it is the timestamp of the RDF triple that triggers $ec_{i,j}$ creation.
- *Value*: it is the monitored variable value of the RDF triple that triggers $ec_{i,j}$ creation.
- *Window*: it represents the window in which the RDF triple that triggers $ec_{i,j}$ is placed.
- *Direction*: it dictates if the obtained value is increasing or decreasing based on threshold violation, i.e., max or min. This information is important in order to support the adaptation action selection phase.
- *Significance*: considering the event context model previously described and current monitored variable values, the event occurrence significance is within the range $[0 \ldots 1]$. The calculation multiplies two variables: the significance of the considered monitored variable with respect to other variables that may trigger an event occurrence (defined by the user within the range $[0 \ldots 1]$) and the normalized variable value (within the range $[0 \ldots 1]$). The normalization is calculated as $1 - func(p_1, p_2, p_3)$, in which *funcNorm()* is the scaling functions of [19] and the set of parameters $\{p_1, p_2, p_3\}$ are the variable value, max and min limits. These limits are the violated indicator threshold (alarming) and the variable maximum (or minimum) existing values the variable can assume. The significance is important (together with impact) to define which event occurrences have priority to be eliminated by the adaptation actions.
- *Severity*: this attribute defines, in a qualitative manner, the severity level of event occurrence $ec_{i,j}$ over one or more goals. In this way, it considers not just the relation between the violated indicator (which is represented in the model by a goal $g_p \in G$) and event e_i, but all goals impact relationships that event e_i has within the model. The relation between the violated indicator and the event occurrence is negative ($-$ or $--$), however the event can have positive ($+$ or $++$) impact over other goals. Thus we consider the set of goals that hold impact a impact relation with event e_i, named as $G' \subseteq G$:

$$e_i \xrightarrow[impact]{[++,+,-,--]} g_p / g_p \in G'$$

The creation of an event occurrence depends on both the set of RDF triples within the current materialized window ($\mathcal{W}$) and the ECM. Algorithm 1 details the process of creating a new event occurrence $ec_{i,j}$. The algorithm input parameter is the RDF triple t_k that satisfies Eq. 3, i.e., an indicator alarming violation with longer duration than it is accepted.

The first step is to find out the set of triples that represent a *red* or *orange* violation, according to Eq. 3. This is done by function `getVtriples()` (line 2), which retrieves the set of triples $\mathcal{W}' = \langle i_h \text{ ind:value } v_h \rangle$. As described in the previous section, goals (G) are defined as indicators thresholds and events (E) in terms of monitored variables. If the calculation of an indicator value depends on more than one variable, it means that one goal is impacted by more than one event. Thus, we need to identify the event $e_i \in E'$, where $E' \subseteq E$ represents the set of candidates events to be raised due to i_h violation. In order to create E' we need to identify the goal g_p that represent the violated indicator threshold i_h through the function `getGoal()` (line 3). Based on g_p we select all events that hold impact relationship like $e_i \rightarrow g_p$ (line 4).

The identification of the right event that is triggering the indicator violation is based on the event context-model and, therefore, we need the select the event context-models with respect the event candidates E' (lines 5–6). An event context-model emc_i may have several positive and negative effects context (line 7). Therefore, an event is raised when negative conditional context rules hold (line 9–10), which are based on current context variable values (line 8). Note that an indicator violation can raise more than one event and each event can create one or more event occurrences, depending on the defined event context conditional rules.

Algorithm 1 Creating new event occurrences

Require: t_k
1: $i_h \leftarrow \text{getInd}(t_k)$
2: $\mathcal{W}' \leftarrow \text{getVtriples}(t_k, i_h)$
3: $g_p \leftarrow \text{getGoal}(i_h)$
4: $E' \leftarrow \forall e_i \in E : e_i \xrightarrow{impact} g_p$
5: **for all** $e_i \in E'$ **do**
6: $\quad emc_i \leftarrow \sigma_{emc_i.event=e_i}(ECM)$
7: $\quad$ **for all** $neg\_effect_j \in emc_i.negEffect$ **do**
8: $\quad\quad$ ctx $\leftarrow$ getContextValues(neg_effect_j)
9: $\quad\quad$ **if** checkCondRule(ctx,neg_effect_j) **then**
10: $\quad\quad\quad ec_{i,j} \leftarrow$ new EC(e_i, i_h, neg_effect_j)
11: $\quad\quad$ **end if**
12: $\quad$ **end for**
13: **end for**

Whenever an event has to be raised, a new event occurrence is created. Algorithm 2 describes the event occurrence constructor, in which all event occurrence attributes are properly filled. The function `getTim_Val_Win()` sets *timestamp*, *value* and *window* respectively (line 1). The direction depends on which alarming threshold is violated, i.e., a^{max} or a^{min} (lines 2–10). The indicator attributes $MaxVal$ and $MinVal$ represent the maximum and minimum values the indicator can assume. On the other hand, the variables $limMax$ and $limMin$ are the maximum and minimum

limits used by the normalization function $funcNorm()$ to scale the indicators value. This function is used as weight in order to define the event occurrence *significance* (line 11). The *severity* attribute is obtained by event context conditional rule (line 12). This value is defined by the user during the context creation. Finally, the function `updateEventStatus` updates the status of the event according to the new event occurrence.

Algorithm 2 Creating new event occurrences - *EC* constructor

Require: e_i, i_h, neg_effect_j
1: $ec_{i,j} \leftarrow$ getTim_Val_Win(e_i, i_h)
2: **if** $i_h.a^{max}$ = 'violated' **then**
3: $ec_{i,j}$.dir $\leftarrow$ 'high'
4: $limMin \leftarrow i_h.a^{max}$
5: $limMax \leftarrow i_h.MaxVal$
6: **else**
7: $ec_{i,j}$.dir $\leftarrow$ 'low'
8: $limMin \leftarrow i_h.MinVal$
9: $limMax \leftarrow i_h.a^{min}$
10: **end if**
11: $ec_{i,j}$.sig $\leftarrow getSigVal(neg\_effect_j) *$ funcNorm(
12: $i_h.Val, limMax, limMin$)
13: $ec_{i,j}$.sev $\leftarrow getSevVal(neg\_effect_j)$
14: updateEventStatus(e_i)

Algorithm 3 describes the normalization function used to weight the event occurrence significance. Depending on the event occurrence monotonic function (identified by the attribute *direction*), the function scales the indicator value within the range [0 . . . 1]. In this way, the bigger the value the higher the significance weight is, where 1 is the highest. This weight is multiplied by the indicator significance defined by the user and, therefore, event occurrences with higher significance have priority to be solved as they cause bigger damage to the system.

Algorithm 3 Creating new event occurrences - Significance normalization function

1: **funcNorm** ($i_h.Val, limMax, limMin$){
2: **if** $limMax - limMin \neq 0$ **then**
3: **if** $ec_{i,j}$.dir = 'high' **then**
4: **return** $1 - \frac{limMax - i_h.Val}{limMax - limMin}$
5: **else**
6: **return** $1 - \frac{i_h.Val - limMin}{limMax - limMin}$
7: **end if**
8: **else**
9: **return** 1
10: **end if**
11: }

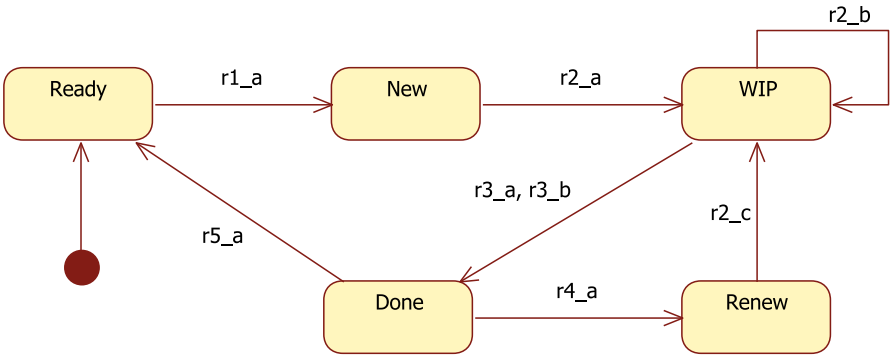

Fig. 5 Event status transition states

4.2 Event analysis

The existence of raised events through event occurrences indicates that adaptation actions have to be enacted. However, we argue the approach should be able to identify the different raised events in order to support the adaptation action instrumentation. The `event analysis` module identifies the current event *status* attribute with respect the event occurrences. The key idea is to map the existing relation between two events such that the system is able to recognize when one event is caused by another through the enactment of its related adaptation action ($E \rightarrow A \rightarrow E$).

In order to represent the many event status transitions, Fig. 5 depicts the states that the event attribute status can assume. The transition from one state to another is defined through several rules, which are defined from 1 to 5. The state `Ready` indicates that $ec_{i,j}$ has no relation with past ones. The state `New` indicates an event occurrence that was not yet managed by an adaptation action. The state `WIP` indicates that actions are under execution and we shall wait for their results. This waiting time is defined by the action attribute *duration*. When all actions are set as 'Finished', the $ec_{i,j}$ assumes the state `Done`. At this point, the event occurrence remains in this state until the action *approvalPeriod* expires, which guarantees the action effectiveness. Thus, if a new occurrence appears regarding to the same event during the approval period, it assumes the state `Renew` and more actions should be executed. Otherwise, the state `Ready` is assumed and new occurrences are not related to past executed adaptation actions.

The detailed description of the rules used in the state transitions depicted in Fig. 5 are expressed as follows:

$$
\begin{aligned}
&\text{“}AdaptationRequired\text{”}\\
&r1_a : \mathcal{W}^{+}(e_i \text{ ev:status 'New'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'Ready'}),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j})
\end{aligned}
$$

$$
\begin{aligned}
&\text{“}OngoingAdaptation''\\
&r2_a : \mathcal{W}^{+}(e_i \text{ ev:status 'WIP'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'New'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \neg\mathcal{W}(a_v \text{ aa:enact 'Finished'})\\
&r2_b : \mathcal{W}^{+}(e_i \text{ ev:status 'WIP'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'WIP'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \neg\mathcal{W}(a_v \text{ aa:enact 'Finished'})\\
&r2_c : \mathcal{W}^{+}(e_i \text{ ev:status 'WIP'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'Renew'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \neg\mathcal{W}(a_v \text{ aa:enact 'Finished'})\\
&\text{“}AdaptationCompleted''\\
&r3_a : \mathcal{W}^{+}(e_i \text{ ev:status 'Done'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'WIP'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \neg\mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}(a_v \text{ aa:enact 'Finished'})\\
&r3_b : \mathcal{W}^{+}(e_i \text{ ev:doneAt } \tau) \text{ :- } \mathcal{W}^{+}\big((e_i \text{ ev:status 'Done'}), ?\tau\big)\\
&\text{“}AdaptationStillRequired''\\
&r4_a : \mathcal{W}^{+}(e_i \text{ ev:status 'Renew'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'Done'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}^{ins}(e_i \text{ ev:occur } ec_{i,j}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}(a_v \text{ aa:enact 'Finished'})\\
&\text{“}AdaptationEffective''\\
&r5_a : \mathcal{W}^{+}(e_i \text{ ev:status 'Ready'}) \text{ :- } \mathcal{W}^{before}(e_i \text{ ev:status 'Done'}),\\
&\qquad\qquad\qquad\qquad\qquad\quad \mathcal{W}(e_i \text{ ev:doneAt } ?doneTime),\\
&\qquad\qquad\qquad\qquad\qquad\quad (e_i \text{ ev:approval } ?approvalPeriod),\\
&\qquad\qquad\qquad\qquad\qquad\quad now() - ?doneTime > ?approvalPeriod
\end{aligned}
$$

where $?time$ represents the current timestamp. In addition, the rules are based on the following materialized windows: $\mathcal{W}$ is current materialized window, $\mathcal{W}^{+}$ contains the derived triples to be added in $\mathcal{W}$, $\mathcal{W}^{before}$ represents the previous materialized window, and $\mathcal{W}^{in}$ represents the new triples that are coming from the `event identification` module, i.e., event occurrences or monitored variables values. Adaptation actions are required only for raised events that hold either `New` or `Renew` status.

4.3 Adaptation selection

When the `event analysis` module generates an event occurrence $ec_{i,j}$ that is related to an event e_i identified as `New` or `Renew` by the attribute status, the `adaptation selection` module process is triggered. The module selects the most suitable adaptation actions in order toeliminate the event occurrence $ec_{i,j}$. A list

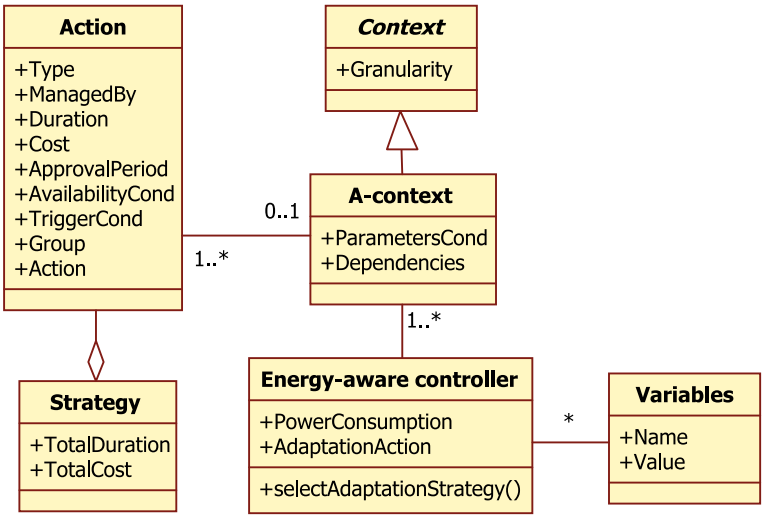

Fig. 6 Adaptation action selection supported by action contexts

of adaptation actions used in this paper for the data center energy-aware framework is shown in Appendix.

As depicted in Fig. 6, an adaptation action is described by the following attributes: `Type` identifies if the action consequence is regarding functionality/quality reduction or resource reallocation; `ManagedBy` identifies if the action is managed by the run-time controller or by the designer at design-time; `Duration` is a time interval attribute that specifies the expected time interval, in terms of maximum and minimum, that the action takes to complete its execution; `Cost` is an interval attribute that represents the expected cost of the action execution and might depend on the action parameters defined by the system manager in the `adaptation parameterization` module; `ApprovalPeriod` is the expected time interval defined to validate the action effectiveness; `AvailabilityCond` represents the set of conditional rules that should be satisfied in order to enable the action execution; `TriggerCond` also represents a set of conditional rules, but it describes triggering conditions that can be either reactive or proactive; `Group` identifies the action group from an energy perspective; and finally `Action` is the adaptation action implementation, i.e., what the action should do.

Considering these attributes, the `adaptation selection` module is responsible for selecting the most suitable set of adaptation actions, which are represented into an adaptation strategy and aims to eliminate event occurrences that are causing one or more indicators violation. Five steps are used to do so. The first step (**Step 1**) identifies the incoming event occurrences that did not trigger adaptation yet. This identification is based on the event status attribute. The next step (**Step 2**) aims to cut off the list of existing adaptation actions in order to keep only the available and suitable ones. In the first case, available actions, we use the action's availability conditional rules. These rules are connected to single BPs that were previously designed to execute the action. For instance, the adaptation action `skip task` can be enacted if and only if the task is defined as `optional task`, i.e., the task is not critical. In the second

case, suitable actions, we use the actions triggering conditions attribute in order to rule out actions from the list that were not designed to stop the incoming event occurrence. Each adaptation action is associated with events (event-trigger) or indicators violation (indicator-trigger) and, depending on the event occurrence, we keep only the actions that are directly or indirectly related to each other through an indicator violation, which is represented by contribution relations within the goal-based model.

Having the subset of actions provided by Step 2, the next step (**Step 3**) determines the optimal set of actions that are supposed to stop the arrival of new event occurrence with minimal side-effects. This step can be divided into four sub-steps, which are:

(i) An indicator violation may represent a violation of other indicators that compose the first one. Thus, the aim of this sub-step is to find the indicators-base that have been violated. This is done through `and/or decomposition` relationships among indicators within the goal-based model.

(ii) Search for previous situations in which the current threat was identified and, most important, what adaptation strategies were selected with their obtained results. Looking at the goal-based model we can have one or more adaptation actions able to mitigate a negative impact relation between one event and one goal. Searching historical log tables we are able to recover the tuple `EventID, ActionID, EventLikelihood_1, EventLikelihood_2, Duration`, in which we can observe the event likelihood reduction after the execution of an adaptation action, i.e., the effective result of an alleviation relation over an impact relation. A list of adaptation actions that did not reduce the event likelihood is created.

(iii) At this point a verification process estimates each action effects, both positive and negative. This is done through the Backward and Forward reasoning algorithms proposed by [13], which the first generates the set of input evidence in order to satisfy high level goals (top-down) and the second propagates the nodes input evidence throughout the model (bottom-up). The actions that propagate more negative effects than positive or do not satisfy minimal duration time or cost are ruled out of the candidate set. After that we ensure that all quality and energy constraints are satisfied (staying at green or warning levels) using the Constraint Satisfaction Optimization Problem.

(iv) Finally, the selection of the adaptation actions and their coordination are performed by the algorithm proposed in [7] performed by the `Energy-aware controller` depicted in Fig. 6. The controller gathers the necessary context information from all the 3-layers (application, middleware, and infrastructure) in order to establish the actions parameters range according to the underlying hardware and software specification. This is represented by the `A-context` class, in which the attribute `ParametersCond` defines this range and the attribute `Dependencies` states possible additional hardware or software limitations for the adaptation strategy enactment.

Once the set of adaptation actions is created, these actions need to be properly coordinated (when there is more than one action involved) in order to compose an adaptation strategy. This is done in *Step 4*, in which input and output parameters of each action are checked in order to identify immediate sequence and parallel patterns. Based on that, the attributes `TotalDuration` and `TotalCost` can be calculated.

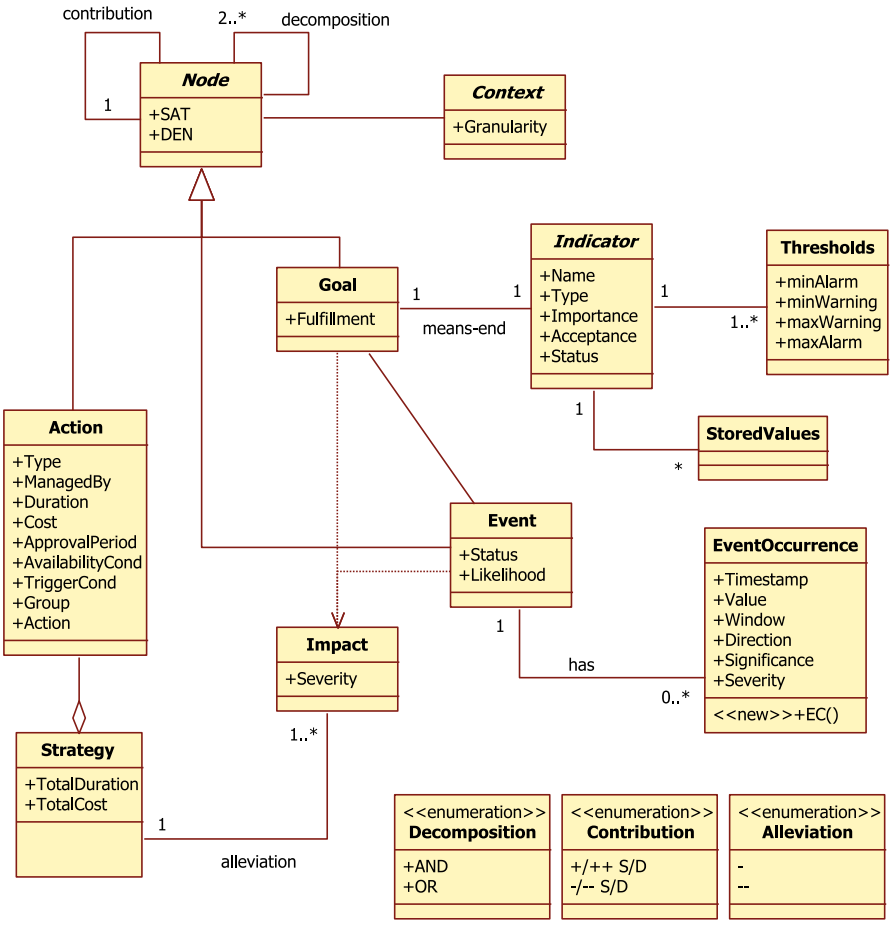

Fig. 7 Meta-model describing the elements inter-relationships

The total duration time is particularly sensible to the adopted flow pattern, sequence or parallel, for the actions execution. If the total duration time or cost exceed their constraints, the process returns to the previous step. Otherwise, the next step (*Step 5*) sends the created adaptation action to the `adaptation parameterization` module, in which the system manager shall validate the initially suggested actions parameters.

The complete meta-model of the described approach is depicted in Fig. 7, in which `node` generalizes goals, events, and adaptation actions. Thus, all nodes contain SAT/DEN evidences that are associated with a context model. Goals are expressed in terms of indicators thresholds, events in terms of event occurrences and actions are aggregated into adaptation strategies. Note that alleviation relations are associated with event occurrences once adaptation actions aim to avoid raised events, which are represented as occurrences.

4.4 History-based analysis

In parallel with the execution of the module `adaptation selection`, the `history based analysis` module analyzes past executed actions and their related events from time to time in order to recognize patterns. These patterns are used to validate and to modify the current version of the goal-based model by creating new impact and contribution relationships (positive or negative) or adjusting the existing relation propagation probabilities. The analysis also benefits the `adaptation selection` and the `event identification` modules by adjusting (refining) the expected duration time attribute of the adaptation actions.

When an adaptation strategy is created to eliminate an event occurrence, the impact relations of each action, defined in the goal-based model, state their expected effects. Sometimes it may happen that unexpected effects are observed and, therefore, we need to know when these situations require model modification. As mentioned, we deal with two types of modifications in our goal-based model: Add or remove impact (event–goal) or contribution (action–event, event–event) relations; Modify SAT/DEN probabilities of contribution relations.

4.4.1 Identification of action feedback

The first important issue to be solved is the identification of an action feedback, which can be: *partially/fully failed* or *successful*. An action is fully failed when it has no impact over the event while partially failed actions are characterized by a temporary action success or creation of many side-effects. On the other hand, an action is identified as successful when it avoids the arrival of new negative event occurrences, which represent system threats. All these scenarios were discussed above through the timed analysis over an incoming event occurrence. However, we also want to be able to justify all the raised events caused by the action enactment. Therefore, we propose a post-enactment rule.

The key point is to express the existing relation between two events such that the system is able to recognize when one event is caused by another through the enactment of its related adaptation action (event–action–event). Through the identification of shared variables is possible to note that an action can have different effects on events, which depend on the monotonicity of the common variable within the events. For example, the events `Server power consumption` and `CPU server utilization` share a dependency relation with the CPU clock speed frequency variable. In these cases, the mitigation of one event may cause the intensification of the other and vice-versa. This property is expressed by the following rule:

$$\begin{aligned}&(e_y, \text{isCausedBy}, (a_z, \text{isEnactedBy}, e_x)) \leftarrow \\ &((a_z, \text{reducesLikelihoodOf}, e_x) \wedge (a_z, \text{increasesLikelihoodOf}, e_y)) \wedge \\ &(Timestamp(e_y.status = \text{'new'}) < Timestamp(e_x.status = \text{'done'}) + a_z.approvalPeriod)\end{aligned} \tag{4}$$

4.4.2 Frequent Item set Mining (FIM) to find frequent patterns

Impact relations concern negative effects of one event over one or more goals. A new impact relation event–goal ($e_x - g_p$) can be added when the appearance of an incoming event occurrence coincides with a goal fulfillment change, like from yellow to red. In some cases, it can be identified in a straightforward way looking at the event and goal shared variables. In this case the rule proposed in Eq. 4 is able to identify that. However, in other cases, it is not so explicit. For instance, let us consider that the event `server payload` has a negative impact relation towards the goal `server cost`. In this example, it is reasonable to suppose that higher payload requests more power and power consumption represent cost. On the other hand, let us consider that the runtime controller allows high payload only during certain periods of the day when power costs are quite low. Thus, the negative impact may be not true while this scenario holds and it would be better to remove or change the impact relation.

Regarding contribution relations, we focus on action–event and event–event as relations that are subject to change. The other contribution relations, i.e., action–action and goal–goal, are considered static as they do not involve an event in the relation. On the other hand, action–event and event–event relations can be added, for instance, when we recognize the appearance of a new raised event e_y just after the execution of a certain adaptation strategy AS_k, which is enacted to eliminate a previous event e_x. In this way we have a complex relation like $e_x - AS_k - e_y$, which means that e_y appears due to the execution of AS_k in order to eliminate e_x. The most problematic issues in this situation are: to correlate the new event occurrence with the enactment of an adaptation strategy, to determine the single action(s) within the adaptation strategy set and to determine which relations shall be added/removed or modified, like $a_z - e_y / a_z \in AS_k$, $e_x - e_y$, or both. As mentioned above, the idea of searching for indicators shared variables in order to identify relationships between two indicators can be used here as well. However, it is not enough to identify unforeseen contribution relations between model nodes when there is no shared variable, like the action `increase CPU P-states` and the event `payload`.

Our objective is to evolve the model with respect to the relations discussed above. In order to provide a proper evolution technique for our goal-based model we search for patterns within historical data. To do so we use FIM, which aims to find groups of items that co-occur frequently in a dataset. Such patterns are normally expressed as association rules in the format: IF [*situation 1*] THEN [*situation 2*].

A FIM algorithm requires as input the item base B set, which is the set of all items under consideration during the mining process, the set of transactions T, which represent some monitored changes of the item in B over a given period of time, and the minimal support $\sigma_{min} \in \mathbb{R}$, $0 < \sigma_{min} \leq 1$, which represent the minimal frequency pattern desired, i.e., how many times an item should appear in the transaction set to be considered frequent. Thus, the expected output of the algorithm is the set of frequent itemsets I that can be represented by the following equation:

$$\Phi_T(\sigma_{min}) = \{I \subseteq B / \sigma_T(I) \geq \sigma_{min}\}$$

Table 2 The rules used to create our transaction T sets

	Relation	Transaction T rule
T_1	Event–goal $(e_x - g_k)$	$ChangedFulfillment(g_p) \wedge (e_x.status = \text{'New'} \vee e_x.status = \text{'WIP'})$
T_2	Action–event $(a_z - e_y)$	$e_y.status = \text{'New'} \wedge e_x.status \neq \text{'Ready'} \wedge a_z.enact = \text{'Ended'}$
	Event–event $(e_x - e_y)$	$e_y.status = \text{'New'} \wedge e_x.status \neq \text{'Ready'} \wedge a_z.enact \neq \text{'Ended'}$

The item base B we consider involves our goal-based nodes, i.e., goals, events and actions whereas the creation of the transaction T set is driven by the rules described in Table 2. As we want to find patterns between raised events and goals that had their fulfillment attribute changed (event–goal, first row), and raised events, enacted actions and another raised events (event–action–event, rows two and three) we define two transaction sets. Let us consider that T_1 represents the impact relations in a given period of time (represented as time slots) and T_2 the set of event–action–event, where $T = (t_1, \ldots, t_n)$ with $\forall k, 1 \leq k \leq n : t_k \subseteq B$. Finally, σ_{min} defines the thresholds that makes this module asks for a model evolution and it is pre-defined by the miner analyst.

There are many different algorithms proposed to mine frequent item sets, such as: Apriori [20], which uses a candidate generation function that exploits the minimal support property; FP-Growth [21], which adopts a divide-and-conquer strategy and a frequent-pattern tree that eliminates the necessity of candidate generation; Tree-Projection [22], which projects the transactions onto the frequent-pattern tree in order to count the ones that have frequent itemsets; H-MINE [23], which dynamically adjusts the links within the mining process.

In our approach we adopt the algorithm proposed by Borgelt [24] called Split and Merge (SaM). The SaM algorithm is divided into two major steps, named SaM. The Split step moves all transactions that start with the same item into a new array and removes the common item. This step is recursively until it finds all itemsets the contain a split item. Then, the Merge step join the created sub-arrays using the well-known *mergesort* algorithm.

In order to analyze the model impact relations we consider only goals that have changed their status, in particular the unfulfilled ones, and raised event that are classified as `New` or `WIP` by the status attribute. Although SaM is not among the fastest approaches, due to the merge step, it uses quite simple data structures and processing schemes. This advantage is important due to memory limitations as we shall have many different mining processes, i.e., with different transaction sets, running in parallel.

Let us consider the incoming event occurrences described by Listing 1.1, where e_2 `Power Consumption` increase has a negative impact over the goal g_4 `Energy Consumption`, represented by the relation $e_2 \xrightarrow{-D} g_4$ where the negative is due to the comparison of e_2 value e g_4 thresholds. In order to stop the arrival of new e_2 event occurrences, action a_2 `Increase CPU P-state` is enacted through the alleviation relation $a_2 \xrightarrow{-} (e_2 \xrightarrow{-D} g_4)$. However, during the action approval time, new event occurrences regarding e_3 `Server Payload` start to arrive.

Listing 1.1 Sample of incoming event occurrences of 30 time slots

```
e[2].status='Ready'
e[3].status='Ready'
⋮
e[2,1]=<07.22.2012 16:02:54,435.00,0006,'I',0.4,'-'>
e[2].status='New'
a[2].action.start()
⋮
e[2].status='WIP'
e[2,2]=<07.22.2012 16:04:24,454.00,0007,'I',0.4,'-'>
e[2,3]=<07.22.2012 16:06:24,427.00,0007,'I',0.4,'-'>
e[2,4]=<07.22.2012 16:07:54,485.00,0008,'I',0.5,'-'>
e[2,5]=<07.22.2012 16:09:54,455.00,0008,'I',0.4,'-'>
⋮
a[2].action.end()
e[2].status='Done'
e[3,1]=<07.22.2012 16:15:24,99.00,0010,'I',0.6,'-'>
e[3].status='New'
e[2].status='Ready'
```

Considering that the situation above have occurred many times, the mining process applies the rules described in Table 2 in order to find new matching patterns. If the value of 'many' is greater than our minimal frequent pattern defined by the variable $\sigma_{min} = 0.5$, rule T_2 identifies a new relation between the enactment of action a_2 and the new event occurrences of e_3. At this point the approach is able to ensure that a_2 reduced the likelihood of e_2 (probability of event occurrences arrival), but also a_2 increased the likelihood of e_3.

In order to recognize this relation, which is not presented in the model, Eq. 4 is applied to ensure that event occurrences of e_3 were caused by the execution of action a_2. The Eq. 5 shows the new contribution relation action–event, which propagates $\{+\}$ evidence, i.e., the completion of a_2 will increase the likelihood of the arrival of e_3 event occurrences.

$$\begin{aligned}&(e_3, \text{isCausedBy}, (a_2, \text{isEnactedBy}, e_2)) \leftarrow \\ &((a_2, \text{reducesLikelihoodOf}, e_2) \wedge (a_2, \text{increasesLikelihoodOf}, e_3)) \wedge \\ &(Timestamp(e_3.status = \text{'new'}) < Timestamp(e_2.status = \text{'done'}) + a_2.approvalPeriod)\end{aligned} \tag{5}$$

4.4.3 Evolution verification

When modifications in the current version of the goal-based model are identified, either through shared variables or mining techniques, the information is sent to the `evolution verification` module. The aim of this module is to show these modifications to the system manager in a GUI way. In this stage of the model evolution an external expert validation is fundamental as special running situations of the system may lead to misleading changes that, instead of improving the current version of the model, can make it worst.

Another important point that we paid attention is regarding to two types of evolution. The system manager can apply a certain modification onto one specific model instance

or the model structure source. When selecting the first option, model instance, the validated modifications do not impact on other model instances, i.e., they are applied exclusively to the current instance. Instead, if the second option is selected, the changes are applied to all current instances of the model. Of course a mixed approach can be used in which some modifications are permanent within the model and others are made for specific instances. In this case, every single modification has to be validated individually by the manager.

5 Evaluation

The aim of the proposed framework is to provide mechanisms that support the identification and enactment of adaptation actions within data centers in order to maximize the trade-off between energy consumption and performance. For this reason we need many other supporting mechanisms that shall be provided by these Service Centers, which are considered as the future generation of current Data Centers. This is the reason why we have a strong link with the EU project Green Active Management of Energy in IT Service centers (GAMES).[3] GAMES architecture fulfills all required mechanisms that were assumed to exist throughout this paper, like the monitoring and runtime environment modules.

Considering that, this section describes how our approach fits within GAMES architecture. The first important module of GAMES used in our framework is the assessment module that supports the creation of the initial version of our goal-based model. Then, without the detailed monitoring system proposed in GAMES, the event analysis of our framework would not have been possible. As we focus on the application design issues that enable the enactment of adaptation actions, the runtime controller proposed in GAMES is essential to analyze the runtime conditions and to validate which actions can be executed based on running context conditions, e.g., one server might not be available due to maintenance actions. After positioning our presented framework within GAMES architecture, we use the obtained results from the project testbeds to fill in a simulated environment in which our approach is implemented and initial results are obtained.

5.1 GAMES architecture

The proposed architecture of GAMES provides the necessary mechanisms to deal with data centers energy issues at different levels simultaneously [14]. GAMES architecture is composed of three main modules:

– Design-Time Environment (DTE): in this module, assessment and mining techniques are used to identify critical situations. It supports SBA and IT infrastructure design when more pervasive and long-term adaptations are necessary. This module also defines the indicators (KPIs and GPIs) that shall drive the adaptations.

[3] http://www.green-datacenters.eu/.

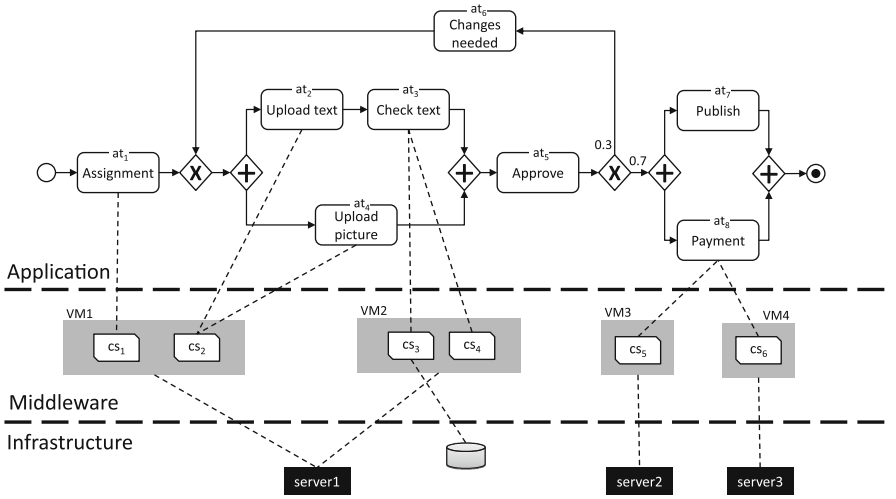

Fig. 8 Online News Publishing System—running example

– Run-Time Environment: this module is able to detect the occurrence of critical situations and, if required, ask for the execution of adaptation. To do so, it combines the DTE output with the current context information provided by the sensing and monitoring system.
– Energy Sensing and Monitoring Infrastructure (ESMI): this module is responsible for collecting, parsing and storing the sensors information. A fundamental element of the ESMI is the Energy Practice Knowledge Base where the data center current and historical information about indicators and context, including configuration parameters, is stored.

The sensing and monitoring infrastructure is in charge of gathering the data produced by the sensor network installed on the data center. These sensors get information not only about the energy consumption of IT devices, but also on the performance and the usage of them. All this data enables the identification of event occurrences and the calculation of defined indicators. Querying GAMES knowledge base repository we can obtain information such as the number of servers that are running or the configurations of the virtual machines installed on a given server. This repository also keeps historical data, which are used by the proposed approach to validate the outcome of adaptation actions in specific contexts.

5.2 Running example

In order to demonstrate how the 3-layers model fits with SOA applications, Fig. 8 shows our BP example within a possible execution environment. The figure clearly separates the layers and their elements after solving the SC problem. For instance, the abstract task at_3 is performed by the composition of two concrete services, cs_3 and cs_4. These services are deployed on $VM2$ and running on server $server1$.

The proposed BP has two possible execution paths ep with their respective probabilities of execution. We define ep_1 as the most probable one (70 %), which represents the article approval. The second execution path, ep_2, the article is rejected and new improved version is requested. In the following we assume that only one modification can be requested. Both execution paths are described below, in which $\langle\ \rangle$ indicates sequential execution, $\{\ \}$ parallel, and $\widehat{\ }$ optional task. In order to provide a more clear representation, ep_{sub} represents the execution of the abstract tasks at_2, at_3, at_4, at_5.

$$
\begin{aligned}
ep_{sub} &= \{\langle at_2, at_3\rangle, \widehat{at_4}\}, at_5 \\
ep_1 &= \langle at_1, ep_{sub}, \{at_7, at_8\}\rangle \\
ep_2 &= \langle at_1, ep_{sub}, at_6, ep_{sub}, \{at_7, at_8\}\rangle
\end{aligned}
$$

We consider that for each abstract task at_x there is at least one concrete service cs_y that can be used to implement the expected functional requirements. The selection of concrete services that attend the minimum functionalities is the selection criteria. After receiving the customer request, all available service providers that are able to functionally satisfy, at least partially, the request requirements are listed. Several approaches deal with this phase [25,26] and it is not our focus. The second selection criteria, called *service selection phase*, relies on the non-functional requirements, which may vary significantly. It consists in ranking and choosing the best concrete service for each abstract task according to global and local constraints, which are represented by our defined indicators thresholds.

Table 3 presents all concrete services candidates for our example during the service selection according to non-functional constraints (we assume that all listed services are functionally equivalent and fulfill the minimum functional requirements). It is worth pointing out that the abstract tasks at_2 and at_4 are functionally equivalent (file upload) and at_3 is composed by two concrete services (spell check and grammar check).

5.3 Event analysis implementation

The `monitoring` module keeps sending monitored data both to the `indicator calculation` and `event identification` modules. Monitored variable values as well as indicators values are provided by GAMES components, which keep sending during specific time intervals. Thus, our framework has to extract meaningful events from this bunch of data that represent system threats. The data is interpreted as continuous streams that are used to keep materialized view of RDF triples. An important implementation issue we had to deal with is the timed-stamped characteristic that incoming streams hold. This issue is solved by using Stream Reasoning techniques proposed by [16] with a special type of RDF triple that represents the different time slots.

The implementation of our timed stream reasoner was made using Jena framework,[4] which is a well-known Java API to deal with semantic web applications and, in particular, with RDF and OWL. The reasoner is divided into two different knowledge types:

[4] http://jena.sourceforge.net/.

Table 3 Candidate concrete services and their qualities dimensions

Abstract task	Concrete service	Execution time	Price	Energy cons.
at_1	cs_{11}	1.30	1.2	65
at_2, at_4	cs_{21}	2.00	5.9	62
	cs_{22}	1.70	4.8	50
at_3	$\langle cs_{31}, cs_{41} \rangle$	2.20	5.2	65
	$\langle cs_{32}, cs_{42} \rangle$	2.50	2.8	55
	$\langle cs_{33}, cs_{43} \rangle$	1.60	3.4	51
	$\langle cs_{34}, cs_{44} \rangle$	1.50	3.3	72
at_5	cs_{51}	2.70	1.1	64
at_6	cs_{61}	0.80	0.5	48
at_7	cs_{71}	1.50	4.0	49
	cs_{72}	1.00	4.4	55
	cs_{73}	1.70	4.3	59
at_8	cs_{81}	7.50	2.0	64
	cs_{82}	5.50	3.0	70

static and dynamic. The static knowledge is represented by the goal-based module and specific technical information about the components within our 3-layers model. At the infrastructure layer, this information is regarding the maximum frequency the processor can operate at, and the capacity of the storage arrays available, as shown in Listing 1.2. At the middleware layer, it is concerned to the VMs configuration parameters that cannot be changed at runtime. Finally, at the application layer it represents the annotated information associated with the BP.

Listing 1.2 Static knowledge sample

```
<rdf:RDF
   xmlns:rdf='http://www.w3.org/1999/02/22-rdf-syntax-ns#'
   xmlns:dt='http://localhost/DC/'>

   <rdf:Description rdf:about='http://localhost/DC/clusterRECS'>
      <dt:hasCPU rdf:resource="http://localhost/DC/node01"/>
      <dt:hasCPU rdf:resource="http://localhost/DC/node02"/>
      .
      .
   </rdf:Description>
   <rdf:Description rdf:about='http://localhost/DC/node01'>
      <dt:type>Intel</dt:type>
      <dt:id>P8400</dt:id>
      <dt:nCores>2</dt:nCores>
      <dt:maxClock>2260</dt:maxClock>
      <dt:minClock>800</dt:minClock>
   </rdf:Description>
   .
   .
</rdf:RDF>
```

The dynamic knowledge refers to the materialized RDF views created initially from the static knowledge and then updated based on the incoming streams. All this information is stored in memory and represents the goal-model instance with all triples set with an infinite expiration time. We represent it by the predicate T. As the window slides over the stream, the incremental maintainer:

1. Puts all incoming triples entering the window in a new predicate called T^{in};
2. Loads the current materialization and T^{in}, which represent the incoming triples;
3. Computes the production rules in order to identify the three different levels of threats;
4. Searches for expired triples;
5. Defines the set of triples to be added T^{+} and removed T^{-} from the materialization;
6. Updates the RDF materialization according to T^{+} and T^{-};
7. Updates the time-stamped triples according to T^{+} and T^{-};

5.3.1 Jena2 inference subsystem

One of the biggest advantages of representing pieces of knowledge as RDF triples is the possibility to apply predefined inferences rules over it. Such rules will lead to new derived pieces of knowledge which combines two or more existing triples in an automated manner and many free tools can be easily found. Example of these tools are MaRVIN [27], Sesame [28] and Jena [29].

Listing 1.3 Jena example - creating the RDF materialization and inference models

```
// Create a model representing the family
Model myRDFmodel = ModelFactory.createDefaultModel();

// Create a new generic rule reasoner to support user defined rules
Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules(ruleSrc));
reasoner.setDerivationLogging(true);

// Create a new inference model over myRDFmodel using the reasoner above
InfModel inf = ModelFactory.createInfModel(reasoner, myRDFmodel);

// Create the RDF stream generator based on GAMES testbeds
RDFStreamsGenerator generator = new RDFStreamsGenerator(inf);
generator.start();
```

MaRVIN states for Massive RDF Versatile Inference Network which performs RDFs inference through a parallel and distributed platform. Its main advantage is regarding to the arbitrary scalability which computes the materialized closure gradually. Such approach releases the users to wait for the full closure to be computed. In Sesame the inference engine is within the *Storage And Inference Layer* (SAIL) which is completely independent and makes possible to implement Sesame on top of a number of repositories without changing any other component. At the semantic level, the *RQL query engine* used in SAIL layer infer new statements using queries that distinguish between schema and data information. However, the most suitable for our purposes is Jena toolkit inference engine, which provides better flexibility

and allows several reasoners to be plugged in it, including a generic rule engine that allows many customization of RDF processing and transformation. Listing 1.3 depicts a piece of Java code in which an empty RDF model and inference model are created together with a new generic rule reasoner. Note that the variable `ruleSRC` contains all predefined inference rules to identify the different threat levels.

Another important feature of Jena inference subsystem is its traceability function. Using a simple method—`InfModel.getDerivation(Statement)`—is possible to trace how an inferred statement was generated, which is one of the key concepts behind the proposed solution. To enable this functionality we just need to start logging all derivations, as it can be seen at line 6 in Listing 1.3.

5.3.2 Threat levels deduction

In order to create an event occurrence, the `event identification` module identifies three different levels of threat, in which only the third level shall actually trigger the creation of an event occurrence. The first level identifies indicators violation, warning or alarming, based on their calculated value and defined thresholds. These indicators are classified as 'yellow' or 'red', respectively. As not all violations lead to an event occurrence, the second level identifies the indicators that are current violating warning thresholds and are also inclined to violate alarming thresholds. Indicators in this situation are classified as 'orange'. Finally, the third level represents indicators that are either classified as 'red' or 'orange' for a longer period than the value defined in the indicator acceptance attribute. In this case, they represent threats to the system and event occurrences shall be triggered.

The identification of all three levels of threats is implemented through deduction rules, in which the stream reasoner analyzes the incoming stream and creates new RDF triples in the materialized view. Based on a set of rules, the mechanism determines the different levels of threat which each monitored indicator value represents to the system. Four inference rules are used for that as depicted in Listing 1.4. The first two rules, rule_1a and rule_1b, define if there is a warning or alarming indicator violation, respectively (threat level 1). Note that the variables $?Y$ and $?R$ represent the possible set of warning and alarming values of indicator $?a$ as defined in Eq. 1. The third rule, rule_2, recognizes indicators that shall be classified as 'orange' (threat level 2). The last rule, rule_3, infers the ones that shall trigger an event occurrence, thus representing a system threat to be eliminated by adaptation actions (threat level 3).

Listing 1.4 Jena inference rules for threat levels identification

```
1  String ruleSrc = ''
2  [rule_1a: (?a http://localhost/EI/hasValue ?b), (?b http://localhost/EI/isIn ?c), (?c rdfs:range ?Y),
3            uriConcat(?a, '$hasValue$', ?b, ?d), (?d http://localhost/EI/hasTime ?time),
4            uriConcat(?a, '$hasStatus$Yellow', ?e)
5            -> (?a http://localhost/EI/hasStatus 'Yellow'), (?e http://localhost/EI/hasTime ?time)]
6
7  [rule_1b: (?a http://localhost/EI/hasValue ?b), (?b http://localhost/EI/isIn ?c), (?c rdfs:range ?R),
8            uriConcat(?a, '$hasValue$', ?b, ?d), (?d http://localhost/EI/hasTime ?time),
9            uriConcat(?a, '$hasStatus$Red', ?e)
10           -> (?a http://localhost/EI/hasStatus 'Red'), (?e http://localhost/EI/hasTime ?time)]
11
12 [rule_2:  (?a http://localhost/EI/hasValue ?b), (?b http://localhost/EI/isIn ?c), (?c rdfs:range ?Y),
13           (?b http://localhost/EI/isIn ?d), (?d rdfs:range ?R'),
14           uriConcat(?a, '$hasValue$', ?b, ?e), (?e http://localhost/EI/hasTime ?time),
15           uriConcat(?a, '$hasStatus$Orange', ?f)
```

```
          -> (?a http://localhost/EI/hasStatus 'Orange'), (?f http://localhost/EI/hasTime ?time)]

[rule_3:  (?a http://localhost/EI/isIn ?b), (?b rdfs:range ?V),
          (?c rdf:first sortByTimeAsc(?V)), (?d rdf:first sortByTimeInv(?V)),
          (?a http://localhost/EI/hasAcceptance ?e), diff(?c ?d ?g), greaterThan(?g ?e)
          uriConcat(?a, '$isIn$', ?b, ?g), (?g http://localhost/EI/hasTime ?time),
          uriConcat(?a, '$triggerEventOccurr$', ?h)
          -> (?a http://localhost/EI/triggerEventOccurr TRUE), (?h http://localhost/EI/hasTime ?time)]'';
```

5.4 The impact on events and actions

Based on some GAMES modules, such as the monitoring system, and the testbed data obtained by Nagios we configure and run our framework modules within a controlled environment. As said before, we do not intend to replace any component of GAMES, but instead, provide additional mechanisms that can be used to improve the existing ones.

Looking at GAMES data we observed that the proposed controllers (Local Control Loop LCL for decisions at the server level and Global Control Loop GCL for decisions at the entire facility) do not have sophisticated mechanisms to identify system threats that require the enactment of adaptation actions, considering as threat all threshold violations. The GCL algorithm, for example, calculates the service center greenness level in order to decide if adaptation actions are needed. If so, the algorithm searches for similar scenarios occurred in the past. In case of new scenarios, reinforcement learning techniques are used to select the actions. The selection process uses a reward/penalty approach towards the service center greenness level. The aim of our proposed event identification and analysis modules is to reduce the number of adaptation actions execution by creating three levels of system threats, in which not all of them shall trigger adaptation actions. In this way, we aim to narrow the set of violated indicators that require adaptation without harming the overall system.

Analyzing the many database tables produced by Nagios we were able to reproduce situations that were responsible for triggering adaptation actions, represented by indicators violations. This was possible only because all experiments were performed three times: without GAMES methodology intervention and two types of controllers approach: fuzzy and bio-inspired [14].

The graph shown in Fig. 9 depicts the accumulated number of raised events that were treated as system threats (i.e., adaptation actions were triggered) within a period of 60 time slots of monitored data. It is possible to notice that GAMES GLC always identified more threats than our Event Identification (EI) module. This is done using our proposed indicator flexibility, represented by the warning thresholds. The `EI_relaxed` represents a wider range for warning indicators violation. Moreover, the indicator acceptance attribute and the three levels of threats avoid the triggering of unnecessary adaptation. For example, design-time actions like SLA renegotiation may cause temporary quality degradation like availability violation. Therefore, if the SLA renegotiation action lasts as expected, the availability violation should not trigger any other adaptation. It is worth mentioning that at time slot 6 the three scenarios (`GAMES GLC`, `EI_strict` and `EI_relaxed`) represent similar levels of goals satisfaction, although the first two have higher expectations with respect to the third one.

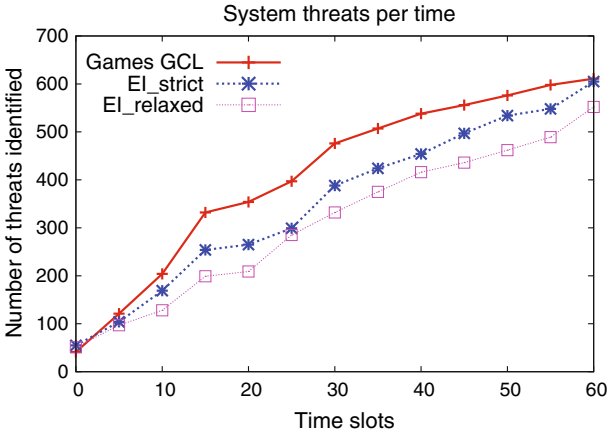

Fig. 9 Graph identifying the number of threats before action enactment

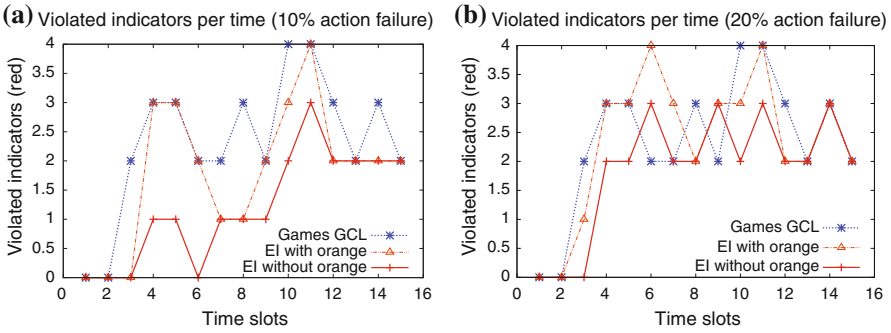

Fig. 10 Number of violated indicators: **a** 10 % of action failure, **b** 20 % of action failure

The proposed approach also reflects in the number of violated indicators. The graphs depicted in Fig. 10 compare the number of alarming indicators violation, i.e., 'red' status, in three scenarios: (i) `GAMES GLC` that was executed following GAMES controllers in a real testbed at HLRS facility; (ii) `EI with orange` that was executed in a simulated environment but with GAMES sensors data as initial input and takes into account the indicators classified as 'orange' status together with the 'red' ones; and (iii) `EI without orange` that was also executed in a simulated environment, but without considering 'orange' indicators as eminent system threats. The simulation was conducted using an expected adaptation action failure rate, in order to fairly compare our simulated experiments against GAMES testbed experiments. Figure 5a represent 10 % of actions failure, while Fig. 5b 20 %.

Our proposed approach reaches the desired levels of goals satisfaction considering a fewer number of violated indicators. Differently from the previous figure, in these figures we do not consider the number of system threats, but the number of violated indicators that can or cannot represent a system threat in our approach. Instead, in GAMES all indicators violations are considered as threats and require adaptation actions. In almost all time slots (in both graphs) our scenarios have less alarming

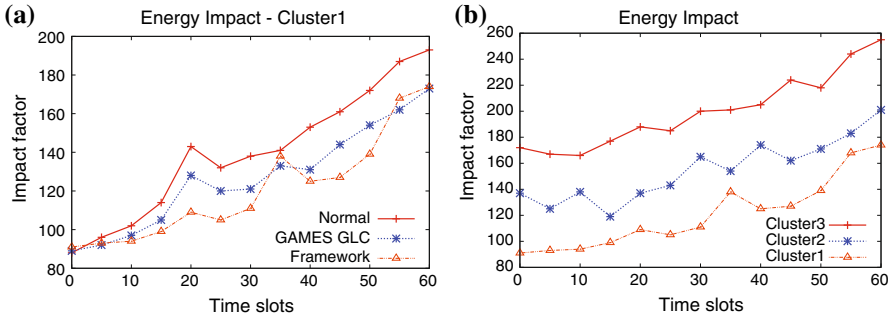

Fig. 11 Energy impact factor comparison during 60 time slots: **a** cluster 1–18 nodes, **b** three different clusters

violated indicators. It does not mean that our enacted actions got better results since we considered the same set of actions, but that warning violation are not seen as threats. Although the second scenario, `EI with orange`, has to deal with more violations, we argue that 'orange' indicators have to be considered as 'red' in this phase as they are very likely to become 'red'.

In order to compare the different solutions from an energy perspective, we adopt an `Energy Impact Factor`. This value is composed of the estimation of two physical components usage: CPU and memory, where CPU utilization has the biggest impact on energy consumption [30]. The idea is to quantify how the execution of different approaches can impact on energy consumption. The adoption of this factor is justified in face of the difficult to obtain of energy consumption values within simulated environments. For example, if the execution of one application service requires more computational power (CPU) than other one, its energy impact is also higher.

The graph depicted in Fig. 11a shows the energy impact factor of the entire cluster (i.e., the 18 server nodes) during a period of 60 time slots, which represent part of the execution of the BP. The first important improvement that both GAMES and our proposed framework provide is the reduction of two peaks of power consumption. Then, we can see that most of the time our framework slightly overtakes GAMES controller by having less energy impact, making clear the advantage of dealing with less raised events and, therefore, less adaptation actions. As adaptation actions are intrinsic related to other actions and events, it is unlikely that the enactment of an adaptation actions does not produce any side-effect and, by doing so, the system can enter into an infinite cycle. The presented approach aims to reduce the number of raised events and enacted actions in order to increase the probability of the overall actions effectiveness.

Making use of a simulated environment, we extend the experiments applied to our solution through three different clusters scenarios depicted in Fig. 11b. The objective of this experiment is to demonstrate our framework behavior within different physical environments by increasing/decreasing two variables: the number of servers available in the clusters and their respective CPU. These variables were doubled from one cluster to another. The important data we can extract from the graph is that the energy impact do not change significantly among the clusters, demonstrating the framework energy consumption advantages even within more complex scenarios.

Our proposed approach was able to keep less violated indicators based on a fewer number of raised events. This was possible due to our refined mechanisms that recognize different types of system threats and, therefore, the adaptation selection can focus on the most significant ones. Also, the FIM mining technique used to support our goal-based model evolution also played an important rule to refine the event identification and adaptation selection modules. Due the identification of unforeseen impact relations, the adaptation selection module could reduce the appearance of new indicators violations as a side-effect of the adaption action execution.

6 Related work

6.1 Green business process metrics

Starting from general software engineering metrics, Kaner and Bond [31] delineate a framework for evaluating software metrics regarding to their purpose, scope, calculation formula, value meaning, and their relationships. In the approach, a metric is generally defined as "the empirical, objective assignment of numbers, according to a rule derived from a model or theory, to attributes of objects or events with the intent of describing them". Distinctively from software engineering metrics, the introduced metrics evaluate BP design attributes and mashup applications performance and energy parameters during its execution. In order to automatically identify the KPI relations and extract their potential influential factors, Popova and Sharpanskykh [32] formalize the concept of performance indicator and their internal (between indicators) and external (indicators and processes) relationships. However, the relation with external concepts (such as process) does not specify the process instance. Runtime variables may cause ambiguous values interpretation of the same process having two or more instances. This problem is partially solved by Rodriguez et al. [33] approach that quantifies relationships in the performance measurement system (PMS) context. This is done by defining the relationships among KPIs and mapping them with PMS in order to create cause-effect relations at business goals level.

The relationships between performance indicators are identified by applying two mathematical techniques over the data matrix. The first, principal component analysis, recognizes cause-effect relations based on each indicator description. The KPIs that contain cause-effect relations are named Business Drivers Key Performance Indicators because of their high factor of impact with respect the others. The second technique, partial least squares models, quantifies the importance degree of each identified cause-effect relation. These models are represented by typical regression equations that predict effect(s) from cause(s) variable(s), called PLS models, which is highly based on the designer expertise.

Taking into consideration the green aspects of the BPs, Nowak et al. [34] introduce the green Business Process Reengineering methodology in order to tackle existing SBAs and energy consumption issues from a holistic approach within modern data centers. The authors introduce the Key Ecological Indicators (KEIs), which are special types of performance indicators to measure up business process greenness. The approach relies on impact of the process execution decisions into the company's busi-

ness goals. For instance, to reduce the KEI "CO_2 emission" of a shipping process, the solution adopted is to reduce the number of times per day the shipper picks up the charge from three to one. Instead, the "CO_2 emission" with respect to the IT resources used to execute the BP is not taken into consideration.

6.2 SBA self-adaptation frameworks

SBAs are characterized by independent services that, when composed, perform desired functionalities [35]. In general, these services are provided by third parties and are utilized by different applications. Thus, SBAs operate in a heterogeneous and constantly changing environment. Thus, they have to be able to constantly modify themselves in order to meet agreed functional and quality constraints in face of a raised problem or an identified optimization or an execution context change [36]. In the ambit of service oriented computing, application adaptive features play an even more important role.

Bucchiarone et al. [37] delineate the effect of application design principles (design-time) within its execution (runtime) recovery capabilities. Thus, a SBA life-cycle that focus on adaptation was created which covers both requirements changes and performance issues. In the proposed life-cycle design-time and runtime are represented together such that they support each other. Adaptation strategies, which are composed by a set of actions, are triggered according to application degradation (functional/non-functional) or context shift that involves both environmental and stakeholders' requirements. They are selected based on the adaptation *scope* and *impact*, which may vary depending on the strategy trigger. The identification of such triggers within complex systems such as a data center is not an easy task due to the many components involved.

Considering more comprehensive approaches, SBA adaptation frameworks deal with large different types of service adaptations and, in particular, propose an integrated view from the infrastructure to the application layer [38–40]. These frameworks dynamic adapt SBAs from both structural and behavioral aspects by taking into account software and hardware changes. The Process with Adaptive Web Services framework proposed by [38] divides service adaptation issues in process design (design-time) and execution (runtime) phases. The importance of process design phase is emphasized as it actually enables autonomous service adaptation at runtime. Focusing on the inter-relationships among the different adaptation actions, [39] present a cross-layer SBA adaptation framework. The aim of the approach is to align the adaptation actions and monitored events from the different layers in order to obtain more effective adaptation results. Three layers are taken into consideration: business process management (BPM composed by the business process workflow and KPIs), service composition and coordination (SCC composed by service compositions and process performance metrics–PPM) and service infrastructure (SI composed by service registry, discovery and selection mechanisms).

The key point made by the authors regarding to their framework is that it takes into consideration the dependencies and effects of such actions within the three different layers. First, they tackle the lack of alignment of monitored events such that events and their mechanisms have to be related in a cross-layer manner. It enables their correlation and aggregation. Second, the lack of adaptation effectiveness is filled by providing a

centralized mechanism able to aggregate and to coordinate different adaptation actions that are triggered by the same event. Third, the lack of adaptation compatibility, which means to identify adaptation necessities across layers by identifying the source of the problem that generated the event. Finally, the lack of adaptation integrity is dealt in terms of foreseeing results. It means to ensure if the selected adaptation actions are enough to achieve the desired results and how many times they need to be enacted.

Mirandola and Potena [41] framework also considers dynamic service adaptation based on optimization models in order to minimize adaptation costs and enforce QoS aspects. The necessity for adaptation is assessed through a context-aware self-adaptation mechanism that captures required data about the environment and triggers appropriate adaptation actions. The novelty of the framework relies on the fact that it handles both software and hardware adaptation from functional and non-functional requirements perspectives. In addition, the framework optimization model is flexible as it is independent from adopted methodology or architectural model. In a similar way, Psaiser et al. [40] present VieCure framework. The framework focuses on unpredictable and faulty behavior of service into a mixed system of Service-based Systems and Human-provided Services. Feedback loop functions are used to provide the framework self-adaptation and behavior monitoring features through a MAPE-k cycle (Monitor, Analyze, Plan, Execute, and Knowledge). The monitoring components are responsible to gather and to store information about different systems, mixed systems, regarding to the infrastructure, application activities, and QoS. The aggregation of such information is therefore presented as events that trigger the diagnosis and analysis component. These components define the required recovery actions by analyzing historical failure data sources.

6.3 Goal-driven SBA adaptation

Goal-driven models have been widely used in Software Engineering field in many different ways and, especially in Requirements Engineering (RE), where the objective is to focus on *why* systems are constructed instead of *what* features the system has to comply with [42]. Goal-Oriented Requirements Engineering provides richer and higher-level abstraction models from which reasoning techniques are used to answer *why*, *who* and *when* questions during early software development phases. The goal-driven models allow the designers to specify the system goals and their relations such that they are aligned with the system requirements.

Gehlert and Heuer [43] propose service replacement adaptation every time there is a new available service that better contributes towards the application goals fulfillment (self-optimization) and provides equivalents functionalities. In order to do so, the authors use a goal-driven approach that enables satisfaction analysis of single goals with respect to the entire model provided by Tropos [10]. After the identification of functional equivalent new services, the approach distinguishes four different situations for adaptation: (i) the new service provides equal goals satisfaction; (ii) the new service provides different goals satisfactions ratios; (iii) the new service adds new functionalities to the application which are expressed as new hard-goals in the model (goal extension); and (iv) the new service has less functionalities, but can be combined

with other services in order to fulfill the expected goals into a better way than before (goal reduction). Based on that, the Tropos quantitative reasoning algorithms are used in order to calculate goals satisfiability and deniability, which properly identify the gains of the adaptation.

Looking at keeping stakeholders goals aligned with the system runtime behavior, **M**onitoring and **A**daptation **E**nvironment for **S**ervice-**o**riented **S**ystems [44] (MAE-SoS) tackles both system design-time and runtime aspects. If goals are not fulfilled, variability models are used to perform semi-automated corrective adaptation actions. These variability models support two types of adaptations based on execution performance and stakeholders' variations. Considering a running example, the authors demonstrate how to trace the impact of runtime system behavior within high-level goals and vice versa. However, the approach does not consider the inter-dependencies among different adaptation actions, which may limit the variability models scope. Such need is partially accomplished by [45] through the usage of the Belief-Desire-Intention agent models of Tropos. Tropos methodology is extended in order to support interrelationships between goals and the system environment where SBA failures and correspondent recovery actions are represented as design abstractions. Based on Tropos models, the authors introduce a fault modeling dimension, which captures errors that may lead to failures, their symptoms, and the linking between symptoms and possible recovery actions.

7 Concluding remarks

Though the research on green computing can follow several directions, this paper aimed to cover service oriented aspects considering both the design and the execution of SBAs. In order to consider the energy aspects of a SBA, we first propose some metrics in order to extract the application main characteristics in a quantitative manner, which are used to support the calculation of GPIs. For this matter, detailed information about these indicators and their dependencies are presented.

In order to support the adaptation action selection, we propose a framework that is based on a goal-based model to analyze the actions impact propagation throughout the system model. The novelty relies on the components, which support the identification of system threats based on pattern recognition and context evaluation. Data-stream reasoning mechanisms are used to evaluate diverse scenarios and to identify relationships among system threats and enacted actions at runtime. In parallel, mining techniques are used to ensure that the considered goal-based model instance is adequate with the underlying system environment by evolving the model elements (like unforeseen relationships) according to monitored data. Thus, it is possible to have different instances of the same model that fit within different environmental configurations.

This framework is integrated within GAMES methodology, which provides the surrounding elements, such as monitoring system, to enable execution of the proposed approach.

The research work presented by this paper has provided solutions to the problem under investigation. But still unsolved issues delineate many directions to be followed. For instance, the solution presented is towards one single service center. Extending it to

federated data centers new issues arise and the complexity of managing desired goals levels become an even more critical aspect. Existing research towards this directions is being studied in $ECO_2Clouds$[5] and FIT4Green[6] EU projects. Further work is also needed to identify which are the best indicators to be analyzed, the decisions about adaptation, and on the effect (short and medium term) of adaptation action.

Acknowledgments This work has been partially supported by the GAMES project (http://www.green-datacenters.eu) and Eco2Clouds EU Project (http://eco2clouds.eu), which are partially funded by the European Commission under the 7th Framework Program grant agreement numbers 248514 and 318048, respectively. This work expresses the opinions of the authors and not necessarily those of the European Commission. The European Commission is not liable for any use that may be made of the information contained in this work.

Appendix: List of adaptation actions

See Table 4.

Table 4 Energy-aware adaptation actions at all architecture layers

Action	Description
Design-time actions at application layer	
1. BP redesign	Redefinition of the process functionalities
2. Structure change	Changes with respect to routing tasks
3. Optional flow definition	One or more execution paths can be skipped
4. Optional task definition	The execution of one or more abstract tasks can be skipped
5. Non-critical task definition	Task quality and energy constraints can be relaxed
6. Redundancy elimination	Tasks that are redundant are decommissioned
7. Service replacement	Service can be replaced by functional equivalent ones
8. Service migration	Services can migrate together with their associated VM
9. SLA renegotiation	Renegotiate to reduce functional and non-functional minimum requirements
Runtime actions at middleware and infrastructure layers	
1. VM migration	The VM container execution from one server to another
2. VM deploy	Create a new VM
3. VM undeploy	Decommission a VM
4. VM reconfiguration	Reallocate more or less resources for the VM
5. Change CPU P-state	Increase or decrease CPU P-state level
6. Change disk mode	Change the disk to acoustic or normal mode
7. Change server mode	Hibernate/wake-up servers that are expected to stay idle for short period of time
8. Shutdown server	Turn-off/on servers that are not expected to be used for long period of time

[5] http://eco2clouds.eu.

[6] http://www.fit4green.eu.

References

1. Cook G (2012) How clean is your cloud? Report, Greenpeace International http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud/
2. Velte T, Velte A, Elsenpeter R (2008) Green IT: Reduce your information system's environmental impact while adding to the bottom line. McGraw-Hill, New York
3. Papazoglou MP, Heuvel WJ (2007) Service oriented architectures: approaches, technologies and research issues. VLDB J 16(3):389–415
4. Turner M, Budgen D, Brereton P (2003) Turning software into a service. Computer 36(10):38–44
5. Lim MY, Freeh VW (2007) Determining the minimum energy consumption using dynamic voltage and frequency scaling. In: Proceedings of the 21th International Parallel and Distributed Processing Symposium. IPDPS, pp 1–8
6. Barroso LA, Hölzle U (2007) The case for energy-proportional computing. Computer 40(12):33–37
7. Cappiello C, Fugini M, Mello Ferreira A, Plebani P, Vitali M (2011) Business process co-design for energy-aware adaptation. In: Proceedings of 4th International Conference on Intelligent Computer Communication and Processing. ICCP'11, IEEE, pp 463–470
8. Mello Ferreira A, Pernici B, Plebani P (2012) Green performance indicators aggregation through composed weighting system. In: Proceedings of ICT as Key Technology against Global Warming, Lecture Notes in Computer Science, vol 7453. Springer, Berlin, pp 79–93
9. Kipp A, Jiang T, Fugini M, Salomie I (2012) Layered green performance indicators. Future Gener Comput Syst 28(2):478–489
10. Bresciani P, Perini A, Giorgini P, Giunchiglia F, Mylopoulos J (2004) Tropos: an agent-oriented software development methodology. Auton Agents Multi Agent Syst 8(3):203–236
11. Yu ESK (1996) Modelling strategic relationships for process reengineering. PhD thesis, Toronto, Canada, UMI Order No. GAXNN-02887 (Canadian dissertation)
12. Dardenne A, Van Lamsweerde A, Fickas S (1993) Goal-directed requirements acquisition. Sci Comput Program 20(1–2):3–50
13. Asnar Y, Giorgini P, Mylopoulos J (2011) Goal-driven risk assessment in requirements engineering. Requir Eng J 16(2):101–116
14. Bertoncini M, Pernici B, Salomie I, Wesner S (2011) GAMES: Green Active Management of Energy in IT Service Centres. In: Information Systems Evolution. Lecture Notes in Business Information Processing, vol 72, pp 238–252. Springer, Berlin. doi:10.1007/978-3-642-17722-4_17
15. Ali R, Dalpiaz F, Giorgini P (2010) A goal-based framework for contextual requirements modeling and analysis. Requir Eng 15(4):439–458
16. Barbieri DF, Braga D, Ceri S, Della Valle E, Grossniklaus M (2010) Incremental reasoning on streams and rich background knowledge. In: Proceedings of the 7th International Conference on The Semantic Web: research and Applications, volume Part I. ESWC'10, pp 1–15. Springer, Heidelberg
17. Manola F, Miller E (2004) Rdf primer. http://www.w3.org/TR/rdf-primer/
18. Aggarwal CC (2006) Data streams: models and algorithms (Advances in database systems). Springer, New York
19. Zeng L, Benatallah B, Ngu AHH, Dumas M, Kalagnanam J, Chang H (2004) Qos-aware middleware for web services composition. IEEE Trans Softw Eng 30(5):311–327
20. Agrawal R, Srikant R (1994) Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases. VLDB'94. Morgan Kaufmann Publishers Inc., San Francisco, pp 487–499
21. Han J, Pei J, Yin Y (2000) Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM International Conference on Management of Data. SIGMOD'00. ACM, New York, pp 1–12
22. Agarwal RC, Aggarwal CC, Prasad VVV (2001) A tree projection algorithm for generation of frequent item sets. J Parallel Distrib Comput 61(3):350–371
23. Pei J, Han J, Lu H, Nishio S, Tang S, Yang D (2001) H-mine: Hyper-structure mining of frequent patterns in large databases. In: Proceedings of the 2001 IEEE International Conference on Data Mining. ICDM'01. IEEE Computer Society, Washington, pp 441–448
24. Borgelt C (2010) Simple algorithms for frequent item set mining. In: Koronacki J, Ras Z, Wierzchon S, Kacprzyk J (eds) Advances in machine learning II, vol 263, Studies in computational intelligence-Springer, Berlin, pp 351–369

25. Plebani P, Pernici B (2009) URBE: Web service retrieval based on similarity evaluation. IEEE Trans Knowl Data Eng 21(11):1629–1642
26. Hao Y, Zhang Y (2007) Web services discovery based on schema matching. In: Proceedings of the Australasian conference on Computer science. ACSC'07, Australian Computer Society, Inc., pp 107–113
27. Anadiotis G, Kotoulas S, Oren E, Siebes R, van Harmelen F, Drost N, Kemp R, Maassen J, Seinstra F, Bal H (2009) Marvin: a distributed platform for massive rdf inference. http://www.larkc.eu/marvin/btc2008.pdf
28. Broekstra J, Kampman A, van Harmelen F (2002) Sesame: a generic architecture for storing and querying RDF and RDF schema. In: Proceedings of International Semantic Web Conference (ISWC), pp 54–68
29. Reynolds D (2009) Jena 2 Inference Support. http://jena.sourceforge.net/inference/
30. Bohra A, Chaudhary V (2010) VMeter: Power modelling for virtualized clouds. In: Proceedings of the 2010 IEEE International Symposium on Parallel Distributed Processing, Workshops and Phd Forum. IPDPSW'10, IEEE Computer Society, pp 1–8
31. Kaner C, Bond WP (2004) Software engineering metrics: What do they measure and how do we know? In: Proceedings of the 10th International Software Metrics Symposium. METRICS'04
32. Popova V, Sharpanskykh A (2010) Modeling organizational performance indicators. Inf Syst 35(4):505–527
33. Rodriguez RR, Saiz JJA, Bas AO (2009) Quantitative relationships between key performance indicators for supporting decision-making processes. Comput Ind Eng 60(2):104–113
34. Nowak A, Leymann F, Mietzner R (2011) Towards green business process reengineering. In: Proceedings of the 2010 International Conference on Service-Oriented Computing. ICSOC'10. Springer, Berlin, Heidelberg, pp 187–192
35. S-Cube Partners (2008) State of the art report on software engineering design knowledge and survey of HCI and contextual knowledge. Deliverable JO-JRA-1.1.1, S-Cube Network of Excellence
36. Kazhamiakin R, Benbernou S, Baresi L, Plebani P, Uhlig M, Barais O (2010) Adaptation of service-based systems. In: Papazoglou M, Pohl K, Parkin M, Metzger A (eds) Service research challenges and solutions for the future Internet., Lecture Notes in Computer ScienceSpringer, Berlin, pp 117–156
37. Bucchiarone A, Cappiello C, Di Nitto E, Kazhamiakin R, Mazza V, Pistore M (2010) Design for adaptation of service-based applications: main issues and requirements. In: Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops. Lecture Notes in Computer Science, vol 6275. Springer, Berlin, pp 467–476
38. Ardagna D, Comuzzi M, Mussi E, Pernici B, Plebani P (2007) PAWS: a framework for executing adaptive web-service processes. IEEE Softw Mag 24(6):39–46
39. Kazhamiakin R, Pistore M, Zengin A (2009) Cross-layer adaptation and monitoring of service-based applications. In: Proceedings of the 2009 International Conference on Service-oriented computing. ICSOC/ServiceWave'09. Springer, Berlin, pp 325–334
40. Psaier H, Skopik F, Schall D, Dustdar S (2010) Behavior monitoring in self-healing service-oriented systems. In: Proceedings of the 34th IEEE Annual Computer Software and Applications Conference. COMPSACW'10, pp 357–366
41. Mirandola R, Potena P (2011) A QoS-based framework for the adaptation of service-based systems. Scalable Comput Pract Exp 12(1):63–78
42. Anton AI (1996) Goal-based requirements analysis. In: Proceedings of the 2nd International Conference on Requirements Engineering. ICRE'96, IEEE Computer Society, pp 136–144
43. Gehlert A, Heuer A (2008) Towards goal-driven self optimisation of service based applications. In: Proceedings of the 1st European Conference on Towards a Service-Based Internet. ServiceWave'08. Springer, Berlin, pp 13–24
44. Franch X, Grunbacher P, Oriol M, Burgstaller B, Dhungana D, Lopez L, Marco J, Pimentel J (2011) Goal-driven adaptation of service-based systems from runtime monitoring data. In: Proceedings of the 35th IEEE Annual Computer Software and Applications Conference Workshops. COMPSACW'11, pp 458–463
45. Morandini M, Penserini L, Perini A (2008) Towards goal-oriented development of self-adaptive systems. In: Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems. SEAMS'08. ACM, New York, pp 9–16