Using Intelligent Agents to Discover Energy Saving Opportunities within Data Centers

Alexandre Mello Ferreira and Barbara Pernici
Dipartimento di Elettronica, Informazione e Bioingegneria
Politecnico di Milano, Milan, Italy
Email: {ferreira,pernici}@elet.polimi.it

***Abstract*—Thinking about the complexity of the data center environment, this paper explores the characteristics of knowledge-based agents to discover energy savings opportunities within these dynamic systems. The goal is to concentrate on the most significant problems while, at the same time, create some flexibility in which undesired metrics are acceptable in face of their positive and negative impact analysis from an energy perspective.**

I. Introduction

Over the last years, managing the energy efficiency of ICT (Information and Communication Technology) has dramatically emerged as one of the most critical environmental challenges to be dealt with. Energy consumption and energy efficiency of ICT centers became priority due to high computing demand and new environmental regulations. Cloud computing paradigm has been an important contributor for increasing the data centers importance, where users are always connected through different devices. The Greenpeace 2012 year report [6] states that "data centers are the factories of the 21st century Information age" which can consume as much electricity as 180,000 homes.

In the present work, we consider the requirements for enabling energy-efficient mechanisms in service-based systems, based on a goal-event-action adaptive approach. One of the problems that emerges when trying to apply adaptive mechanisms based on monitoring in this area is that we need to select adaptation actions in an uncertain environment, in which event occurrences and actions to achieve goals depend on the context of execution.

Looking to provide an integrated solution for reasoning about such complex environment, this paper focuses on the data center energy aspect. Mechanisms are described to: (I) identify potential energy saving without compromising performance metrics and (II) reason about the best action to be taken in order to achieve desired levels of satisfaction. The solution is mainly composed by two knowledge-based agents [18], which are able to discover new opportunities of savings considering a partially observable and highly dynamic environment like data centers.

This paper is organized as following: Section two introduces metrics, called indicators, for controlling performance and energy related issues of an data center; section three discusses about the problem of identifying system threats when we are dealing with many and heterogeneous variables; the proposed solution, an Integrated Energy-aware Framework (IEF), is described in section four; and finally section five briefly discusses some remarking points.

II. 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. They are monitored by several sensors installed throughout the facility, which provide information both from physical (e.g., server energy consumption) and logical layers (e.g., web-service response time). Such data represent levels of satisfaction when we use them to calculate performance and energy indicators. They are defined in terms of thresholds, which can trigger alarming or warning alerts when their limits are not respected. When it happens, repairing actions are selected and enacted in order to restore desired levels of satisfaction.

Such elasticity is only possible when there is a comprehensive view of the system, in which the different elements relationships are stipulated and alerts are properly signed as system threats. In this section we describe these indicators and how they can be used to identify system threats. *An indicator is defined as a metric that provides information about the status of the underlying system* and is created according to the meta-model described in Figure 1. The abstract class *Indicator* defines both quality and energy related indicators which are identified by the attribute *type*. The attribute *importance* is based on the user's preferences and it dictates the indicator priority in case of multiple violations.

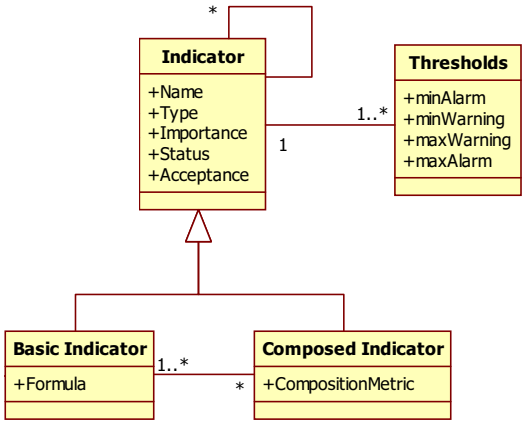
The indicator violation is defined by the *Threshold* class, which is composed by two lower-bounds, `minAlarm` (a^{min}) and `minWarning` (w^{min}), and two higher-bounds, `maxAlarm` (a^{max}) and `maxWarning` (w^{max}). Indicators alarming thresholds represent critical boundaries that should not be 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 zones. An indicator is not violated, i.e. **green**, when its

current value is within both warning and alarming boundaries. Otherwise, we can have warning or alarming violation. In the first case, **yellow**, indicator values are between the warning and alarming thresholds. In the second case, **red**, the indicator value violates both warning and alarming thresholds. It shows an unacceptable situation of one or more system components. Alarming indicators have priority to be solved with respect to 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 (warning or alarming violation) 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 example, the acceptance attribute shall dictate the maximum time interval allowed for which the indicator availability can stay violated without triggering a new adaptation action, i.e., the maximum amount of time the VM migration action can take without causing a new side-effect.

The indicator value is obtained through the indicator formula calculation, which uses the information provided by the monitoring system, which is composed by several sensors. 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 a formula. Instead, composed indicators (*ComposedIndicator* class) formulae use other basic indicators as input. They do not use other composed indicators in order to avoid infinity recursive compositions.

Fig. 1. Indicators meta-model.



Green Performance Indicators : Focusing on the design of SBAs, energy consumption can be constantly monitored by specific indicators called *Green Performance Indicators (GPIs)* [12], [10]. The aim is to guarantee the satisfaction of energy requirements, specified on these GPIs, together with the more traditional functional and non-functional (i.e., QoS) requirements. In order to properly design energy-aware applications, it is fundamental to consider the relationship holding between the structure of an application and the energy consumed by the underlying infrastructure which is executing this application.

III. The problem of 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 system threats introduces new complexity boundaries to the approach, which need to be selective in mining the monitored data in order to identify relevant information to support the decision making mechanisms.

These threats are identified through timed event occurrences, which can appear within four different scenarios. In these scenarios, an event occurrence is associated with the execution of adaptation actions or indicators violations, which occur within defined time windows. The scenarios are:

I - The adaptation action is not effective and event occurrences are created following the same pattern.

II - The adaptation action is temporary effective, solving the problem during a short time interval which defines the action approval period. In this scenario, the same event occurrence comes back within the action approval period which means that the enacted action was not suitable to eliminate the threat and a different solution should be taken.

III - The adaptation action is fully effective to solve the identified threat. However, its execution generates non expected side-effects, creating a completely new threat. This scenario is quite dangerous as it may create an infinity cycle of action enactment and threat creation.

IV - The adaptation action seems to be fully effective, but the same threat appears again after some time. Differently from scenario II, in this scenario the time gap is quite long and the threat appears in this first event occurrence.

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*). A more detailed analysis with respect to an adaptation action feedback and the implications of failed actions is described in the following section.

IV. Integrated Energy-aware Framework

As 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 use a knowledge-based agent approach. In this section we detail all the elements present in an Integrated Energy-aware Framework (IEF), which are responsible for identifying and eliminating threats that may compromise the system.

Figure 2 shows the main elements that compose the proposed framework. First, the `runtime and monitoring` system provides raw data about the underlying environment. This information is used to calculate the defined indicators by the `indicator calculation` module. The first agent, `Agent 1`, is responsible to verify both monitored variables and indicators values in order to recognize possible situations that represent a threat to the overall system. If a threat is identified, the agent creates several event occurrences that are analyzed by the second agent, `Agent 2`, looking to separate the different types of events and, in particular, the ones that require the enactment of adaptation actions (meaningful events). To do so, both agents inference rules are based on the system `knowledge base`, which contains current monitored data and all past experiences. As new information is arriving, the knowledge base is constantly updated by both inference engines. These changes impact on both the conceptualization of system threats and the selection of adaptation actions.

Finally, the `adaptation parameterization` module is responsible to approve or to disapprove the selected adaptation actions by the system manager and, if approved, he should provide the necessary actions parameters values.

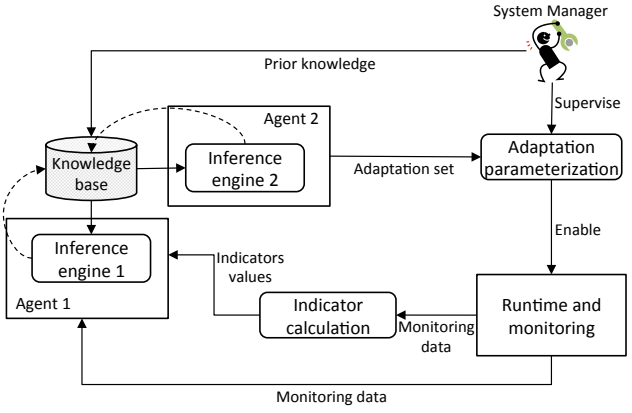


Fig. 2. Framework overview

A. Agent 1

Based on the monitored data and the indicator calculated values, `Agent 1` recognizes three different levels of threats, which might or might not result in the creation of an event occurrence. An event occurrence represents an event that is currently harming one or more indicators fulfillment and, therefore, adaptation actions shall be enacted in order to restore desired levels of satisfaction. The creation of an event occurrence is based on the underlying environment, which is represented by the instantiation of some context rules.

Due to the spread usage of the word 'context' and to avoid misunderstandings, we adopt the context definition provided by [2]: "context is a partial state of the world that is relevant to an actor's goals", where *world* is the agent underling environment captured by the monitoring system, *actor* is the agent itself and *goals* represent the indicators satisfaction. Defined by assert predicates in first-order logic, the context is important to limit the agent observation scope and to clarify the impact relations polarity (positive/negative).

This agent PEAS description is:

- Performance: Number of warning and alarming indicators violation;
- Environment: Indicators levels identified as yellow, orange and red;
- Actuators: Identification of three levels of system threats and creation of event occurrences;
- Sensors: Indicators value and thresholds.

1) Threats identification: Considering that data is continuously produced by the monitoring system, the agent is 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 [4] where streams generated by the monitoring system are represented as materialized views of RDF [1] 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 [1] 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 of 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 3 time slots long and a time slot is 5 seconds long, a triple entering at time τ will expire at time $\tau + 15s$. The expiration time of derived triples depends on the minimum expiration time of the triples it was derived from.

Considering the current materialized predicates window, the `event identification` module derives different levels of threat in order to raise an event based on the a set of rules. A threat is identified if one or more indicators are violated with

[1]Resource Description Framework (RDF) is a W3C recommendation for resource description [11]

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 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:

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

where i_h.value 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 does 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 $\{a_h^{min}, w_h^{min}, v_h, w_h^{max}, a_h^{max}\}$ such that $a_h^{min} \leq w_h^{min} \leq w_h^{max} \leq a_h^{max}$.

The **second level** of a threat tries to identify warned violated indicators (*yellow*) that are likely to become alarmed violated indicators (*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. Let us consider 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 normal alarming thresholds, we have $R'_h \cap Y_h$. Thus, an indicator status is defined as *orange* according to the following:

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

Finally, the **third level** of threat actually raises an event after we have identified indicators as *red* or *orange*. However, differently from the first level of threat, the violation lasts more than it is accepted by the system. The indicator alarming violation is linked with an acceptance value acp_h, measured in number of monitored time slots the indicator can stay violated without the enactment of adaptation action. 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* violation, it is 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 holds:

$$\begin{aligned} &MaxTime(V_h) - MinTime(V_h) > acp_h / \\ &\forall v \in V_h : Status(v, \text{'red'}) \vee Status(v, \text{'orange'}) \end{aligned} \tag{3}$$

2) Creation of event occurrences: 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: $\langle$ *timetsamp, value, window, direction, significance, severity* $\rangle$, 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*: it is a value between $[0..1]$ resulted from two variables: variable significance and normalized variable value based on the indicator thresholds. 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 quantitatively the severity level of event occurrence $ec_{i,j}$ over one or more goals.

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.

Algorithm 1 Creating new event occurrences

Require: t_k
1: $i_h \leftarrow$ getInd(t_k)
2: $\mathcal{W}' \leftarrow$ getVtriples(t_k, i_h)
3: $g_p \leftarrow$ 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**

The first step is to find out the set of triples that represent *red* or *orange* violations, according to Eq. 3. This is done by function `getVtriples()` (line 2), which retrieves the set

of triples $\mathcal{W}' = \langle i_h$ ind:value $v_h \rangle$. 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 represents 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 the right event that is triggering the indicator violation is based on the event context-model and, therefore, we need to 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.

B. Agent 2

Whenever event occurrences are identified and raised by `Agent 1`, the system shall be able to recognize events relationships and properly enact corrective actions. This is done by `Agent 2`, which analyzes situations when one event is caused by another through the enactment of its related adaptation action. By knowing these relationships, the agent drives the system adaptive behavior.

`Agent 2` PEAS description is:

- Performance: Number of raised event occurrences;
- Environment: Current and previous materialized windows;
- Actuators: Analysis of triggered adaptation actions (*AdaptationRequired, OngoingAdaptation, AdaptationCompleted, AdaptationStillRequired and AdaptationEffective*) and selection of new ones;
- Sensors: Event occurrences, event status and enacted actions within materialized windows.

1) 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 selection.

There are five different status transitions which the relation status can represent: (i) `Ready`, $ec_{i,j}$ has no relation with past ones; (ii) `new`, an adaptation was not yet selected; (iii) `wip` (work in progress), there are actions under execution; (iv) `done`, all actions are set as 'finished'; and (v) `renew`, new $ec_{i,j}$ regarding to the same event during the approval period.

The detailed description of the rules used in the states transitions are expressed as follows:

$$
\begin{aligned}
&\text{“}AdaptationRequired\text{”}\\
r1_a : \mathcal{W}^{+}(Status(e_i, \text{‘new’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘ready’})),\\
&\mathcal{W}^{ins}(Occur(e_i, ec_{i,j}))
\end{aligned}
$$

$$
\begin{aligned}
&\text{“}AdaptationStillRequired\text{”}\\
r2_a : \mathcal{W}^{+}(Status(e_i, \text{‘renew’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘done’})),\\
&\mathcal{W}^{ins}(Occur(e_i, ec_{i,j})),\\
&\mathcal{W}(Enact(a_v, \text{‘finished’}))
\end{aligned}
$$

$$
\begin{aligned}
&\text{“}OngoingAdaptation\text{”}\\
r3_a : \mathcal{W}^{+}(Status(e_i, \text{‘wip’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘new’})),\\
&\mathcal{W}^{ins}(Occur(e_i, ec_{i,j})),\\
&\neg\mathcal{W}(Enact(a_v, \text{‘finished’}))\\
r3_b : \mathcal{W}^{+}(Status(e_i, \text{‘wip’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘wip’})),\\
&\mathcal{W}^{ins}(Occur(e_i, ec_{i,j})),\\
&\neg\mathcal{W}(Enact(a_v, \text{‘finished’}))\\
r3_c : \mathcal{W}^{+}(Status(e_i, \text{‘wip’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘renew’})),\\
&\mathcal{W}^{ins}(Occur(e_i, ec_{i,j})),\\
&\neg\mathcal{W}(Enact(a_v, \text{‘finished’}))
\end{aligned}
$$

$$
\begin{aligned}
&\text{“}AdaptationCompleted\text{”}\\
r4_a : \mathcal{W}^{+}(Status(e_i, \text{‘done’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘wip’})),\\
&\neg\mathcal{W}^{ins}(Occur(e_i, ec_{i,j})),\\
&\mathcal{W}(Enact(a_v, \text{‘finished’}))\\
r4_b : \mathcal{W}^{+}(DoneAt(e_i, \tau) \text{ :- } &\mathcal{W}^{+}\big((Status(e_i, \text{‘done’})), ?\tau\big)
\end{aligned}
$$

$$
\begin{aligned}
&\text{“}AdaptationEffective\text{”}\\
r5_a : \mathcal{W}^{+}(Status(e_i, \text{‘ready’})) \text{ :- } &\mathcal{W}^{before}(Status(e_i, \text{‘done’})),\\
&\mathcal{W}(DoneAt(e_i, ?doneTime),\\
&(Approval(e_i, ?approvalPeriod)),\\
&now()-?doneTime > ?approvalPeriod
\end{aligned}
$$

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.

2) Adaptation selection: When there are event occurrences $ec_{i,j}$ identified as "new" or "renew" by the relation `Status`, the adaptation selection is triggered. Actions are combined in order to create an adaptation strategy, which is a composed by a set of coordinated adaptation actions, i.e., actions that are executed in sequence and/or parallel (coordination). It can happen that this set is composed by only one action, so it does not require any coordination.

An adaptation action is described by the following attributes: `Type` identifies if the action consequence is regarding functionality/quality reduction or resource reallocation; `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, Agent 2 selects the most suitable set of adaptation actions looking to eliminate event occurrences that are causing 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;

iii A verification process estimates each action effects, both positive and negative. 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 we ensure that all quality and energy constraints are satisfied, Constraint Satisfaction Optimization Problem is used;

iv Finally, the selection of the adaptation actions and their coordination are performed by the algorithm proposed in [5] performed by the Energy-aware controller. The controller gathers the necessary context information to establish the actions parameters range according to the underlying hardware and software specification.

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. 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.

V. Related work

A. Green business process metrics

Starting from general software engineering metrics, Kaner and Bond [7] 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 [15] 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. [17] approach that quantifies relationships in the performance measurement system context (QRPMS). This is done by defining the relationships among KPIs and mapping them with Performance Measurement System (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 (PCA), recognizes cause-effect relations based on each indicator description. The KPIs that contain cause-effect relations are named Business Drivers Key Performance Indicators (BDKPI) 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. [14] introduce the green Business Process Reengineering (gBPR) methodology in order to tackle existing SBAs and energy consumption issues from a holistic approach within modern data centers. The authors introduce the Key Ecologial 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 business 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.

B. Service-based application self-adaptation frameworks

Service-based Applications are characterized by independent services that, when composed, perform desired functionalities [19]. 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 [8]. In the ambit of service oriented computing (SOC), application adaptive features play an even more important role.

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 [3], [9], [16]. These frameworks dynamic adapt SBAs from both structural and behavioral aspects by taking into account software and hardware changes. The PAWS (Process with Adaptive Web Services) framework proposed by [3] 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 interrelationships among the different adaptation actions, [9] 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: BPM (business process management composed by the business process workflow and KPIs), SCC (service composition and coordination composed by service compositions and process performance metrics - PPM) and SI (service infrastructure 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 assess whether the selected adaptation actions are enough to achieve the desired results and how many times they need to be enacted.

Mirandola and Potena [13] 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 the adopted methodology or architectural model. In a similar way, Psaiser et al. [16] present VieCure framework. The framework focuses on unpredictable and faulty behavior of service into a mixed system of Service-based Systems (SBSs) and Human-provided Services (HPSs). 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.

VI. Concluding remark

The solution described in this paper presents an integrated mechanism to identify and to reason about energy savings opportunities within complex environments such as data centers. It takes into consideration several aspects of a partially observable environment using knowledge-based agents. Simulated results have being performed in order to evaluate the agent behavior under stressed situations and are expected to be published in the future. A similar solution is also under evaluation which is based on a goal-based model instead of agents.

Acknowledgment.

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.

References

[1] C. C. Aggarwal. *Data Streams: Models and Algorithms (Advances in Database Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[2] R. Ali, F. Dalpiaz, and P. Giorgini. A goal-based framework for contextual requirements modeling and analysis. *Requirements Engineering*, 15(4):439–458, November 2010.

[3] D. Ardagna, M. Comuzzi, E. Mussi, B. Pernici, and P. Plebani. PAWS: A framework for executing adaptive web-service processes. *IEEE Software Magazine*, 24(6):39–46, November 2007.

[4] D. F. Barbieri, D. Braga, S. Ceri, E. Della Valle, and M. Grossniklaus. Incremental reasoning on streams and rich background knowledge. In *Proc. of the 7th International Conference on The Semantic Web: research and Applications - Volume Part I*, ESWC'10, pages 1–15, Berlin, Heidelberg, 2010. Springer-Verlag.

[5] C. Cappiello, M. Fugini, A. Ferreira, P. Plebani, and M. Vitali. Business process co-design for energy-aware adaptation. In *Proceedings of 4th International Conference on Intelligent Computer Communication and Processing*, ICCP'11, pages 463–470. IEEE, Aug. 2011.

[6] G. Cook. How clean is your cloud? Report, Greenpeace International, April 2012. http://www.greenpeace.org/international/en/publications/Campaign-reports/Climate-Reports/How-Clean-is-Your-Cloud/.

[7] C. Kaner and W. P. Bond. Software engineering metrics: What do they measure and how do we know? In *Proc. of the 10th International Software Metrics Symposium*, METRICS'04, 2004.

[8] R. Kazhamiakin, S. Benbernou, L. Baresi, P. Plebani, M. Uhlig, and O. Barais. Adaptation of service-based systems. In M. Papazoglou, K. Pohl, M. Parkin, and A. Metzger, editors, *Service Research Challenges and Solutions for the Future Internet*, volume 6500 of *Lecture Notes in Computer Science*, pages 117–156. Springer Berlin / Heidelberg, 2010.

[9] R. Kazhamiakin, M. Pistore, and A. Zengin. Cross-layer adaptation and monitoring of service-based applications. In *Proc. of the 2009 international conference on Service-oriented computing*, ICSOC/ServiceWave'09, pages 325–334, Berlin, Heidelberg, 2009. Springer-Verlag.

[10] A. Kipp, T. Jiang, M. Fugini, and I. Salomie. Layered green performance indicators. *Future Gener. Comput. Syst.*, 28(2):478–489, Feb. 2012.

[11] F. Manola and E. Miller. Rdf primer. http://www.w3.org/TR/rdf-primer/, February 2004.

[12] A. Mello Ferreira, B. Pernici, and P. Plebani. Green performance indicators aggregation through composed weighting system. In *Proc. ICT as Key Technology against Global Warming*, volume 7453 of *Lecture Notes in Computer Science*, pages 79–93. Springer Berlin / Heidelberg, September 2012.

[13] R. Mirandola and P. Potena. A QoS-based framework for the adaptation of service-based systems. *Scalable Computing: Practice and Experience*, 12(1):63–78, 2011.

[14] A. Nowak, F. Leymann, and R. Mietzner. Towards green business process reengineering. In *Proceedings of the 2010 International Conference on Service-oriented Computing*, ICSOC'10, pages 187–192, Berlin, Heidelberg, 2011. Springer-Verlag.

[15] V. Popova and A. Sharpanskykh. Modeling organizational performance indicators. *Information Systems*, 35(4):505–527, June 2010.

[16] H. Psaier, F. Skopik, D. Schall, and S. Dustdar. Behavior monitoring in self-healing service-oriented systems. In *Proc. of the 34th IEEE Annual Computer Software and Applications Conference*, COMPSACW'10, pages 357 –366, July 2010.

[17] R. R. Rodriguez, J. J. A. Saiz, and A. O. Bas. Quantitative relationships between key performance indicators for supporting decision-making processes. *Computers & Industrial Engineering*, 60(2):104–113, February 2009.

[18] S. J. Russell and P. Norvig. *Artificial intelligence - a modern approach: the intelligent agent book*. Prentice Hall series in artificial intelligence. Prentice Hall, 1995.

[19] S-Cube Partners. 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, July 2008.