




## Article

# Dynamic Knowledge Management in an Agent-Based Extended Green Cloud Simulator

Zofia Wrona <sup>1,\*</sup> , Maria Ganzha <sup>1,2,\*</sup> , Marcin Paprzycki <sup>2</sup>  and Stanisław Krzyżanowski <sup>3</sup><sup>1</sup> Faculty of Mathematics and Information Science, Warsaw University of Technology, 00-662 Warsaw, Poland<sup>2</sup> Systems Research Institute, Polish Academy of Sciences, 01-447 Warsaw, Poland;  
marcin.paprzycki@ibspan.waw.pl<sup>3</sup> CloudFerro Sp. z o. o., 00-511 Warsaw, Poland

\* Correspondence: zofia.wrona.stud@pw.edu.pl (Z.W.); maria.ganzha@ibspan.waw.pl (M.G.)

**Abstract:** Cloud infrastructures operate in highly dynamic environments, and today, energy-focused optimization become crucial. Moreover, the concept of extended cloud infrastructure, which, among others, uses green energy, started to gain traction. This introduces a new level of dynamicity to the ecosystem, as “processing components” may “disappear” and “come back”, specifically in scenarios where the lack/return of green energy leads to shutting down/booting back servers at a given location. Considered use cases may involve introducing new types of resources (e.g., adding containers with server racks with “next-generation processors”). All such situations require the dynamic adaptation of “system knowledge”, i.e., runtime system adaptation. In this context, an agent-based digital twin of the extended green cloud infrastructure is proposed. Here, knowledge management is facilitated with an explainable Rule-Based Expert System, combined with Expression Languages. The tests were run using Extended Green Cloud Simulator, which allows the modelling of cloud infrastructures powered (partially) by renewable energy sources. Specifically, the work describes scenarios in which: (1) a new hardware resource is introduced in the system; (2) the system component changes its resource; and (3) system user changes energy-related preferences. The case study demonstrates how rules can facilitate control of energy efficiency with an example of an adaptable compromise between pricing and energy consumption.

**Keywords:** carbon-aware cloud computing; resource management; knowledge management; extended green cloud; rule-based expert systems; expression languages



**Citation:** Wrona, Z.; Ganzha, M.; Paprzycki, M.; Krzyżanowski, S. Dynamic Knowledge Management in an Agent-Based Extended Green Cloud Simulator. *Energies* **2024**, *17*, 780. <https://doi.org/10.3390/en17040780>

Academic Editors: Pedro Faria and Ramiro Barbosa

Received: 28 November 2023

Revised: 31 January 2024

Accepted: 2 February 2024

Published: 6 February 2024



**Copyright:** © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Modern software systems, operating in complex and dynamic environments, should facilitate on-demand changes in their internal structure. In particular, in some cases, the logic of the systems must be modified in order to effectively manage new environmental conditions. Here, specifically interesting are the cases when the environmental alternations not only impact the system’s behavior but also introduce new information, necessitating broadening and sometimes exchanging parts of the system’s knowledge. Furthermore, the most challenging cases arise when the system behavior should, at least in theory, be instantiated in a running system, i.e., without the system being shut down, to complete adaptations.

This aspect is especially relevant to cloud infrastructures, where new types of resources can be added (or removed) from the system according to, for example, clients and market demands. Furthermore, in the context of carbon-aware computing [1–3], which has been gaining popularity over recent years, system resource modifications may occur due to variations in the availability of energy that strictly depend on fluctuating weather conditions. Such changes may also involve introducing (or removing) processing units with resources that were previously non-existent within the system (e.g., introducing the server with Graphical Processing Units) or interchanging parts of the existing system components (e.g., upgrading the server’s processor). The subject of on-demand knowledge modification

can be addressed, among others, by introducing a formal representation of the knowledge, e.g., in the form of knowledge graphs; however, this consideration is outside the scope of this work.

The following contribution proposes a novel way of adapting the system's behavior on the basis of changes in its internal knowledge. The system used as a case example is the Extended Green Cloud Simulator (EGCS) [4], which is a multi-agent digital twin of the so-called Extended Green Cloud (EGC) that allows simulating cloud infrastructures powered, among others, using heterogeneous green energy sources. Here, the system strategies imposing the behavior of the agents have been represented in the form of rule sets facilitated by the explainable Rule-Based Expert System (RES) combined with Expression Languages (EL). The proposed approach enables dynamic behavior adaptations that can streamline the knowledge-management process. In particular, the focus has been placed on adaptations performed within the already-running system without the necessity of it to be rebooted. With respect to resource management, the work introduces an object-oriented representation of the anonymous (from the perspective of current system knowledge) resources and, by incorporating the EL, presents a way of dynamic knowledge acquisition and/or modification.

More specifically, the proposed approach addresses the use cases in which resource modification is connected with energy control aspects of the system. In particular, three use cases are presented in the work: (1) the introduction of new hardware resources in response to their increasing potential in the cloud computing market, (2) the on-demand changes of the resources of existing components, and (3) the adaptation of local resource pricing and client energy-related preferences accordingly to shifts in the energy availability. However, the proposed concepts are fully generalizable and can be used in any type of distributed system in which knowledge is appropriately represented.

The remaining part of this work is structured as follows. Section 2 discusses approaches and problems related to resource management that have been studied in the existing literature. Next, Section 3 contextualizes the work by describing a way in which resources are managed in the EGCS. Afterwards, Section 4 introduces a new conceptualized model of resource representation, with a particular focus on resource management, facilitated by EL. The integration of the model, within the knowledge of EGCS agents, is described in Section 5, whereas Section 6 illustrates the application of the proposed approach in three different use cases. Finally, Section 7 summarizes the findings and proposes future research directions.

## 2. Related Works

The topic of resource management in cloud infrastructures remains an active area of research. Specifically addressed are aspects such as the allocation of existing cloud resources, the provisioning and consolidation of Virtual Machines (VMs), or the assignment of processing units based on clients' demands.

In order to more effectively examine these concerns, the specificity of the cloud infrastructure has been studied. In [5], the authors investigated the nature of the cloud environments, and workload demands, by performing the analysis of the publicly available Google trace data. One of the key findings was the heterogeneity and variability of cloud infrastructure, with respect to both the specification of individual machines and resource demands originating from different clients. Additionally, the authors stated that the cloud environment's diversity comes not only from different available resource types (e.g., CPU, memory, GPU, etc.) but also their underlying attributes (e.g., clock speed of CPU, type of disk drive, etc.), which can be subject to the client's constraints. Here, note that in the context of distributed cloud infrastructures, an additional level of heterogeneity may, for instance, be introduced by: (1) contractors owning separate cloud regions adopting individual resource policies, or (2) bandwidth of network connections between individual resources.

All of these aspects have to be taken into consideration while designing resource management strategies. In [6], the authors proposed a workflow-scheduling algorithm

consisting of two phases: (a) task selection and (b) VM assignment. The proposed resource model took into account the heterogeneity of VMs. However, the VMs were assumed to reside within the same data center, therefore bandwidth differences were out of the scope of the work. Moreover, the available client's constraints included only budget and deadline, whereas, as pointed out in [5], in real-world cloud infrastructures, the demands usually are also set with respect to resource characteristics, which significantly complicates the scheduling process.

On the other hand, the works [7,8] took into account both the diversity of cloud resources, as well as the client's demands. They specifically proposed multi-resource allocation algorithms, focused on process fairness, with respect to the clients. The approach, described in the first work [8], was based on defining a dominant resource for each user, whereas the cloud infrastructure was modelled as a collection of homogeneous servers. Such model excludes real-life applications, in which the servers have different characteristics. Therefore, the second work [7] extended these considerations by taking into account a heterogeneous server pool. However, both these methods focused on mapping general resource types (e.g., CPU, memory) to individual demands, with no consideration of constraints that could, possibly, be imposed on resource attributes.

Similarly, in [9], authors proposed an adaptive deep reinforcement learning method for energy-efficient scheduling. The conceptualized system model takes into account individual machine resource types, and user resource demands, but omits cases in which the attributes of resources may differ between processing units (e.g., different generations of processors in servers). Addressing this aspect is particularly important, since the cloud infrastructure and the client demands may change over time, according to the evolving technological standards. For example, when a client would like to execute a given task on an Intel Xeon Platinum 8280 processor, it is possible that, in the model presented in [9], either such demand would be ignored, or there will be no matching Service Level Agreement (SLA) on the side of the cloud provider, possibly resulting in a task not being executed.

In this context, the differences in the resource specification of SLAs, on the provider side and the client side, has also been studied. For example, the authors of [10] addressed the problem of the lack of standardization and liquidity in SLAs by proposing an adaptive mapping method. The method is based on a modification of an existing semantics of public SLA, carried out by the service provider, to match the upcoming users' demands. It mostly considered adding new parameters to the existing SLA structure, or changing the attributes' names. Similarly, in [11], the authors extended the ontological specification of the service description, proposed in [12], with the possibility of new knowledge derivation. These models could be used to represent the knowledge of individual resources within a processing unit. However, it should be noted that the only operation on the resources, which was taken into account, is the evaluation of sufficiency, with respect to the client's requirements. As such, presented models are not comprehensive enough to adequately describe how the system should handle new resource types, injected into the system knowledge. Here, the proposed models focus strictly on the dynamic representation of the knowledge base, in cases when the way of handling a given resource type is already known to the system. Consequently, they do not capture the full scope of potential cloud infrastructure adaptations.

The adaptations of the cloud infrastructure, from the system logic perspective, have also been discussed. The survey [13] described a scope of existing approaches for the reconfiguration of cloud resources in an autonomic manner. In particular, the gathered works considered adaptations on one of three levels: (1) VM-level (e.g., adjustment of resource amounts, such as CPU, memory, etc.), (2) node-level (e.g., migration of VMs), or (3) cluster-level (e.g., adding/removing nodes). One of the works [14] considered all levels of adaptation, and facilitated them by using MAPE-K-based self-adaptive system architecture. Separately, the work [15] proposed a method of dynamic virtual machine migration and deployment. Nevertheless, all of these studies take into account only the scenario of performing adaptations of already-known resources, including CPU, memory, or

bandwidth. Thus, alternations of the cloud infrastructure, leading to the adoption of new resource types, are not considered.

The adaptability in terms of resource management has also been considered in the broader scope of energy systems. For instance, in [16], authors introduced a Reinforcement Learning (RL)-based adaptive energy management model optimizing the energy utilization for a group of islands that utilize renewable energy sources. The energy transmission in the considered environment was realized by the electric propulsion vessels. The model was based on the discrete-continuous action space combining two parallel actor networks. Its states combined information about the electric load, input from renewable energy sources, the capacity of the storage and the capacity of electric propulsion vessels. The effectiveness of the model was proved in four different use cases. However, none of them modelled emergency situations. Specifically, in the considered environment, additional restrictions put on the transmission of energy may come from the fluctuating weather conditions that could reduce the availability of safe shipping routes. Moreover, the presented model is strictly domain-specific (similarly as in work [17]), and as such, it would be difficult to generalize it to other energy systems. Similarly, the introduction of new knowledge within the proposed model (e.g., new type of electric propulsion vessels with different characteristics) could require significant modifications to the algorithm's flow.

In the generalized context, on-demand adaptability has been studied in terms of agent systems, among others, in work [18] that proposed an ontological and semantical approach to the management of virtual organizations. Specifically, the authors studied the adaptability of agents in terms of both (1) the adaptation of the organization's knowledge and (2) the modification of internal agents' behaviors. The latter adaptability aspect was also considered in [19], which facilitated the implementation of "agent intelligence" using a semantic-based approach. The concepts presented in all of these works could be potentially applied in support of the aspect of resource management; however, they need to be tailored (especially considering current technological standards) to provide a full scope of adaptation.

A review of the existing literature, in the area of resource management and alternation, reveals a gap in the adoption and management of new knowledge. In particular, none of the works directly addresses the problem of on-demand addition of new types of resources, specifically in terms of (1) adapting the system to support their full handling, as well as (2) verifying their attributes' suitability against the client's requirements. Finally, the ability to adapt the system "on the fly" has not been considered.

In particular, to illustrate the issue, let us discuss an example in which an unknown resource type, a server with a Graphical Processing Units (GPU), is introduced to the system. As the servers with GPUs, up to this moment, have not been present, the system does not have any defined "rules" or behaviors to effectively manage them, especially to integrate their usage into current system strategies (such as energy optimization strategy). For example, it may be beneficial for the system to change the server's selection method so that the presence and the amount of GPUs contribute to the servers' prioritization. On the other hand, there is also the possibility that the estimation of the compatibility of the GPU-equipped server with the client's job requirements may differ from the estimation applied to other types of servers (since GPUs have different characteristics than CPUs). In some more complicated cases, it can also happen that introducing a processing unit with a new resource type may require changing or including a new energy consumption model.

While some of these scenarios can be resolved by feeding the system with manually defined parametrized configurations (similarly to the approach introduced in [10]), the others require modification of the entire behavior of the system at the code-base level. The latter cases were not addressed in current research, and, as such, handling them would involve stopping the system, applying modifications and then running it again. This approach, however, is prohibitively expensive, as well as not suited to handling time-critical situations (e.g., changing resource management strategies in order to prevent system failures or adapt to current energy market trends). Hence, the framework introduced in this article, focuses specifically on covering such cases by proposing a novel approach which

gives a possibility of flexible runtime modifications of system behaviors applied upon a new system's knowledge acquisition/modification. However, before proceeding with the description of the developed method, let us introduce the specificity of the resource management process used in EGCS.

### 3. Resource Management in the Extended Green Cloud Simulator

The EGCS is a scalable multi-agent system, designed to simulate distributed cloud infrastructures, utilizing (at least in part) green energy sources (for more details, see [4]). In this context, the modelled topology of the cloud is divided into sub-regions (potentially belonging to separate owners), in which the system processes, including resource management, are handled locally in a decentralized manner. Here, let us recall that the developed system is represented by six types of agents.

1. *Client Agent (CA)*—represents the client's job execution demands.
2. *Scheduler Agent (SCHA)*—receives the clients' jobs and is responsible for their scheduling and prioritization.
3. *Cloud Network Agent (CNA)*—represents the regional manager that owns one or more servers, and is responsible for job execution management within a given region.
4. *Server Agent (SA)*—represents a processing unit that receives job requests from *CNA* and is responsible for their execution. It powers the jobs using either: (1) energy coming from connected green energy sources, or (2) "back-up power" produced by a non-renewable, emergency power generator.
5. *Green Energy Source Agent (GSA)*—represents a green energy source that supplies servers with power. It evaluates the available amount of energy, based on the current weather conditions obtained from the monitoring unit.
6. *Monitoring Agent (MA)*—fetches the information about weather forecasts and passes that information to the connected *GSA*.

The hierarchy, and the connections between these agents, are represented in Figure 1. It should be noted that the architecture of the system enforces that the agents are able to communicate directly only with the ones that are "one level below" (and "one level above") in the hierarchy. Therefore, *SCHA* is communicating with *CNAs*, *CNAs* are communicating with connected *SAs*, while *SAs* communicate with *GSAs*.

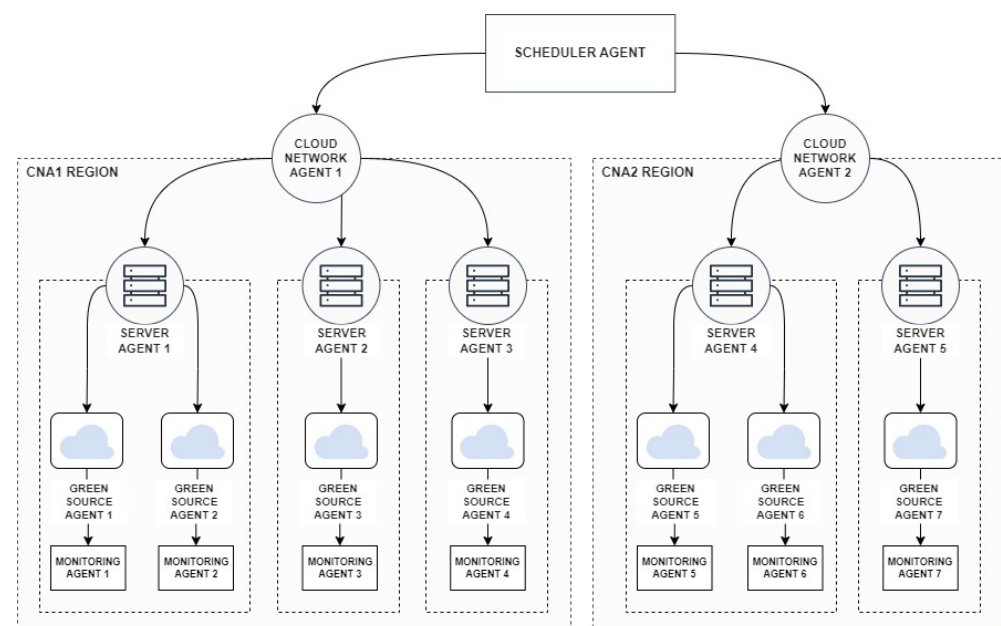
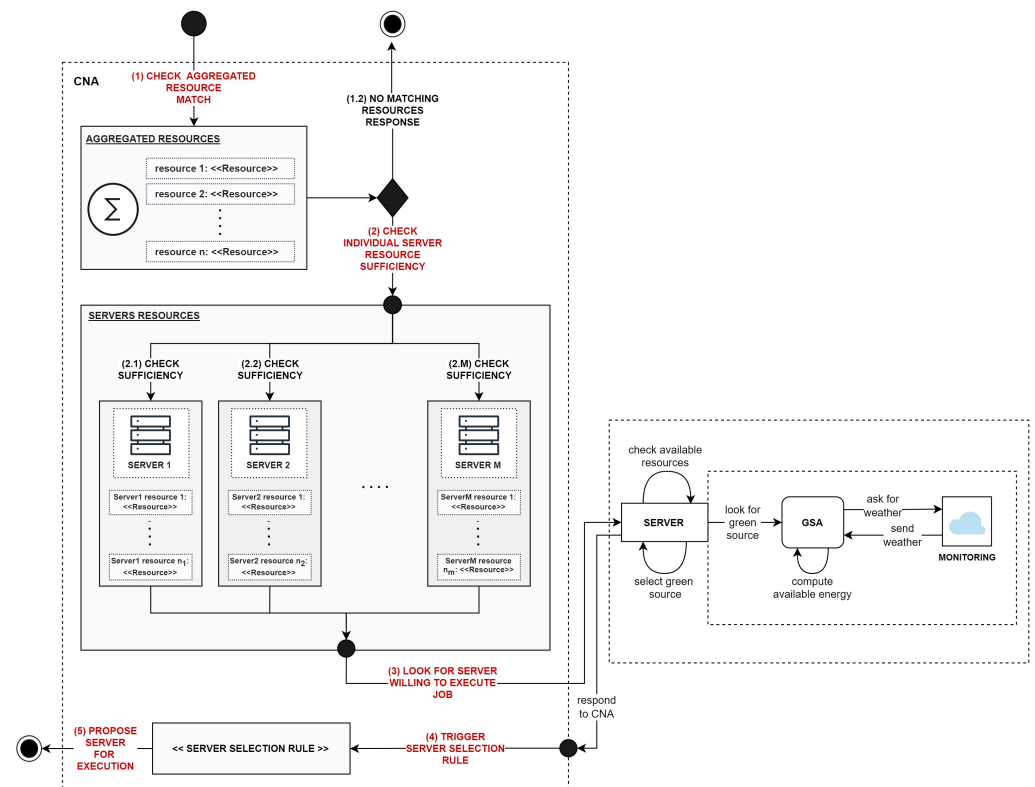


Figure 1. Architecture of the agent system in EGCS.

The information about the resources, available in the system, is stored in two types of agents: *SAs* and *CNAs*. *SAs* are initialized along with the information about the resources

(e.g., CPU, memory) that are available to execute clients' jobs. They have access only to the information about their local resources and do not know the resources of other servers within a given region (as depicted in Figure 1). Their responsibility is to control the resource usage and handle events, such as power overflow, drops in energy supply, or unexpected internal server errors. SAs do not directly manage the resource allocation. The responsibility of assessing which server resources should be allocated for which job lies on the side of CNAs.

CNAs hold information about both (1) aggregated resources available within a given region and (2) the resources of each individual server. The process of selection of the server for the execution of the given job is presented in Figure 2.



**Figure 2.** Process of allocating the resource for the job execution.

When CNA receives a message requesting it to execute a job, first, it evaluates if the aggregated resource pool (i.e., all resources available within a given region) matches the specified job demands (step 1, in Figure 2). In particular, during the process of matching, CNA verifies if the region contains resource types, for which the client's demands have been explicitly stated. For example, if the resource demand was specified for the "CPU" and its attribute "clock speed", then if a corresponding region does not have a server, which provides that information, a given job is rejected. Note, that this process is similar to the matching process between SLAs on the client and the provider sides and could be applied to other types of demand-response systems as well.

In cases when the match was found, for all resources requested by the client, CNA proceeds to the next step (step 2, in Figure 2), which evaluates the sufficiency of resources of the individual servers. The resources must be evaluated separately, for each SA, because of the assumed heterogeneity of individual processing units. Performing this operation locally, within the CNA, improves the scalability of the system, since it minimizes the number of messages exchanged between CNA and SAs.

The group of SAs, with sufficient resources, is asked to evaluate which of them is able to execute a given job (step 3, in Figure 2). The pre-selection of the server cannot happen

without this step, since SAs have to communicate with GSAs, to establish the availability of energy. Recall that CNA cannot access that information directly, due to the hierarchical structure of the system (Figure 1).

When CNA receives the SAs proposals, it proceeds to the selection of SA that is going to be allocated for job execution (step 4, in Figure 2). The selection of SA is facilitated by triggering a corresponding rule (see Section 5.2), responsible for performing a comparison between different SA's offers. By utilizing such an approach, the comparison logic and the criteria can be dynamically interchanged, depending on the current system conditions (e.g., changing energy availability).

Finally, a proposal to execute a job by a chosen SA is passed to SCHA (step 5, in Figure 2), which selects among available offers.

As can be noted, the core of the resource management is carried out in a decentralized manner by the CNAs. Therefore, modification of system resources, conducted by adding/removing the SAs or changing their internal structure, requires updating the knowledge base of CNAs. Moreover, the process of knowledge acquisition/modification involves not only updating information about available resource types but also adapting the approaches used to handle new resources (e.g., evaluating resource sufficiency). Additionally, the adaptation may need to be performed specifically for the selected system region, without influencing the remaining components of the infrastructure. To address that, a resource representation model that facilitates EL has been proposed, as described in Section 4.

#### 4. Generalized Resource Representation Model

The generalized resource model, conceptualized in this work, allows for the on-demand definition and injection of new types of resources without the necessity of changing any part of the existing system logic (i.e., without requiring the system to be rebooted). Moreover, the objective was to support the full heterogeneity of resources, defined in different processing units, system regions, as well as in the client demands. In this context, the following high-level assumptions have been made.

**Assumption 1.** *Each resource type (e.g., CPU, memory) is composed of one or many resource characteristics that describe its individual attributes (e.g., amount, processor type, dual-core support, etc.).*

**Assumption 2.** *The units specifying resource characteristics (e.g., "core" in case of the "amount" of "CPU") are not standardized, and may differ between processing units or client demands.*

**Assumption 3.** *Each resource type that belongs to the given processing unit has to be assigned a unique name (e.g., two resource types with the name "CPU" cannot be assigned to a single processing unit at once). Similarly, within a single resource type, the names of its characteristics must be unique.*

**Assumption 4.** *The characteristics of resources with the same name, assigned to different processing units, may differ (e.g., "CPU" resource of Server1 may have characteristic "SSE3 support" while "CPU" resource of Server2 may not have that defined).*

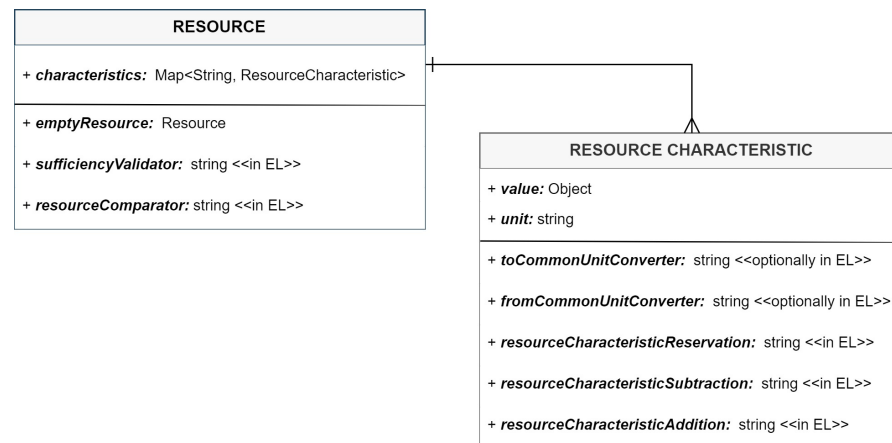
**Assumption 5.** *The way of handling a given resource type is defined by the processing unit that introduces a given resource in the cloud infrastructure (e.g., after the server maintenance or when the server is being added to the system).*

**Assumption 6.** *The way of handling a resource type with the same name, materializing in two separate processing units may differ (e.g., in the evaluation of resource sufficiency with respect to client demands). This specifically refers to cases outlined above, where the same resource types have heterogeneous characteristics.*

The resources have been represented in an object-oriented manner. In particular, the proposed model was made to be compatible with Java-based EGCS. Therefore, it should

be pointed out that due to the use of a specific type of EL (i.e., MVEL [20]) that is only compilable into Java executable code; reusing the proposed model in different languages would require adopting other technological approaches. However, it should be noted that while a different representation of resources may be applied, it does not require changing the conceptual foundation of the proposed approach. Specifically, the approach based on the use of ontologies and semantic technologies, similar to the one described in [21] can be explored. However, this is one of several future research directions.

Based on the listed assumptions, the objects *Resource* and *ResourceCharacteristic*, properties of which have been illustrated in Figure 3, have been defined. Let us now describe each of these objects separately.



**Figure 3.** Model of resource and resource characteristic representation.

The object *Resource* is composed of four properties: *characteristics*, *emptyResource*, *sufficiencyValidator*, and *resourceComparator*. The *characteristics* property represents the heterogeneous characteristics (attributes) assigned to the given resource type. It is defined in the form of a map of key-value pairs  $(k_i, v_i)$ , in which  $k_i$  refers to the unique name of the given *i*th resource characteristic, and  $v_i$  is its corresponding *ResourceCharacteristic* object.

The next property, *emptyResource*, allows defining the representation of the given *Resource*, in the case when it is being “fully occupied”. Such representation can be used by the system, for example, when it performs the aggregation of multiple resources. This property is being set by default to a copy of the given *Resource* object, with the value of the property “amount” (if specified) set to 0. Hence, it is not required to define this property explicitly. It has been added to the model mainly to support exceptional cases, in which the *characteristics* of a given *Resource* are more complex, e.g., when the capacity of the given resource is described by more than one characteristic.

Both properties, described up to this point, allow providing information on the structure and content of the given resource. On the other hand, properties *sufficiencyValidator* and *resourceComparator* specify the way a given resource is handled in the system. Both of these fields are defined as single-line expressions, written in EL, specifically, MVEL. Note that by including these expressions among properties of the individual resources, Assumptions 5 and 6 are met. Moreover, it provides the flexibility of defining the personalized resource handling methods in the case of heterogeneous resource characteristics that may differ with respect to the individual management policies, or the energy-related resource adaptations, performed in different systems regions. Consequently, this implies that the proposed approach can be easily generalized for other types of distributed systems that need to handle different, not only resource-related, system processes in a decentralized manner.

The property *sufficiencyValidator* is a predicate, returning the *true* or *false* value, indicating if a given resource is sufficient to fulfil the specified resource demands. The expression uses two parameters: (1) *resource* that represents a resource-at-hand, the sufficiency of which is being evaluated, and (2) *requirements*, which is another *Resource* object, specifying



the demands that need to be fulfilled. For example, the *sufficiencyValidator* of the form “resource.getAmount()  $\geq$  requirements.getAmount()” would mean that if the value of the characteristic “amount” of the given resource is greater or equal to the requested amount, then the resource fulfils a corresponding requirement.

Similarly, the property *resourceComparator* allows comparing two resources of the same type. It is mainly used when the system compares resources of different processing units, while performing selection between them. This expression behaves similarly to the comparator, which returns values  $-1$  (when the first resource is worse than the other),  $0$  (when resources are considered equivalent) or  $1$  (when the first resource is better). The expression uses parameters *resource1* and *resource2*, which correspondingly represent the resources that are to be compared.

Let us now move to the definition of individual resource characteristics, represented by the object *ResourceCharacteristic*. As presented in Figure 3, it contains seven properties: *value*, *unit*, *toCommonUnitConverter*, *fromCommonUnitConverter*, *resourceCharacteristicReservation*, *resourceCharacteristicSubtraction* and *resourceCharacteristicAddition*.

The property *value* refers to the content of the given characteristic. It is represented as an anonymous object, which means that it can be of any form, including numbers, Boolean values, or text. For example, characteristic “amount” may have *value* equal to 10, whereas characteristic “processorType” may have *value* equal “HX”. Additionally, it is possible to specify the *unit* property that corresponds to the given *value* (e.g., *unit* equal to “cores”).

In this context, since according to Assumption 2, the units in the system are not standardized, the properties *toCommonUnitConverter* and *fromCommonUnitConverter* were defined. They support performing operations on *values* originating from different sources. Here, the understanding of “common unit” by default refers to the measures, such as bytes and cores. However, it is not fixed and can be adjusted according to the specific needs of a given simulation. Similarly to the previous methods, converters can be specified using EL.

The remaining properties that determine the way of handling individual resource characteristics, are *resourceCharacteristicReservation*, *resourceCharacteristicSubtraction* and *resourceCharacteristicAddition*. The first one specifies how a *value* of a resource characteristic should be allocated, for example, for the execution of a client job with given demands. The latter two define basic mathematical operations of subtracting and adding *value* of *ResourceCharacteristic*, for example, when aggregating the resources. Specifying these operations is necessary, since aggregation of resource characteristics into one collection (e.g., combining two processor types to represent that both of them are available within a given system region) may depend on a particular resource management scheme.

Leveraging the EL in the definition of the methods used to handle individual resource types, allowed them to be dynamically compiled during the operation of the system. Consequently, such an approach enables the addition of new methods, or exchange of existing ones, along with the alteration of the resources.

Here, let us also emphasize the model’s versatility. Although it has been used to represent the resource profile of cloud network components and the content of the SLA agreements, the abstractions *Resource* and *ResourceCharacteristic* are arbitrary and, as such, may be applied to other types of systems as well. For example, in the context of the energy communities, the *Resource* can be used to represent smart appliances such as electric vehicle (EV) charging stations. In such case, the *emptyResource* property would specify a state of the station that is already occupied, while *sufficiencyEvaluation* would estimate if maximum charging capacity (*ResourceCharacteristic*) meets given demands. Moreover, note that the model is not limited to energy-related systems only. For instance, in traffic control systems, the *Resource* may simulate traffic signals with cycle length as *ResourceCharacteristic*, while in smart hospital buildings, *Resource* may refer to patient bed with *ResourceCharacteristic* capacity describing their amount in a given ward.

Let us now introduce how the proposed resource model was incorporated into EGCS, to support dynamic knowledge management.

## 5. Dynamic Knowledge Management

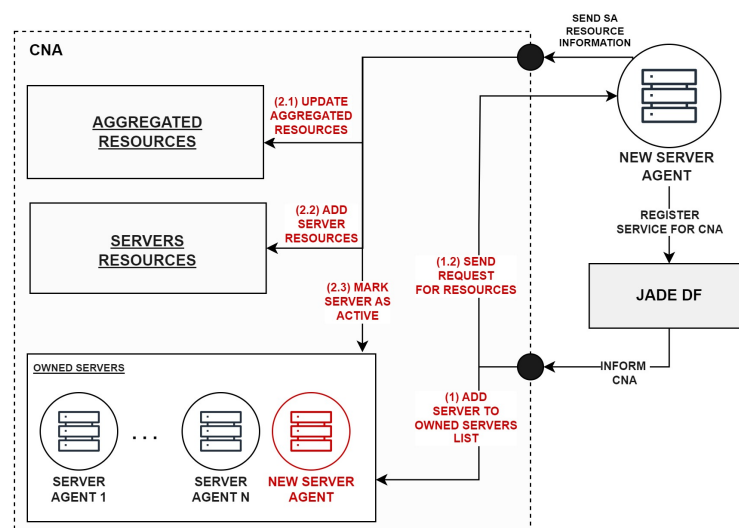
The aspects related to dynamic knowledge management can be divided into two distinct subareas. The first one focuses on the alternation of the knowledge base of system agents, specifically by adding, removing or updating its content. The second area concentrates on the adaptation of the system's behavior and logic, in order to adjust the operation of the system to be able to properly process new information. Both of these areas were addressed in the EGCS as described in what follows.

### 5.1. Knowledge Base Alternation

In EGCS, the adaptation of the knowledge base was related specifically to the cases in which the resources of the SAs (within a given system region) were modified, which required updating the knowledge of the corresponding CNA. In particular, three types of adaptations were considered: (1) adding a new SA to the system region, (2) permanently removing SA from the system region, or (3) updating the resources of existing SA. However, obviously, these energy/resource management anchored adaptations, naturally generalize to other classes of agent systems (e.g., in a smart energy community, it may be adding/removing/upgrading EV charging stations).

The first two cases, in substantial part, were handled by utilizing the possibilities of the agent management services. Specifically, the Directory Facilitator (DF) [22], offered within the JADE agent platform [23], in which EGCS agents were implemented, was used. The DF is a component of the agent platform, which serves as a yellow page directory, allowing agents to register their services. As such, other agents in the system, which look for a given service, can find which agents are able to provide it. Here, particularly useful is the subscription mechanism [24], which allows the agents to be notified whenever a new agent registers or de-registers a particular service.

In the context of knowledge management in EGCS, this mechanism was used to discover newly created, or removed, SAs. In particular, each of the CNAs, in the cloud infrastructure, subscribes to the SA service. Similarly, each of the SAs, when being initialized, registers its service under a given CNA name to which it is to be connected. Due to that, the CNA receives a notification message from the DF, each time the corresponding SA is being added to, or removed from, the system. The process of the discovery of SA, followed by the CNA knowledge base alternation, is depicted in Figure 4.



**Figure 4.** Alternation of Cloud Network Agent knowledge base in response to adding new Server Agent in a given region.

When the CNA receives a message informing about the addition of a new SA, it updates its connections list (step 1, in Figure 4). The connection lists contain addresses

(more particular, global names) of the agents with which the CNA can communicate. Initially, newly added SA is not taken into consideration for the execution of incoming jobs. First, the CNA has to obtain the information about its resources, which is carried out by sending a corresponding request message (step 1.2, in Figure 4). Upon receiving the response, it updates the total availability of resources within a given system region (step 2.1, in Figure 4). It is completed in two steps. First, currently available resource types and their characteristics are compared, using their unique names, with new resources that are to be added to the system (i.e., resources of new SA). The objective is to add to the existing resource structure all missing entries (e.g., adding GPU if it was not previously present). Then, in the second step, the values of each *ResourceCharacteristic*, in each *Resource*, are aggregated by dynamically calling the property *resourceCharacteristicAddition*. Afterwards, the resources of added SA are also put into the individual SA resource list (step 2.2, in Figure 4). Finally, after updating the CNA knowledge base, the added SA is being “activated” (step 2.3, in Figure 4) so that it can be taken into account for the client’s job execution.

The process of the SA removal works in an analogical way. First, the SA sends the information to the CNA to get deactivated, preventing it from receiving new job requests. As such, it can complete all of the ongoing processes, before being removed from the system. After completing the execution of the remaining jobs, SA de-registers its service in the DF, and leaves the cloud infrastructure. When the CNA receives the notification, it removes a given SA from its connection list and, by using the property *resourceCharacteristicSubtraction*, updates knowledge of the total amount of available resources. Additionally, the CNA traverses the aggregated resources to remove the resource types and characteristics, which are no longer available in a given system region. As a result, it optimizes further assessment of resources with respect to the upcoming clients’ demands.

The last type of knowledge base adaptation involves a runtime modification of the resources of an existing SA. Here, it should be noted that the update of the SA’s (hardware) resources can be carried out only when the SA is in a deactivated state so that it does not violate existing system processes. Specifically, after the SA updates its resources, it sends the information to the corresponding CNA, with a new resource structure inside the message content. Upon receiving such a message, during the first step, CNA adapts its knowledge by aligning the total available resources. This is conducted by, first, (1) subtracting the outdated SA resources by calling corresponding *resourceCharacteristicSubtraction* properties, and then (2) by adding new, modified resources, using *resourceCharacteristicAddition* properties. The final step of the knowledge base adaptation involves exchanging the individual SA resources for new ones.

After completing the adaptation, CNA operates on a new knowledge base. The sufficiency of the SA’s resources is being evaluated using the corresponding *sufficiencyValidator* property applied to each relevant resource type. On the other hand, when a given SA is assigned for the job execution, its resources are marked as allocated through the use of the *resourceCharacteristicReservation* property.

Note that the processes, related to knowledge base adaptation and management are conducted by runtime compilation and execution of resource-specific handlers (i.e., *toCommonUnitConverter*, *resourceCharacteristicSubtraction* etc.). Here, resource-specific handlers refer to both: (1) the pre-defined handlers (i.e., common system methods, such as *FROM\_KI\_TO\_BYTES\_CONVERTER*, which the user can pass to, for example, *toCommonUnitConverter* and *fromCommonUnitConverter* by giving their names), and (2) the ones explicitly defined by the user (i.e., by using EL). The second type of handlers supports the universality of the approach and can be used in the context of different types of systems since it is not implementation-dependent and can be fully specified by the user. On the other hand, in the current implementation of the system, the definition of pre-defined handlers is static and adapting it to specific simulation needs may require changing the code base. The dynamic configuration of these handlers is going to be explored in the future.

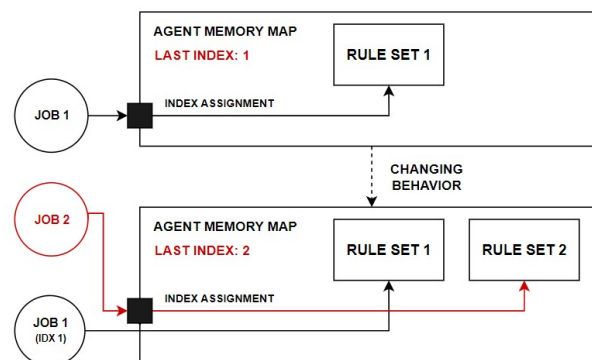
Both types of handlers can be called from the standard, pre-defined behaviors of the agents without the necessity of their modification. Such behaviors do not contain any logic relevant to the current system state, since they are only responsible for executing EL statements. However, such an approach may not always be sufficient to handle the entire process of knowledge alternation. In some cases, the adaptation of the knowledge may require adjusting other behaviors of the system, to maintain the efficiency of its remaining processes. Performing these operations during system runtime involves dynamic adaptation of the agents' behaviors. One way to achieve that is by facilitating the explainable RES, which is described in the following section.

### 5.2. Rule-Based Expert System in EGCS Adaptation

The initial proposal of the utilization of the RES in the dynamic modification of the agent's behaviors has been outlined in [25]. Let us start by recalling the most important concept of the suggested approach.

The work introduces the RES extension to the JADE agent platform, which has been implemented with the use of the framework EasyRules [26]. In particular, the extension provides the implementation of agent-specific rules, facilitating the representation of the logic of the agent's behaviors. The rules are defined separately for each type of agent and are encapsulated in an abstraction called *RuleSet* that specifies the full behavior of the specific agent in the system. The *RuleSet* is referred to by its unique name, and is being stored in the component called *RulesController*, with which each of the agents is being connected. The *RulesController* is responsible for triggering the rules evaluation (i.e., calling *RulesEngine*) that happens every time any of the agent behaviors is being executed. To identify, which rule is to be enacted in which place, the rules are identified by their *ruleTypes*, and possibly, *subRuleTypes* or *stepTypes* (e.g., when executing complex, template-based behavior, such as Call For Proposal). It is possible to specify multiple pre-defined rule sets, information about which is stored in *RuleSetRestAPI* component in the map *availableRuleSets* with keys being rule sets' names.

The *RuleSet*, of each selected agent, may be changed during the system runtime, while performing the behavior adaptation. In particular, it is completed by exchanging some of the rules, identified by the selected *ruleTypes*, to the new ones that correspond to modifications in the system logic. In order to be able to perform such an operation, without violating any of the system processes, each *RuleSet* is assigned its unique rule set index, by which it is being recognized. A rule set exists in the agent's "memory" until a process, in which it is being used, is completed. The *RulesController* stores the information about the last assigned rule index, as well as the map of indexes and rule sets that are being used in a given moment by the agent. Figure 5 illustrates the rule set adaptation, in which the system processes correspond to the execution of individual client jobs.



**Figure 5.** Process of the adaptation of rule set in the agent knowledge map.

In particular, the adaptation process proceeds as follows. When the rules of the current rule set are being exchanged, a new copy of this rule set is created. The new rule set is

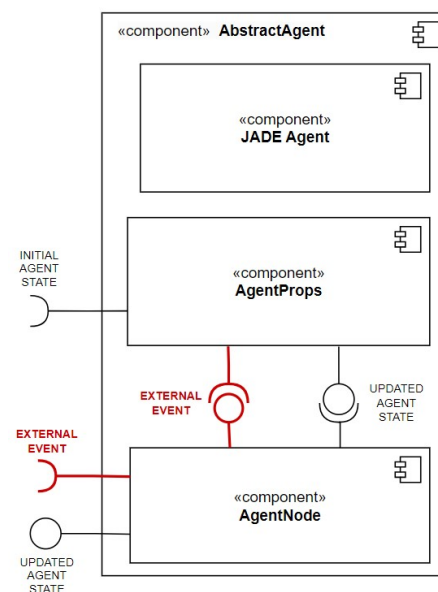
assigned with an index obtained by incrementing the last rule set index, and then it is put into the agent's rule set memory map. Due to this, when a new job is sent to the system, it is assigned to be processed by the rule set that corresponds to the last rule set index (i.e., the rule set containing the latest modifications). Note that the remaining system processes are not disrupted, since the previous jobs, assigned with prior indexes, are being executed using the old rule sets, which are still present in the agent's memory.

The process of rule set adaptation is essential from the perspective of the EGCS, since it is a core of the modification of system logic, initiated in response to the alternations of the agent's knowledge base. The necessity to perform the system adaptation may be recognized either internally (i.e., within the agent) or externally, by sensing the environmental state changes.

The internal recognition of the need of the system adaptation is based on the evaluation of the agent's state, after the knowledge base has been altered. Let us take as an example the alternation of the *CNA* knowledge, corresponding to the case in which new *SAs* resources are being added. Furthermore, assume that the newly added *SAs* operate on new generation processors. After the knowledge base of a given *CNA* is changed, it performs the assessment of new resource availability. It may recognize, that newly added *SA* is going to be more efficient in the job execution. As such, in case of execution of long-term jobs that have higher power and resource demands, its selection would be more optimal. Therefore, *CNA* may decide to adapt the *SA* comparison factors, to align them with the updated knowledge base. This is carried out by creating a new rule, encompassing the revised selection criteria, and exchanging it with the currently applied rule, so that it can be used to newly upcoming client jobs.

On the other hand, the external perception of adaptation is facilitated by sensing the environmental events. Such events impact the agent's knowledge about the current environmental state (e.g., sudden change in weather conditions), and impose changes in the agent's behaviors. Note, that perception of external events is an essential aspect of different kinds of systems (e.g., detection of changes in bidirectional flows of energy in smart grid), and since the concepts proposed in this work are sufficiently generalized, they can also be applied to other application domains as well.

In order to understand, how the environment is being sensed in the EGCS, it is necessary to recall the architecture of the agents, proposed in [25]. Figure 6 presents its most important components.



**Figure 6.** Architecture of EGCS agents.

Here, each agent consists of three components: (1) *JADE Agent*, which encapsulates core agent-specific methods (e.g., behaviors) offered by the *JADE* agent platform,

(2) *AgentProps*, which stores the information about the current agent state, including agent knowledge, and (3) *AgentNode*, which serves as a connector between the agent entity and its environment (e.g., Graphical User Interface).

Note that the proposed architecture may be used in any type of agent system that is based on the JADE agent platform. All components of the model are fully abstract and do not contain any fields related to some domain-specific application. For a more detailed description on each of these components and their interactions, see [25].

From the perspective of the external events perception, the most important component is the *AgentNode*. It is connected to the environment via “sensors” (i.e., connectors, marked in red color in Figure 6) that, from the technical point of view, are implemented in a form of a dedicated WebSocket.

In the context of EGCS, the user can simulate and send the events to the WebSocket using the Graphical User Interface (GUI), which makes it possible to provide data relevant to a specific event. However, implemented functionality is extendable and, after proper reconfiguration, can be used in any kind of agent system (e.g., restaurant recommendation system proposed in [25]) that relies on the agent model architecture proposed in Figure 6.

The retrieval of new events is conducted as follows. When the *AgentNode* receives an external event via the WebSocket, it puts it into a separate *EventQueue* that is placed in the *AgentProps* component. Each of the EGCS agents periodically executes the behavior responsible for retrieving the last event from the queue. If the queue is not empty, and a new event has been registered, the agent triggers a rule responsible for its handling. Correspondingly to the event that was registered, the executed rule modifies the agent’s knowledge about the environment. For example, in case of weather fluctuations, it may lead to switching the *SAs* from the use of the green energy, to the use of the “back-up power” (e.g., a battery or a standard power source).

## 6. Tests of the Implemented Solution

For a better understanding of the presented considerations, let us demonstrate the implemented approach for the three different use case scenarios. (1) Addition of a new type of hardware resource. (2) Modification of existing system resources. (3) Dynamic changes in the prices and individual client preferences. Here, it should be noted that each of these scenarios were introduced primarily to give a context of possible real-life situations in which the proposed approach can be applicable. For the sake of simplicity, the simulations developed in EGCS were focused on resource and knowledge management aspects; hence, the reasoning process of decisions undertaken (e.g., analysis of the market which led to introducing new resource type) in each of the three presented use cases was not explicitly implemented. However, if necessary, it could also be modelled using EGCS by specifying corresponding agent rules.

### 6.1. Scenario 1: Addition of New Hardware Resources

In the first scenario, the increased market demand for a particular hardware resource type entails its introduction in a selected system region. In particular, let us consider the following case.

Assume that in the cloud market, there are two main companies: company ABC and company CAB. The company CAB, among its available resources, owns the GPU. Consequently, since there is a high demand for GPUs in the current market state, CAB generates a high level of profits. Based on that, the company ABC, after conducting a market analysis, determines that owning the latest generation GPU processors, from vendor X, would be highly beneficial. In addition, it confirms that it has sufficient energy availability to support the addition of a new component to its infrastructure. Therefore, it decides to add processors with GPU to its resource portfolio.

The presented business case can be modelled in EGCS, using the agent system whose topology is presented in Figure 7.

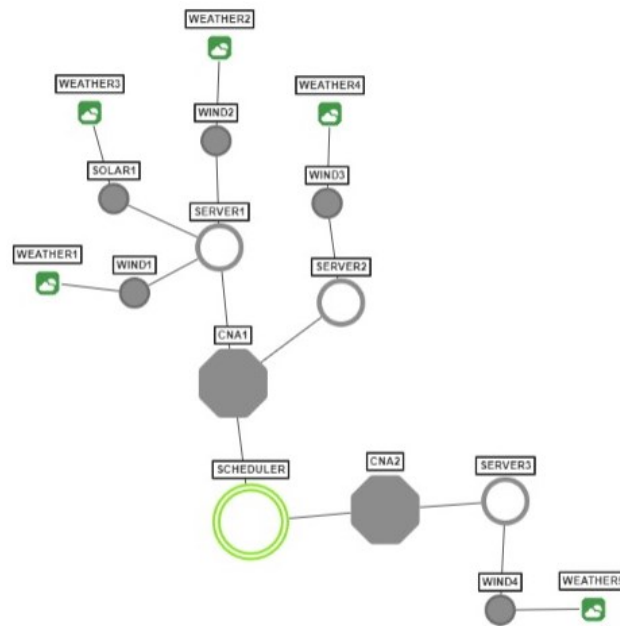


Figure 7. Topology of the system used in Scenario 1.

The system consists of two regions represented correspondingly by *CNA1* and *CNA2*, which refer to two different cloud provider companies (CAB and ABC). In the region of *CNA1* there are two servers: *Server1* and *Server2* (belonging to CAB). On the other hand, the region of *CNA2* includes a single server—*Server3* (owned by ABC). The resources available in each of the *SAs* are presented in Table 1.

Table 1. Hardware resources of servers in Scenario 1.

Server Name	CPU	Memory	Storage	GPU
<i>Server1</i>	1000 cores	2 TB	1 TPB	—
<i>Server2</i>	1000 cores	2 TB	1 TPB	40 kCUDA cores
<i>Server3</i>	1000 cores	2 TB	1 TPB	—

It can be noted that the only server, which allows leveraging GPU is *Server2*. As such, the client jobs, which demand to be executed using GPU(s) can only be assigned to the servers in the *CNA1* region. The fact that client jobs are being consecutively assigned to *Server2* can be monitored and analyzed using the statistics displayed on the GUI.

When the company ABC decides to add a new resource to its portfolio, agent *Server4* is created and introduced in the region of *CNA2*. In order to create the *SA* and specify its resource profile, the user inputs the data to the form within GUI, which later on sends the information using the corresponding WebSocket to the agent platform that initializes the agent container. For this scenario, the characteristics of a newly added server are presented in Table 2.

Table 2. Hardware resources of Server 4.

CPU	Memory	Memory → VRAM	GPU
10 cores	10 GB	256 GB	30 kCUDA cores

Here, note that the *CNA2* did not have prior knowledge about the GPU resource type. Moreover, let us also assume that adding *Server4* is also associated with the introduction of a new memory characteristic—VRAM, which was not previously specified in any component within the entire system. As such, it requires to perform the modification of the current system knowledge.

After adding *Server4* to the cloud infrastructure, *CNA2* received a corresponding notification, as depicted on the system logs in Figure 8. It indicates that *Server4* was able to register its service in DF, a message of which was passed and received by *CNA2*.

```
Server4 INFO [AbstractAgent:166][jobId: ][ruleSetId: ] Setting up Agent Server4@MainPlatform
Server4 INFO [AgentConnector:29][jobId: ][ruleSetId: ] [Server4@MainPlatform] Agent connected with the GUI controller
CNA2 INFO [SubscribeServerServiceRule:52][jobId: 2][ruleSetId: 0] Found 1 new servers in the network!
```

Figure 8. System logs presenting the recognition of Server 4 in CNA 2.

The comparison of the knowledge of *CNA2* before and after the addition of *Server4* is presented in Figure 9. Here, it can be observed that *CNA2* has properly updated its knowledge base (using an approach introduced in Section 5.1). In particular, it added the VRAM resource characteristic and the GPU resource type.

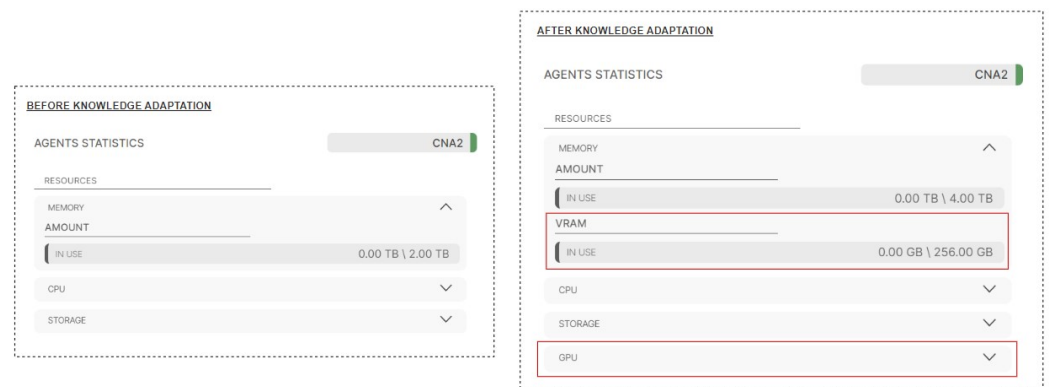


Figure 9. Before and after of the CNA 2 knowledge adaptation.

Let *GPU-Client* be a new client, which is “just” joining the system. The initialization of such clients was also carried out using the corresponding form in GUI. It wants to execute a job with the requirements as presented in Table 3.

Table 3. Hardware requirements of GPU-Client job.

CPU	Memory	Memory → VRAM	GPU
10 cores	10 GB	10 GB	3 kCUDA cores

Since *Server4*, associated with *CNA2* (company ABC), offers a GPU of a newer generation than *Server2* present in *CNA1* region (company CAB), it has a lower job execution price (calculating the job execution price can be specified in a particular rule added as part of the current system’s rule set). Therefore, it was selected for the execution of the *GPU-Client* job as illustrated in Figure 10. Note that a similar effect could be caused by the fact that *CNA2* just joined the ecosystem and does not have (too many) jobs that it is committed to executing. Hence, it can offer a number of other advantages, e.g., the immediate start of job execution and/or faster task completion. While these criteria have not been implemented, their addition would be equivalently simple by modifying existing rules.

Overall, this means that *CNA2* was able to properly evaluate the sufficiency (and, possibly, “attractiveness”) of the resources of *Server4* by utilizing its corresponding *sufficiencyValidator* property that was specified in EL during the *Server4* creation.



EXECUTION INFO	
SERVER EXECUTING JOB	SERVER4
ESTIMATED JOB EXECUTION PRICE	58.67 \$

**Figure 10.** Allocation of Server 4 for the execution of GPU-Client job.

This first use case has demonstrated that the implemented approach is capable of accurately updating the knowledge base of pertinent system agents. Moreover, the provided EL handlers facilitate the dynamic management of new knowledge in processes such as resource sufficiency evaluation, and resource allocation. Let us now move to the second scenario, which will present the alteration of the system knowledge in response to the maintenance of the processing units.

### 6.2. Scenario 2: Modification of the Existing System Resources

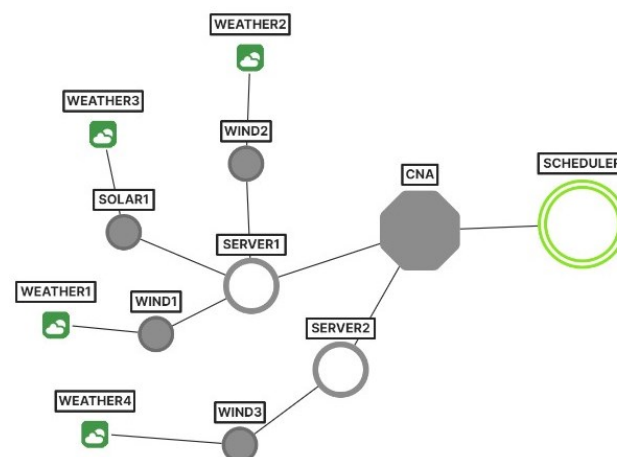
The second use case illustrates a situation in which the energy availability in a particular system region increases. Let us again consider a company ABC, which, in the context of green energy, relies primarily on its owned wind farms.

Assume that in the initial system state, current weather conditions impose limitations on the available energy supply. Therefore, the processing units owned by the company ABC can only support processors of the Intel Xenon 8280 type. The company cannot increase the number of available processors nor upgrade them to the newer generation (with higher efficiency), due to their high power consumption, which is beyond current energy limitations.

However, consider that after some time, the weather in the geographical region of company ABC has changed, leading to an increase in the energy potential. To gain maximum profit from this situation, company ABC decided to add two additional wind turbines in a region of wind farm Z. As a result, it becomes possible to employ more powerful resources in the system.

Furthermore, assume that one of the company nodes (node X) was reaching the limit of its hardware replacement cycle. Therefore, after considering both of these factors, the company made a decision that node X should be upgraded. The node was gratefully taken down from the system, and all of its old 2000 processors were exchanged with new 4000 processors of the Intel Xenon 8380 type.

In the EGCS this scenario can be represented by an infrastructure with a single CNA (company ABC) (as depicted in Figure 11) that has two SAs: *Server1* and *Server2*.



**Figure 11.** Topology of the system used in Scenario 2.

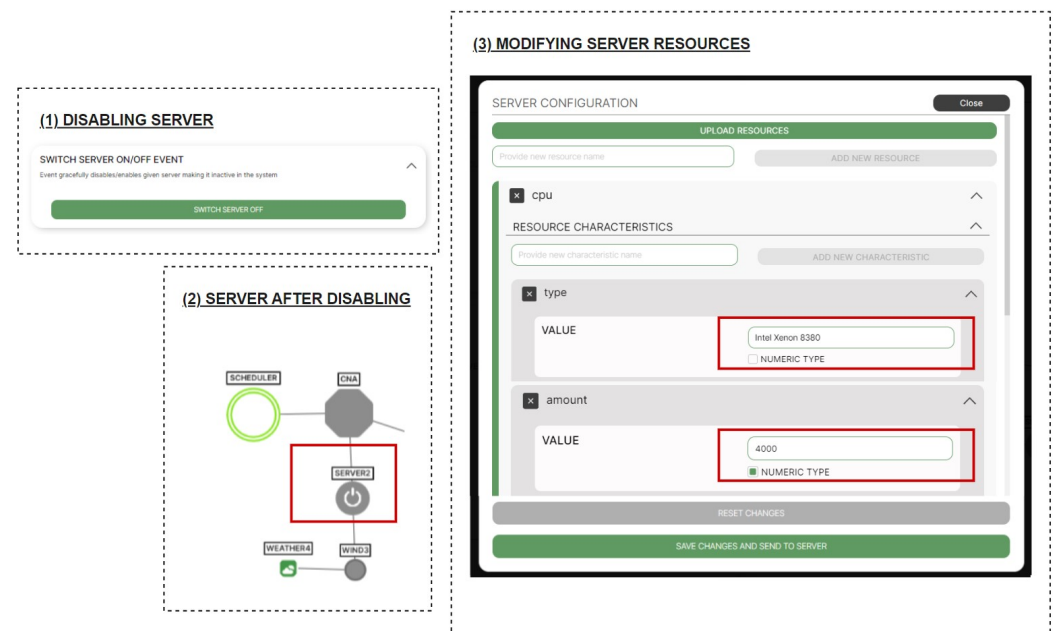
The resources of both of the servers are presented in Table 4. In this case, the CPU of each of the processing units have two characteristics: “amount” and “type”.

Initially, both of the SAs have the same processor type, namely Intel Xenon 8280. Let *Wind3* represent a wind farm Z for which the weather conditions changed and additional wind turbines were added, and let *Server2* be the representant of node X.

**Table 4.** Hardware resources of servers in Scenario 2.

Server Name	CPU		Memory	Storage
	Amount	Type		
<i>Server1</i>	1000 cores	Intel Xenon 8280	5 TB	2 TPB
<i>Server2</i>	2000 cores	Intel Xenon 8280	2 TB	1 TPB

The process of performing the maintenance of *Server2* can be simulated in the EGCS using the GUI as presented in Figure 12.



**Figure 12.** Runtime modification of *Server2* processor.

First, *Server2* is disabled using a dedicated event button (part 1 of Figure 12) available after pressing a *Server2* node on the graph. The event is sent from the GUI to the *AgentNode* corresponding to *Server2*, which triggers a graceful server disabling process, described in Section 5.1. The results of successful *Server2* disabling can be observed on GUI (part 2 of Figure 12). It is indicated by the appearance of the “off” icon on top of the *Server2* node.

In the next step, its resources are exchanged (part 3 of Figure 12) by providing information about new resources in a dedicated resource form. Additionally, the resource form allows the specification of the handlers that are going to be used to manage the new processor type, which, in the case of *Server2*, is depicted in Figure 13.

In particular, associated with the new processor type, (1) addition, (2) removal, and (3) sufficiency evaluation methods have been defined. The processor type addition is achieved by concatenating its name, enclosed in square brackets, to the existing string of other processor type names. On the contrary, the removal is carried out by subtracting a given name from the string. Next, the evaluation of resource sufficiency is performed by the verification if the client has specified a demand for a particular processor type, and if so, if the value of this demand is equal to Intel Xenon 8380. All handlers are specified using EL syntax.

Finally, all of the modifications are saved, and the results of the adaptation are displayed, as shown in Figure 14.

The results indicate that all of the adaptation processes were performed successfully. The resource adaptation request was correctly passed through the WebSocket, to the *Server2*. Afterwards, *Server2* successfully modified its internal resources (according to the process introduced in Section 5.1). And finally, the corresponding CNA adapted its knowledge base, by changing the *Server4* processor type. Figure 15 presents the state of the CNA knowledge before, and after, the *Server2* adaptation.

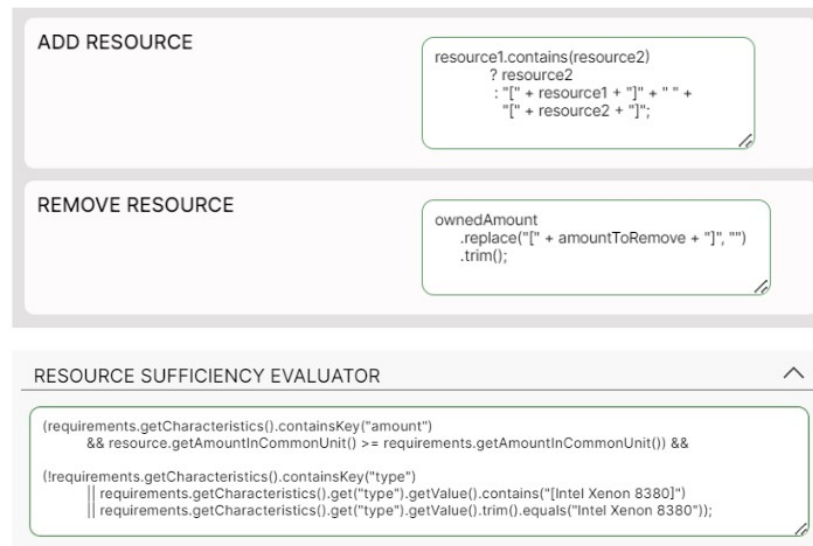


Figure 13. EL handlers specified for managing new Server 2 knowledge.

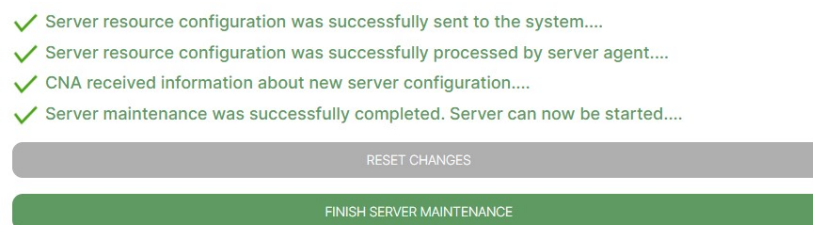


Figure 14. Results of modification of *Server2* resources.

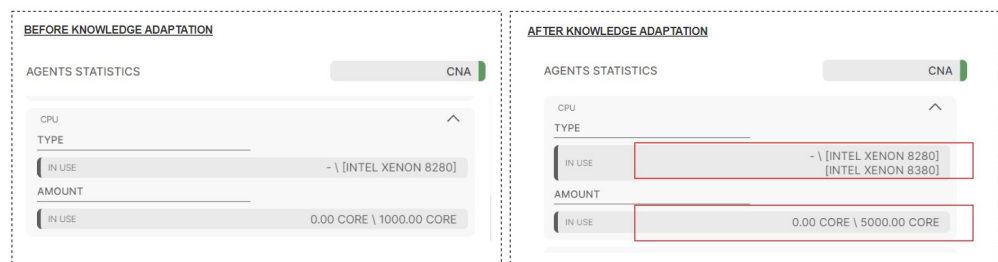


Figure 15. Results of the adaptation of CNA knowledge.

Note that the CNA, by using a predefined *resourceCharacteristicAddition* property, was able to properly manage the resource aggregation, which combined two separate processor types into a single string. Moreover, the number of available CPU cores has also increased.

The two scenarios considered up to now illustrated that the proposed approach can be effectively facilitated in cases when the variability of the simulated ecosystem requires on-the-fly knowledge base alternations. Let us now describe the last example, which

focuses on dynamic modifications to agent behaviors, in response to the indirect effects of knowledge adaptation.

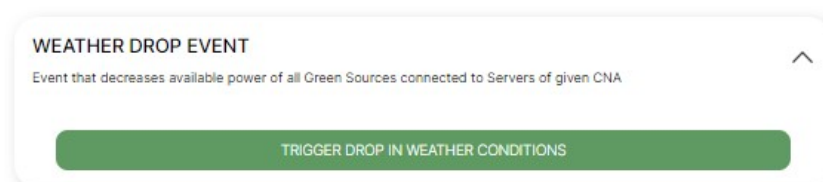
### 6.3. Scenario 3: Dynamic System Pricing and Personalized Client Preferences

The third scenario introduces dynamic system pricing and personalized client preferences. In particular, consider a case in which there are two companies: company ABC and company CAB. Both of them own the resources in two separate geographical locations.

Assume that initially, the environmental weather conditions in the region of company ABC are stable, and hence, the amount of energy produced is sufficient to ensure the execution of the jobs with green energy only. However, after some time, the weather forecast in that region indicates that the weather conditions are going to deteriorate (from the point of view of green energy generation). Consequently, the local centre, responsible for the management of regional resources of company ABC, receives an alert informing about the predicted power drop. In response, the company has to react and appropriately adjust its processing units so that they switch from the usage of green energy to the “back-up power”.

The emergency strategy that is considered (in this use case) is as follows. Before the actual power drop occurs, the servers should be assigned only for the jobs which have a predicted execution time that is shorter than the time until the expected power reduction. On the other hand, during the power drop, the servers should execute only these jobs for which client(s) do not demand the usage of green energy (since, by default, they are going to use “back-up power”). Moreover, due to that, the company also has to adjust the prices for new job execution since it is assumed that the cost of using non-renewable energy is cheaper than the cost of the use of green power.

The topology and resources of the cloud infrastructure, which models the aforementioned scenario, are the same as in the case of Section 6.1. The *CNA1* represents the regional manager of company ABC, which receives an alert about the power drop. The event is triggered using the event button available on the GUI as presented in Figure 16.



**Figure 16.** Button allowing to generate weather deterioration event.

When *AgentNode* of *CNA1* receives an event from GUI, it triggers a rule responsible for behavior adaptation. After the rule is triggered, it puts into the set of facts (i.e., information passed among the rule) the time of predicted power drop received in data obtained from WebSocket. Afterwards, it selects from the pre-defined available rule sets (see Section 5.2), the ones with names *WEATHER\_PRE\_DROP\_RULE\_SET*, and *WEATHER\_DROP\_RULE\_SET*.

The first rule set contains modifications that are to be applied before the power drop occurs. In particular, the rule set is composed of a single rule that defines a method of assessment (in *CNA1*) if an upcoming client job is to be accepted. An assessment mechanism compares the execution time of the job with the amount of time left until the power reduction (information about which is retrieved from stored facts).

On the other hand, the second rule set is to be applied when the power reduction starts. It contains modified rules applicable to servers owned by *CNA1* (i.e., *Server1* and *Server2*). In particular, the rules exclude from the behaviors of *SAs*, the steps of the job execution process and server assignment that involve communication with *GSAs*. After applying these rules, *Server1* and *Server2*, by default, assign jobs to the execution with “back-up power”. Moreover, the *WEATHER\_DROP\_RULE\_SET* also has a rule which modifies the method of pricing in both servers.

The CNA1 firstly, changes its rule set to WEATHER\_PRE\_DROP\_RULE\_SET (see Section 5.2). This is carried out right after the alert event is received. In order to apply a new rule set, CNA1 begins by sending the information to Server1 and Server2 that a new rule set is to be applied. Upon that, Server1 and Server2 inform about the new rule set their underlying green sources (i.e., Wind1, Solar1, Wind2 and Wind3). The CNA1 starts operating on a new rule set only when it receives responses that all agents in its region have successfully adopted a new rule set. Note that this procedure prevents coordination issues that may occur due to delays in communication between agents.

The transition to the second rule set (i.e., WEATHER\_DROP\_RULE\_SET) is scheduled using extended RES JADE-based WakerBehaviour (for more details see [25]). The behavior is triggered exactly at the time of the power drop, using information stored previously as facts. Switching to WEATHER\_DROP\_RULE\_SET is conducted in the same manner as in the case of WEATHER\_PRE\_DROP\_RULE\_SET.

Figure 17 illustrates that the SAs were able to correctly switch their behaviors, firstly, to the WEATHER\_PRE\_DROP\_RULE\_SET, and then to WEATHER\_DROP\_RULE\_SET.

```

Server1 INFO [ProcessRuleSetUpdateRequestRule:45] CNA asked Server to update its rule set to WEATHER_PRE_DROP_RULE_SET!
Server1 INFO [ProcessRuleSetUpdateRequestRule:45] Passing information to underlying Green Sources.
Server1 INFO [RequestRuleSetUpdateInGreenSourcesRule:53] System components are changing rule set to WEATHER_PRE_DROP_RULE_SET!
Server2 INFO [ProcessRuleSetUpdateRequestRule:45] CNA asked Server to update its rule set to WEATHER_PRE_DROP_RULE_SET!
Server2 INFO [ProcessRuleSetUpdateRequestRule:45] Passing information to underlying Green Sources.
Server2 INFO [RequestRuleSetUpdateInGreenSourcesRule:53] System components are changing rule set to WEATHER_PRE_DROP_RULE_SET!

Server1 INFO [ProcessRuleSetUpdateRequestRule:45] CNA asked Server to update its rule set to WEATHER_DROP_RULE_SET!
Server1 INFO [ProcessRuleSetUpdateRequestRule:45] Passing information to underlying Green Sources.
Server1 INFO [RequestRuleSetUpdateInGreenSourcesRule:53] System components are changing rule set to WEATHER_DROP_RULE_SET!
Server2 INFO [ProcessRuleSetUpdateRequestRule:45] CNA asked Server to update its rule set to WEATHER_DROP_RULE_SET!
Server2 INFO [ProcessRuleSetUpdateRequestRule:45] Passing information to underlying Green Sources.
Server2 INFO [RequestRuleSetUpdateInGreenSourcesRule:53] System components are changing rule set to WEATHER_DROP_RULE_SET!
    
```

Figure 17. System logs reflecting the successful exchange of Server 1 and Server 2 behaviors.

The result of the application of the adapted logic used in the evaluation of job lengths in CNA1 is presented in Figure 18. As can be noted, CNA1 properly rejected the job, execution of which would go beyond the remaining time until the power drop.

```

CNA1 INFO [AgentBasicRule:40][jobId: 30][ruleSetId: 1] The duration of job execution is 03 h 35 min.
CNA1 INFO [AgentBasicRule:40][jobId: 30][ruleSetId: 1] It is too long in comparison to time (17 min.) left until green energy drop.
CNA1 INFO [AgentBasicRule:40][jobId: 30][ruleSetId: 1] Rejecting job execution.
    
```

Figure 18. System logs reflecting that CNA 1 rejects the jobs for which execution is too long.

Adaptation of the strategy can also be observed on the cloud infrastructure graph (Figure 19). A “green battery symbol”, displayed on the graph node of Server2, indicates that the “back-up power” is being utilized. Simultaneously, a gray color of the node of Wind3 shows that it is inactive.

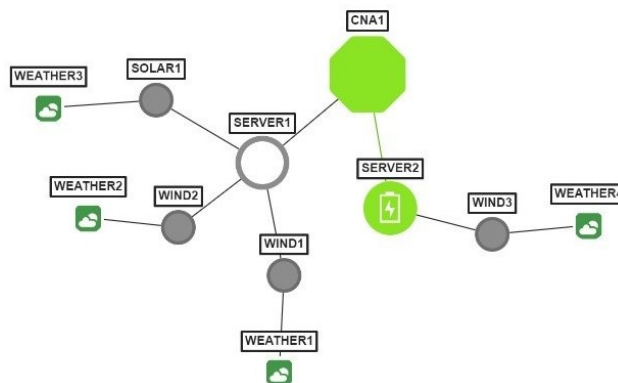


Figure 19. Adaptation of the system illustrated on the cloud infrastructure graph.

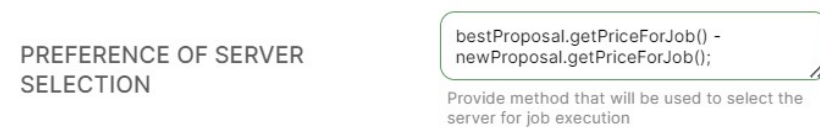
Moreover, as depicted in Figure 20, it can be seen that Server1 and Server2 properly applied the new job pricing criteria (here, they lowered the price) for the upcoming clients’ jobs.

```

Server1 INFO [AgentBasicRule:1120][jobId: 31][ruleSetId: 1] Computing price for the job execution.
Server1 INFO [AgentBasicRule:1120][jobId: 31][ruleSetId: 1] The price has been lowered due to the usage unavailability of green energy.
Server2 INFO [AgentBasicRule:1120][jobId: 31][ruleSetId: 1] Computing price for the job execution.
Server2 INFO [AgentBasicRule:1120][jobId: 31][ruleSetId: 1] The price has been lowered due to the usage unavailability of green energy.
    
```

**Figure 20.** System logs indicating successful adaptation of pricing criteria for Server 1 and Server 2.

Since the price of the job execution is used as a primary criterion for the server selection, most of the upcoming jobs are executed in *CNA1* system region (i.e., by processing units of company ABC). However, consider now a new client, called *GreenClient*, which puts more value on the execution of the job with green energy rather than on the job execution price. Therefore, she is willing to pay more so that his job will be executed in the region of *CNA2* (i.e., the region of company CAB, which still has green energy available). For the purpose of the simulation, the preference of *GreenClient* was specified using the GUI, as presented in Figure 21.



**Figure 21.** Specification of individual client pricing preference.

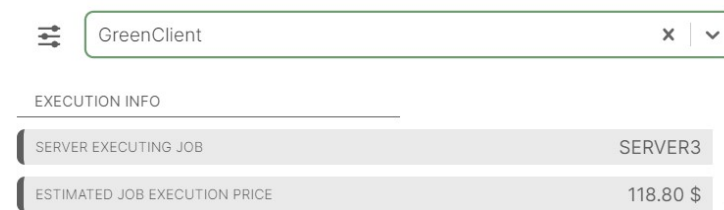
When the *SCHA* received a job execution request from *GreenClient*, it modified its rule set by initializing a dedicated rule, responsible for performing the personalized offer comparison. The system logs, indicating a successful adaptation of *SCHA* behaviors, are presented in Figure 22.

```

Scheduler INFO [ProcessNewClientJobRule:65][jobId: 19][ruleSetId: 2] Client GreenClient@MainPlatform requested to use custom server comparison.
Scheduler INFO [ProcessNewClientJobRule:65][jobId: 19][ruleSetId: 2] Adding rule set CUSTOM_CLIENT_COMPARATOR_GREENCLIENT@MAINPLATFORM
Scheduler INFO [AgentBasicRule:1120][jobId: 19][ruleSetId: 2] Comparing CNA offers using custom comparator (job: 19).
    
```

**Figure 22.** System logs presenting application of custom offer comparator for GreenClient.

After collecting all job execution offers for *GreenClient*, the *SCHA* selected *Server3*, which is the only server that, in the current system state, was not affected by the weather-caused power disruption (offers green energy for job execution). Figure 23 shows the information of the execution of *GreenClient* job in *Server3*.



**Figure 23.** Information about GreenClient job being assigned for execution using Server 3.

Overall, this scenario has proven that the system is able not only to alternate the agents' knowledge base, but also perform the consecutive adaptation of their behaviors. Moreover, the example, in which personalized client's preferences were applied, proves that the proposed representation of demands allows for defining more complicated specifications of the requirements. This can be specifically useful, when considering the construction of more complex SLAs, both on the client side and the provider side. Furthermore, this concept may easily go beyond the considerations of the cloud, e.g., in the context of smart buildings, it could be used to specify any complex requirement, such as personalized workspace adjustments.

## 7. Concluding Remarks

The main goal of this contribution was to provide an in-depth discussion of the proposed approach that facilitates the dynamical management of agent knowledge and allows on-the-fly agent system behavior adaptation. Specifically, an object-oriented model of resource representation, in which the EL were used to specify methods tailored to handle arbitrary resource types, has been proposed. Additionally, by using a rule-based representation of the agent's behaviors, it was possible to further extend the scope of knowledge base modification, to take into account cases in which the system logic has to be adapted within a working system.

The developed model was incorporated into the existing Extended Green Cloud Simulator, to assess its feasibility in facilitating knowledge management of cloud infrastructures. In this context, the proposed method addresses the problems inherent to cloud environments' unpredictability. Specifically, the approach offers a promising solution to dynamically control and optimize the aspects related to energy and power availability.

Moreover, since the developed model is generalized and modular, it can be further used as a basis for other types of systems, the primary concern of which involves resource management. Here, the term "resource" is subjective and, may not refer only to hardware resources. For example, it could be perceived in terms of human resources (e.g., in companies and organizations), natural resources (e.g., agriculture systems), social resources (e.g., social connections and different volunteer groups in community organizations) or other types of SLA contracts specific to given domain fields (e.g., smart energy communities, smart buildings).

As part of the introduced approach, the methodology for facilitating runtime system adaptation was proposed. The incorporation of the EL and the RES enabled the dynamic modification of the system logic, which, in particular, could streamline the resource management process. From the practical perspective, the approach can be incorporated into systems which analyze different strategies and behaviors in time-critical emergency situations. On the other hand, from the scientific side, the presented consideration addresses the area of research in fully adaptable resource management, which, up to now, has not been extensively studied. The majority of the approaches presented in the current state-of-the-art puts a focus on the restricted space of possible modifications that can be enacted on the systems. In particular, they consider either (1) changes in system resources that have been specified beforehand (as in MAPE-K-based resource adaptations) or (2) control knowledge of the system by using parametrized configurations. As such, the methodology introduced in this work broadens this scope by enabling modelling cases in which new, previously unknown knowledge is added to the system during runtime. The term knowledge refers here to both new system resources and new system behaviors. By combining these two, the proposed approach allows for modelling complex cases, which would not be possible by considering only one of these aspects.

Moreover, note that by facilitating the RES, the system's decisions and behaviors are fully explainable. It can be an essential factor used to tailor given system strategies, based on the assessment of results obtained from consecutive simulations. For systems focused on energy and power control, making explainable decisions can contribute to the social acceptance of smart systems.

Potential further enhancements to the developed model will be primarily focused on language independence and extendability. The current implementation can only be used in Java-based systems; therefore, adapting it to other types of systems requires substantial changes in the technology selection. For example, MVEL can be used only when referring to Java, and other EL may behave in a different way during their compilation. Additionally, the current approach may be directly applied only to the agent systems that are based on the JADE agent platform since it is a core of the RES extension. Furthermore, the object-oriented representation does not allow extending the model by adding additional properties, restricting the simulation to a pre-defined set of handlers. It also poses a limitation in terms of representing resource characteristics' values being structures, such as lists, sets, etc.

Moreover, other alternatives to the direct representation of user-defined handlers should be explored. Even after generalizing the model to be language agnostic, usage of any kind of EL requires the user to have a basic understanding of the programming languages. Therefore, a valuable area of research would involve improving the current model in a way that would allow the user to specify preferences and behaviors using more “natural language”. This would, however, require introducing, in the system, a next layer of abstraction. Such a research direction could possibly involve exploring areas of Natural Language Processing (NLP) in terms of translating and interpreting human language instructions [27].

Due to those factors, the future plans are to place more emphasis on the aspect of formal knowledge representation. Additionally, the conceptualization of more user-friendly and human-understandable approaches is to be explored. In terms of the EGCS, the implemented approach is to be used in the assessment and comparison of different allocation strategies [28]. The implementation of the EGCS and described extensions can be found under <https://github.com/Extended-Green-Cloud/green-cloud> (accessed on 1 February 2024).

**Author Contributions:** Conceptualization, Z.W.; methodology, Z.W.; software, Z.W.; validation, Z.W.; resources, S.K.; writing—original draft preparation, Z.W.; writing—review and editing, M.G. and M.P.; visualization, Z.W.; supervision, M.G. and M.P. All authors have read and agreed to the published version of the manuscript.

**Funding:** Work of Zofia Wrona and Maria Ganzha is funded in part by the Centre for Priority Research Area Artificial Intelligence and Robotics of Warsaw University of Technology within the Excellence Initiative: Research University (IDUB) programme (PW-IDUB-SzIR-3-FENIDA). Work of Maria Ganzha and Marcin Paprzycki is partially supported by the European Union’s Horizon Europe program for Research and Innovation through the aerOS project under Grant No. 101069732.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The code of the Extended Green Cloud Simulator is publicly available in open-access at the address: <https://github.com/Extended-Green-Cloud/green-cloud> (accessed on 1 February 2024).

**Conflicts of Interest:** Author Stanisław Krzyżanowski was employed by the company CloudFerro. The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

## Abbreviations

The following abbreviations are used in this manuscript:

EGCS	Extended Green Cloud Simulator
CA	Client Agent
SCHA	Scheduler Agent
CNA	Cloud Network Agent
SA	Server Agent
GSA	Green Source Agent
NLP	Natural Language Processing
MA	Monitoring Agent
RES	Rule-Based Expert Systems
EL	Expression Language
VM	Virtual Machine
SLA	Service Level Agreement
OWL	Web Ontology Language
DAML-S	DARPA Agent Markup Language for Services
MVEL	MVFLEX Expression Language
DF	Directory Facilitator
GUI	Graphical User Interface
RL	Reinforcement Learning



## References

1. Switzerland, M. Carbon-Aware Computing: Measuring and Reducing the Carbon Footprint Associated with Software in Execution. 2023. Available online: <https://news.microsoft.com/de-ch/2023/01/10/carbon-aware-computing-whitepaper/> (accessed on 8 November 2023).
2. Radovanović, A.; Koningstein, R.; Schneider, I.; Chen, B.; Duarte, A.; Roy, B.; Xiao, D.; Haridasan, M.; Hung, P.; Care, N.; et al. Carbon-Aware Computing for Datacenters. *IEEE Trans. Power Syst.* **2023**, *38*, 1270–1280. [[CrossRef](#)]
3. Morais, H.; Vale, Z.A.; Soares, J.; Sousa, T.; Nara, K.; Theologi, A.; Rueda, J.; Ndreko, M.; Erlich, I.; Mishra, S.; et al. Integration of Renewable Energy in Smart Grid. In *Applications of Modern Heuristic Optimization Methods in Power and Energy Systems*; John Wiley & Sons: Hoboken, NJ, USA, 2020; pp. 613–773. [[CrossRef](#)]
4. Wrona, Z.; Ganzha, M.; Paprzycki, M.; Krzyzanowski, S. Extended Green Cloud—Modeling Cloud Infrastructure with Green Energy Sources. In *Proceedings of the Advances in Practical Applications of Agents, Multi-Agent Systems, and Cognitive Mimetics. The PAAMS Collection*; Mathieu, P., Dignum, F., Novais, P., De la Prieta, F., Eds.; Springer: Cham, Switzerland, 2023; pp. 428–433.
5. Reiss, C.; Tumanov, A.; Ganger, G.R.; Katz, R.H.; Kozuch, M.A. Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing (SoCC'12)*, Seattle WA USA, 1–4 November 2012. [[CrossRef](#)]
6. Zhou, N.; Lin, W.; Feng, W.; Shi, F.; Pang, X. Budget-deadline constrained approach for scientific workflows scheduling in a cloud environment. *Clust. Comput.* **2020**, *26*, 1–15. [[CrossRef](#)]
7. Wang, W.; Liang, B.; Li, B. Multi-Resource Fair Allocation in Heterogeneous Cloud Computing Systems. *IEEE Trans. Parallel Distrib. Syst.* **2015**, *26*, 2822–2835. [[CrossRef](#)]
8. Ghodsi, A.; Zaharia, M.A.; Hindman, B.; Konwinski, A.; Shenker, S.; Stoica, I. Dominant Resource Fairness: Fair Allocation of Multiple Resource Types. In *Proceedings of the Symposium on Networked Systems Design and Implementation*, Boston, MA, USA, 30 March–1 April 2011.
9. Kang, K.; Ding, D.; Xie, H.; Yin, Q.; Zeng, J. Adaptive DRL-Based Task Scheduling for Energy-Efficient Cloud Computing. *IEEE Trans. Netw. Serv. Manag.* **2022**, *19*, 4948–4961. [[CrossRef](#)]
10. Maurer, M.; Emeakaroha, V.C.; Brandic, I.; Altmann, J. Cost–benefit analysis of an SLA mapping approach for defining standardized cloud computing goods. *Future Gener. Comput. Syst.* **2012**, *28*, 39–47. [[CrossRef](#)]
11. Kritikos, K.; Plexousakis, D.; Plebani, P. Semantic SLAs for Services with Q-SLA. *Procedia Comput. Sci.* **2016**, *97*, 24–33. [[CrossRef](#)]
12. Kritikos, K.; Plexousakis, D. OWL-Q for Semantic QoS-Based Web Service Description and Discovery. In *Proceedings of the 2007 International Conference on Service Matchmaking and Resource Retrieval in the Semantic Web (SMRR'07)*, Busan, Republic of Korea, 11 November 2007; Volume 243, pp. 114–128.
13. Hummida, A.R.; Paton, N.W.; Sakellariou, R. Adaptation in Cloud Resource Configuration: A Survey. *J. Cloud Comput.* **2016**, *5*. [[CrossRef](#)]
14. Maurer, M.; Brandic, I.; Sakellariou, R. Adaptive resource configuration for Cloud infrastructure management. *Future Gener. Comput. Syst.* **2013**, *29*, 472–487. [[CrossRef](#)]
15. Xu, B.; Peng, Z.; Xiao, F.; Gates, A.M.; Yu, J.P. Dynamic Deployment of Virtual Machines in Cloud Computing Using Multi-Objective Optimization. *Soft Comput.* **2015**, *19*, 2265–2273. [[CrossRef](#)]
16. Yang, L.; Li, X.; Sun, M.; Sun, C. Hybrid Policy-Based Reinforcement Learning of Adaptive Energy Management for the Energy Transmission-Constrained Island Group. *IEEE Trans. Ind. Inform.* **2023**, *19*, 10751–10762. [[CrossRef](#)]
17. Imran, R.; Alraja, M.N.; Khashab, B. Sustainable Performance and Green Innovation: Green Human Resources Management and Big Data as Antecedents. *IEEE Trans. Eng. Manag.* **2023**, *70*, 4191–4206. [[CrossRef](#)]
18. Fraackowiak, G.; Ganzha, M.; Paprzycki, M.; Szymczak, M.; Han, Y.S.; Park, M.W. Adaptability in an Agent-Based Virtual Organization—Towards Implementation. In *Proceedings of the Web Information Systems and Technologies*; Cordeiro, J., Hammoudi, S., Filipe, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2009; pp. 27–39.
19. Ganzha, M.; Mesjasz, M.M.; Paprzycki, M.; Ouedraogo, M. Inserting “Brains” into Software Agents—Preliminary Considerations. In *Proceedings of the Internet and Distributed Computing Systems*; Fortino, G., Di Fatta, G., Li, W., Ochoa, S., Cuzzocrea, A., Pathan, M., Eds.; Springer: Cham, Switzerland, 2014; pp. 3–14.
20. MVFLEX Expression Language Documentation. 2023. Available online: <http://mvel.documentnode.com/> (accessed on 8 November 2023).
21. Wasielewska, K.; Ganzha, M.; Paprzycki, M.; Bădică, C.; Ivanovic, M.; Lirkov, I. Semantic technologies in a decision support system. *AIP Conf. Proc.* **2015**, *1684*, 060002. [[CrossRef](#)]
22. FIPA Agent Management Specification. 2023. Available online: <http://www.fipa.org/specs/fipa00023/XC00023H.html> (accessed on 8 November 2023).
23. JADE Agent Platform Website. 2023. Available online: <https://jade.tilab.com/> (accessed on 8 November 2023).
24. FIPA Subscribe Interaction Protocol Specification. 2023. Available online: <http://www.fipa.org/specs/fipa00035/SC00035H.html> (accessed on 8 November 2023).
25. Wrona, Z.; Ganzha, M.; Paprzycki, M. Dynamic modification of agent behaviours without disrupting a running system. In *Proceedings of the 2023 IEEE 8th International Conference on Recent Advances and Innovations in Engineering (ICRAIE)*, Virtual, 17–19 November 2023.
26. Easy Rules Website. 2023. Available online: <https://www.jeasy.org/> (accessed on 8 November 2023).

- 
27. Gillioz, A.; Casas, J.; Mugellini, E.; Abou Khaled, O. Overview of the Transformer-based Models for NLP Tasks. In Proceedings of the 2020 Federated Conference on Computer Science and Information Systems, Sofia, Bulgaria, 6–9 September 2020; pp. 179–183. [[CrossRef](#)]
  28. Singh, S.P.; Nayyar, A.; Kaur, H.; Singla, A. Dynamic Task Scheduling using Balanced VM Allocation Policy for Fog Computing Platforms. *Scalable Comput. Pract. Exp.* **2019**, *20*, 433–456. [[CrossRef](#)]

**Disclaimer/Publisher’s Note:** The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.