Componentware for Autonomic Supervision Services: The CASCADAS Approach

Component-ware for Autonomic Supervision Services

The CASCADAS Approach

Peter H. Deussen, Edzard Höfig
Fraunhofer Institute for Open Communication Systems
Berlin, Germany
peter.deussen@fokus.fraunhofer.de
edzard.hoefig@fokus.fraunhofer.de

Matthias Baumgarten, Maurice Mulvenna
School of Computing & Mathematics
University of Ulster
Belfast, UK
m.baumgarten@ulster.ac.uk
md.mulvenna@ulster.ac.uk

Antonio Manzalini, Corrado Moiso
Telecom Italia
Torino, Italy
antonio.manzalini@telecomitalia.it
corrado.moiso@telecomitalia.it

Abstract This paper describes two complementary mechanisms for the supervision of large scale and highly distributed systems structured as a cloud of autonomic computing components. The first one is based on the creation of supervision pervasions, for the supervision of clusters of components (i.e., aggregates structurally organized through one or several contracts) implementing specific services in accordance to service-specific management policies. This mechanism is designed as a supplementary service that can be requested by operational components and is structured as an ensemble of self-contained objects that implement an autonomic control loop, which does not require any a priori knowledge on the structure of the supervised system. The second mechanism promotes supervision logic embedded in the autonomic components, which exploit autonomic features and cooperate through dedicated protocols over self-organized overlay networks; this mechanism is suitable for supervising infrastructural (service-independent) functions of autonomic components, and their aggregates. The main contribution of the paper is to define those two mechanisms and to shown that they are complementary, and can be combined to achieve cross-layer supervision.

Keywords — autonomic computing, pervasive supervision, embedded supervision, distributed systems, self-reconfiguration self-adaptation, self-organization.

I. INTRODUCTION

Networks today are composed of a wide variety of network elements that introduce a high degree of heterogeneity. The Telecommunications Management Network (TMN) is a model defined by ITU-T for supervising open systems in a communication network, implementing the fault, configuration, accounting, performance, and security (FCAPS) management areas. The TMN model can hardly meet the requirements of future trends of Telecommunication, Information and Communication Technologies (ICT) and the Future Internet (e.g., emerging of Cloud Computing as well as global pervasive environments). As a matter of fact pervasive diffusion of powerful smart devices for efficient human-computer interaction as well as increased systems heterogeneity are complicating the management and control of the whole network and service infrastructures. As such, there is a need for identifying technology and solutions to simplify the configuration and management of distributed systems whilst, at the same time, reducing the associated operational expenses. This is the main objective of Autonomic Computing [2], as argued in 2003 by IBM's homonymous manifesto. Due to the increasing complexity of large-scale computing systems, computers and applications need to be "*capable of running themselves, adjusting to varying circumstances, and preparing their resources to handle most efficiently the workloads we put upon them*" [3]. This vision took inspiration from the biological characteristics of the human *autonomic nervous systems*, where the autonomic system constantly monitors and optimizes its own status and automatically adapts itself to changing conditions.

As depicted in Figure 1, autonomous operating managers define a control loop for autonomic computing that performs functions associated to the Monitoring, Analyzing, Planning and Executing (MAPE) of processes. Autonomic managers continuously observe the managed system and its environment and handle events on which some (re-)action measures may be executed upon. Sensors and effectors provide observation and control interfaces to the managed elements. Nevertheless, in this model all autonomic "intelligence" is contained in the network of autonomic managers and in which a knowledge base encodes the know-how and practices of human operators.

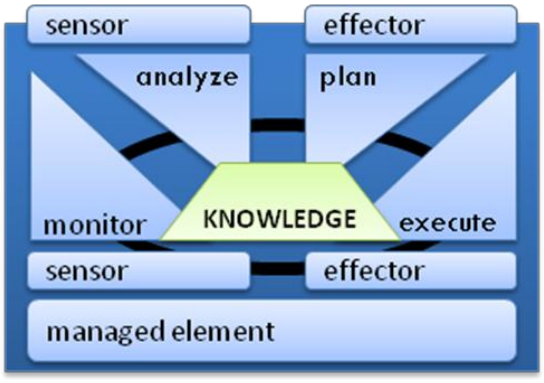

Figure 1. MAPE principle architecture (courtesy to [2]).

This paper is an extended version of the work presented originally in [1]. It outlines an approach for the supervision of highly dynamic and fully distributed systems structured as ensembles of Autonomic Components (ACs), which are

based on two mechanisms: (a) a number of ACs providing a set of basic supervision services, including services aiming at predicting possible evolutions of the *system under supervision* (SUS), that dynamically self-organize to form MAPE-like control loops according to the structure of the SUS and its changes; (b) a set of supervision logic embedded in the ACs themselves exploiting autonomic features and cooperating through dedicated protocols over self-organized overlay networks. The former mechanism is mainly orientated to the supervision of clusters of ACs implementing specific services in accordance to service-specific management policies; on the other hand, the latter one is oriented to define the logic to supervise infrastructural (service-independent) functions in distributed systems that consist of large numbers of ACs, where the same service is provided by multiple instances.

The remainder of this paper is structured as follows. In Section II, the foundations of autonomic communication systems are recapitulated also introducing the notion of Autonomic Communication Elements (ACEs), developed in the context of CASCADAS Project [4], that reflect the framework the proposed supervision component-ware is implemented upon. ACEs are supposed to form services, which are configured in a self-organized way. Section III recalls two self-organization approaches, namely gossiping and rewiring, which will be important in the later course of the argument. Section IV gives a first overview over the two supervision approaches considered in this paper, namely supervision pervasions (Section IV.A) and ACE embedded supervision (Section IV.B). Supervision pervasion is addressed in detail in Section V. Its architecture, components, and interaction mechanisms are described in Section V.A. Long-term supervision is addressed in Section V.B. Section 0 describes how supervision pervasions configure themselves according to the architecture of the targeted system under supervision. An experimental framework for pervasive supervision is explained in Section VI. Section VII concentrates on ACE embedded supervision. Two applications are addressed: load balancing (Section VII.A) and power saving (Section VII.B). Section VII.C presents evaluation results for these applications. Section VIII describes how to combine pervasive and embedded supervision. Section VIII.A describes scenarios in which such a combination will be useful. Sections VIII.B and VIII.C target on a more technical level on how to use self-organization mechanisms to place supervisors, and how achieve a communication between supervisors. Section IX addresses advances beyond the state of the art. Application scenarios are described in Section X. Section XI draws conclusions and indicates further work.

II. AUTONOMIC COMMUNICATION SYSTEMS

Autonomic communication systems are composed of distributed interacting ACs, where an AC is defined to be an entity capable of sensing and adapting to environment changes whilst also performing autonomic capabilities that are related to self-CHOP (Configuration, Healing, Optimization, Protection) through the interaction with other ACs.

Although the general principles of the proposed approach on the supervision of distributed autonomic systems is independent of specific AC models, in the CASCADAS Project [4] they have been designed and experimentally evaluated, by integrating it in the CASCADS ACE Toolkit [5], by considering systems composed of several interacting ACEs. Figure 2 shows the structure of an ACE highlighting individual ACE *organs* (grey components). On the level of a particular ACE, autonomic behavior is achieved through the Facilitator, which utilizes a self-model that describes the business logic in terms of *Extended Finite State Machine*s (EFSMs) [6] capable of dealing with internal and external events, storing and accessing data and invoking task specific functionalities. Several of those state machines can be executed concurrently utilizing an asynchronous event based communication mechanism. The facilitator selects and adapts a set of those state machines in accordance to pre-defined criteria (in particular at ACE startup time) or based on incoming events from other ACEs, internal decisions made during the execution of a previous set of plans, or user interference. The executor organ is then responsible for the parallel execution of plans and their selection.

Self-models and plans implement coordination of a set of elementary ACE internal activities (i.e., they provide a "choreography" for these activities). ACEs therefore comprise a functional repository where the implementations of these activities are stored. Activities (JAVA method calls) make use of so-called *session objects* providing dynamic associative memory to store and to access data as key/value pairs. In addition to this, there is a *global session object,* which can be accessed by every currently executed plan, as well as *local session objects,* which are specific to a particular plan.

The CASCADAS approach takes the perspective that services are provided by (potentially large) ensembles of relatively simple entities realized as ACEs. The functional composition of these entities is done in a self-organized way, using the so-called Goal Needed/Goal Achievable (GN/GA) protocol as the basic means of service discovery within the ACE universe. Through GN/GA dynamic service composition is facilitated by matching a GN to available GAs, which both semantically describe the type of services or functions ACE's desire and offer, respectively. After discovery, ACEs may establish specific contracts among each other to provide for efficient and, more importantly, secure message exchange over multilateral communication channels.

The *gateway organ* is responsible to drive the GN/GA protocol and comprises two core message types:

- Goal Needed (GN): the GN message is broadcasted to all ACEs within a certain ensemble of ACEs. Those ensembles are composed in the following way: At startup, an ACE registers itself with a broker. Several brokers form a network for achieving basic service discovery. Hence, GN messages are distributed by means of this backbone network. A GN message contains a description of a function that is desired by the sending ACE.

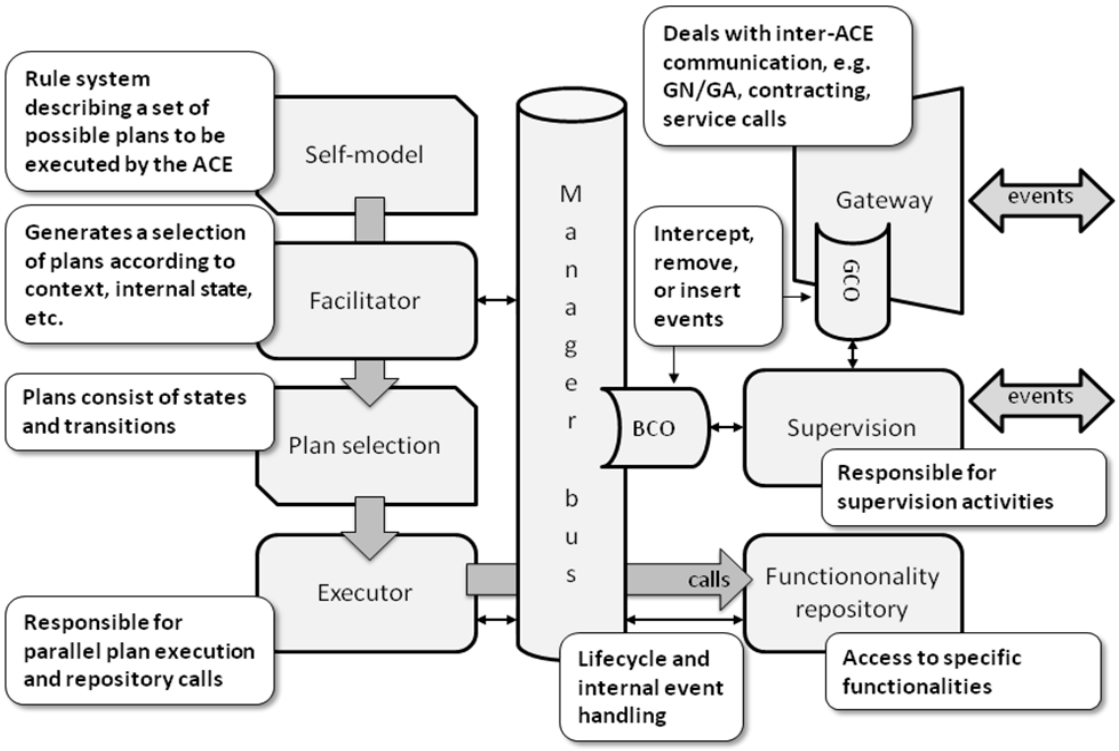

Figure 2: Autonomic Communication Element (ACE).

- Goal Achievable (GA): if an ACE receives a GN message then it compares the included functional specification with its own capabilities, and if it is capable to perform the required function, it replies with a GA (goal achievable) message containing its own address. Since the incoming GN message contains the address of its sender, the broker network is not used to transport the GA answer (the CASCADAS ACE Toolkit uses an underlying agent framework called DIET [7], which provides address-based message communication).

After the exchange of GN/GA message pairs, the initiating ACE selects a partner from the ACEs that have answered its request, and establishes a *contract*. Contracts provide a reliable multicast communication channel incorporating two or more ACEs. Within these contracts specific roles (i.e., symbolic names) are assigned to each of the ACEs involved thus providing a semantically aware way of communications. A message sent over a contract can be addressed either by a specific role (in which case the ACE assuming this role received the message), or is sent without a specific receiver role (in which case all ACEs involved in the contract would receive it). Contract establishment and cancellation is performed through the exchange of so-called contract establishment and contract cancellation events.

Hence the gateway provides the basic mechanisms for service discovery, service composition, and service internal communication. On the basis of these mechanisms, various self-organization algorithms (see Section III) have been implemented.

Before concluding the discussion of the basic ACE architecture, another ACE organ needs to be considered, namely the internal *management bus*. This bus provides ACE internal communication and coordination. Similar to inter-ACE communication, internal processes are also coordinated asynchronously by events. For instance, if the facilitator decides to establish the execution of another plan, it sends a corresponding event containing this plan to the executor. Hence, monitoring the events that travel over the management bus, provides a complete picture of the internal activities of an ACE. We will make use of this property when we describe the supervision organ later on in Section V).

III. SELF-ORGANIZATION

The "cognitive" approaches in Autonomic Computing and Communication (for which MAPE is a paradigmatic example) are opposed by "grass root" approaches, i.e., ideas towards unmanaged self-organization of autonomic systems and services. In the CASCADAS Project, a number of self-organization mechanisms have been defined and experimentally validated to provide efficient and purpose based self-organization. A system comprising a (probably large) number of actors (e.g., ACEs) exhibits complex capabilities that emerge from the interaction of these actors. The actors itself are envisioned to be relatively simple, although no limitation on their complexity is imposed. Depending on their purpose and complexity, they possess a certain number of behavioral rules. In opposite to the

cognitive approach, which has a certain flavor of centralized management, self-organization is – by definition – completely decentralized. Scalability becomes a build-in property of those systems. In this paper, we consider two self-organization approaches, namely *gossiping* and *rewiring*, which are discussed next.

A. Gossiping

This approach has been discussed in detail in [8][9][10]. It provides a peer-to-peer communication protocol, performed by a number of entities organized in a network. The protocol is generic in the sense that it is based on the abstract notion of a "state" and a "state update".

Figure 3 illustrates how this gossiping protocol works for active as well as passive nodes. An entity *A* may assume an *active* or a *passive* role (both behavioral alternatives are executed in parallel). In its active role, the entity waits for a *trigger* (e.g., a timeout, an external event, etc.). If the trigger is received, it selects one of its neighbors *B*, and sends its internal state *S* to *B*. In exchange, it receives *B*'s state *S*'. Then it updates its internal state using *S* and *S*', and awaits the next trigger. In its passive role, it receives information from an active entity sending its own state back also updating its internal state.

```
role active A is
   forever {
        await trigger;
        select neighbour B;
        send S ⇒ B;
        receive S' ⇐ B;
        S ← update(S, S')
   }
```

```
role passive B is
   forever {
        receive S' ⇐ A;
        send S ⇒ A;
        S ← update(S, S')
   }
```

Figure 3: Gossiping protocol [9].

A number of applications of this protocol have been discussed in [9]. To give a simple example, consider a sensor network measuring relevant environmental parameters (e.g., temperature, light intensity, etc.). To compute the average of these values and to diffuse this "new knowledge" to each element in the network, we can define a state as the currently measured parameter value, and a state update as the computation of the average of these values. It is easy to see that the state values of all elements of the network converge towards the average of all measured values.

For our purposes, we define a *state* as ACE internal state (i.e., a state of an EFSM including those session entries that are of interest for the supervision task in question).

B. Rewiring

Another approach for self-organization explored by the CASCADAS Project is graph rewiring [11]. In opposite to gossiping (which does not alter the structure of the underlying communication network) rewiring aims at changing the neighborhood-relationship in support of the formation of clusters of entities that are related by certain criteria. To this end, consider a match criterion between the elements of the communication graph. The idea is to alter the structure of the graph in a way that matching entities are directly connected.

Figure 4 visualizes this on a conceptual level. Here, a node *a*, acting as an *initiator*, requests a matching node from one of its neighbors *m*. This node acts as a match-maker and selects a matching candidate *b* (if such a node exists). In this case, the match-maker establishes a connection between *a* and *b* as indicated on the right side of the drawing. In the negative case, it will report back to a, which will try another neighbor as a match-maker or, alternatively, will wait until the global graph structure has been altered and a suitable match-maker/candidate pair becomes available. Note that, depending on the concrete problem to be solved by the rewiring, the edge between *a* and *m* may be deleted or may be maintained for further processing).

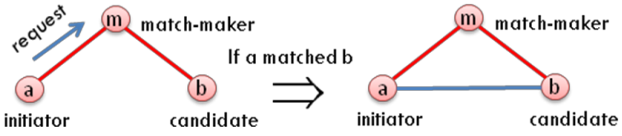

Figure 4:Rewiring

Similar to the gossiping approach, which is parametric in the notion of states and state updates, the rewiring algorithm depends on the selection of a match criterion. For instance, consider a load-balancing scenario. Here, the goal is to gradually connect all nodes that are capable to process the same type of jobs. Thus, in this case the matching criterion used is the node type. This will allow any node to, over time, find an increased number of nodes to which it can distribute the load it cannot process.

In the CASCADAS Project, rewiring has been applied to organize the clustering of ACE based services according to various criteria such as service logic, communication pattern, load distribution, fault tolerance, etc. (see Section VII). With regard to the discussion of communication mechanisms available for ACEs (Section II), the communication graph structure corresponds to contracts a particular ACE is involved in. Within the context of ACEs, graph rewiring (edge deletion and insertion) is realized through the establishment, modification, and cancellation of communication contracts.

IV. SUPERVISION FOR ACE-BASED SYSTEMS

Local and global control loops enable a component (or an aggregate of components) to react in an autonomous way to changes of the internal state and to events propagated by its environment. This feature can be fruitfully applied to implement supervision features for controlling the behavior of a component, and for actuating corrective or optimization measures when a critical situation is detected, such as a failure state, a performance problem, or a configuration error. Such autonomic capabilities should be able to address several supervision areas, such as FCAPS at different levels, from single ACEs to groups/clusters of ACEs, e.g., implementing specific services.

The approach for the supervision of distributed autonomic systems proposed in this paper is based on two mechanisms:

- *Supervision pervasions*, for the supervision of clusters (i.e., an *aggregate* of ACEs providing a common service, which is structurally organized in one or several overlapping contracts) of ACEs implementing specific services in accordance to service-specific management policies.
- *ACE embedded supervision*, for the supervision of infrastructural (service-independent) functions of ACEs and their aggregations through contracts. In opposite to supervision pervasions, which take place on the level of aggregates, embedded supervision is realized on the level of singular ACEs and their contracts.

A. Supervision Pervasions

Supervision pervasions as visualized in Figure 5 are clusters of ACEs implementing specific services according to service-specific management policies. ACE based supervision is performed through supervisors, which provide supervision as a supplementary service: As all services in the CASCADAS framework, supervisors are implemented as an aggregation of ACEs, each of them offering basic supervision functions for filtering, correlating, and elaborating events provided by the supervised ACEs, and for autonomously elaborating corrective or optimization measures. The configuration of supervisors (which is named *pervasion* because it is architecturally not separated from the SUS but *pervades* it) is dynamically set-up and updated (e.g., through self-organization techniques) to align itself to the evolution of configuration of the ACE ensemble under supervision. In this way, the ACE based supervision is able to provide autonomic control loops without any a-priory knowledge on the structure of the (ACE-based) supervised system.

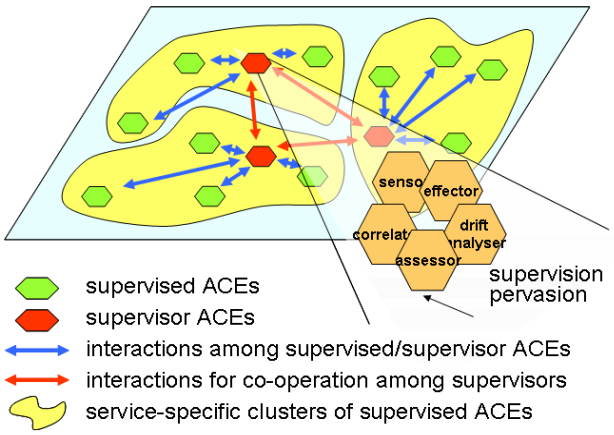

Figure 5: Supervision pervasions

The supervision service is programmable in order to implement the service-specific management policies for the monitoring and the management of groups of ACEs. Supervised ACEs are grouped into meaningful clusters, each of them controlled by a supervisor. A supervisor is able to collaborate with the supervisors of other ACEs clusters.

B. ACE Embedded Supervision

ACE embedded supervision as depicted in Figure 6 can be used to supervise the basic functions of ACEs and the interaction among them (e.g., the active contracts). This mechanism is aiming at performing supervision activities in a highly distributed way, by exploiting the self-adaptation features of ACEs, and self-aggregation of data exchanged among them. Local supervision logic, executed by each ACE, cooperates by exchanging data through a self-organized overlay [11], e.g., by means of gossiping protocols as described in Section III.A.

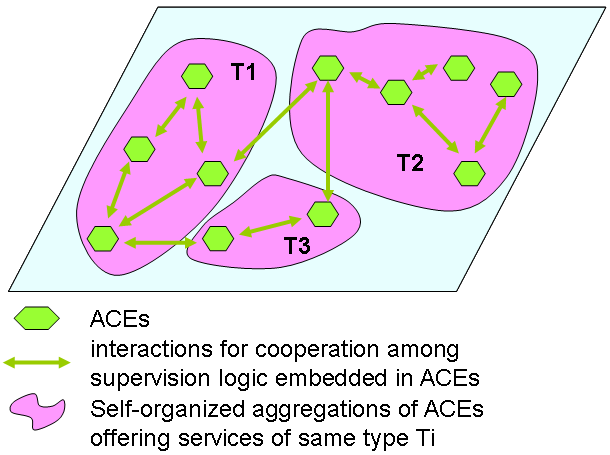

Figure 6: ACE embedded supervision

Thus this mechanism can create an approximated knowledge of the (dynamically changing) global properties of the overall system, and use them to perform local supervision decisions. This mechanism is suitable to supervise systems where multiple ACE instances provide the same "type" of services (possibly implemented differently), e.g., replicated for performance or fault tolerance reasons or deployed on end-users devices. The distribution of the logic and the interactions though overlay networks guarantee the development of scalable and robust algorithms.

As detailed in Section VIII, *ACE embedded supervision* is complementary to and synergic with *Supervision pervasion* approach. In fact, supervision pervasions are more suitable for the supervision of service specific clusters of ACEs, whereas ACE embedded supervision is oriented to the fine grain monitoring and optimization of system generic properties (such as self-repair, load distribution, and energy consumption optimization) of distributed systems. In particular, it is suitable for supervising distributed systems structured as huge amount of ACEs, where each type of service is provided by multiple instances, for instance due to redundancy and performance requirements, or as deployed on end-users terminals.

Moreover, the two mechanisms can fruitfully co-operate. For instance, a supervision pervasion must be able to react to events that are produced by ACE local supervision logic when it is not able to resolve certain situations. For examples, when the fault management supervision logic embedded in an ACE is not able to replace a failed contract for a service of a given type *T*, then it can inform its supervision pervasion, which elaborates and returns the corrective actions by e.g., reconfiguring the internal plan in order to use a service of an alternative type.

V. SUPERVISION PERVASION

Conforming to the CASCADAS architecture, supervision capabilities are realized as ACEs and as such offer a supplementary service to any ACE or ACE ensemble. A supervisor itself is an ensemble of ACEs, which are dynamically (re-)configured through service discovery and, subsequently, self-organized via the establishment of contracts, which define the relation among individual components. Thus, basic supervision functions can be provided as default services to allow for e.g., filtering and the processing of events that are produced by the supervised ACEs, and for autonomous elaboration of corrective and optimization actions.

Figure 7 depicts the architecture of a supervisor, where the use of the interaction protocols is indicated through individual arrows. For the sake of simplicity, only one instance of each component is shown, while in practice there will be always a number of supervised ACEs, sensors, effectors, etc. In addition to the basic supervision functions, supervisors may include other components in order to perform, e.g., predictions, contingency planning, etc.

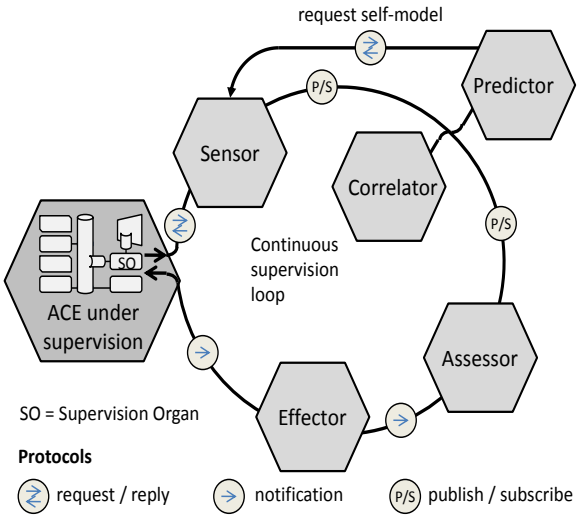

Figure 7: Organization of supervision components

This service centric perspective allows formulating and implementing supervision infrastructures, which go beyond the supervision of singular ACEs towards a more flexible and dynamic set of autonomic control loops, which are able to adjust their own structure and function to the structure of the SUS, thus forming enhanced service configurations, which are able to secure themselves against faults, performance problems, etc. To emphasize the close relationship and organizational similarity between the SUS and supervisor, those infrastructures have been named *supervision pervasions*.

Therefore, the supervised ACEs and the supervisor ACEs work synergistically realizing a supervision pervasion in the following way:

- The supervisors' autonomic behavior co-operates with and complements the autonomic behavior of the supervised ACEs.
- The structure of the supervisor is interwoven with the one of the SUS and as such is also aligned with its changes.
- Supervision can be performed based on internal and external stimuli as well as in accordance to service-specific management policies.

A. Architecture and components

As introduced in the previous section, the pervasive structure of a supervisor enhances the SUS by "completing" it through ACEs that implement basic supervision capabilities that are based on interfaces for observation and control between the SUS and the supervisor. As discussed in Section II, ACEs are built upon an event-driven architecture where effectively all processes are controlled by events that are propagated through the internal communication bus (for intra-ACE communication), and the gateway (for cross ACE communication, i.e., GN/GA based discovery and contract based message exchange). Hence, observing and controlling the bus and the gateway provides sufficient information to understand and to influence all ongoing processes within an ACE. Effective observation and control can be performed by interception, removal, and insertion of events sent over these communication channels.

1) Interfacing the SUS

The interface between the supervision layer and the SUS is realized by so-called supervision checker objects (SCOs), which are implemented as *gateway checker objects* (GCO), and *bus checker objects* (BCO) and which:

- provide basic filtering functionalities to identify events that are of interest for supervision;
- can be used to query specific information about the supervised ACEs such as its current internal state;
- provide control functionalities to steer the internal processes of ACEs;
- establish a communication channel to sensors and effectors.

SCOs can be deployed at run-time by a supervisor (the deployment process is handled on both sides by the supervision organ of an ACE; as sensors and effectors are ACEs too, they have supervision organs as well). Therefore, this mechanism provides a very flexible and generic way to set-up task specific interfaces for monitoring and actuation.

A number of functions have been implemented to provide various monitoring and actuation capabilities as summarized next:

- Insertion, deletion, and modification of events travelling over the management bus and the ACE gateway.
- Interrogative requests to retrieve the current state (the session objects), as well as the currently running plans, and the self-model of an ACE. Moreover, mechanisms are available to obtain the contracts an ACE is involved in.
- Denial and enforcement of transition execution within the executor. As discussed earlier, ACE plans are essentially extended finite state machines comprising states, variables vectors (i.e., session objects), and transitions that lead from one state to another while modifying variables also calling other repository functions.

2) Communication mechanism

As described below, a number of protocols are available for the various components to facilitate communication within an ACE based supervision pervasion:

- Notifications are unacknowledged messages used to distribute data between two components.
- Request/reply pairs are used to actively retrieve specific data.
- Publish/subscribe communication scheme is based on the provisioning of a certain *topic*, where a topic is a symbolic concept that is used as a category identifier for certain types of information. A supervision component, which is interested in a certain topic, broadcasts a subscription within the supervision pervasion. Any ACE providing this topic adds the requestor to a subscription list for this topic. If it obtains information matching a given topic, it distributes this information to all subscribers in the subscription list.

3) Supervision Pervasion Components

Sensors link the supervision system with an ACE under supervision by deploying SCOs into the supervised ACE and by establishing a dedicated communication channel for monitoring, their goal is to translate events delivered by the SCOs into the internal message format used by the supervision infrastructure, and to distribute them to other components of the supervision infrastructure, in particular to correlators and components that deal with the long term supervision of ACE's and associated services.

Correlators are responsible to aggregate monitored data from distributed sources and to correlate them with other information in order to extract meaningful indicators of the current condition of the SUS.

Predictors provide long-term supervision functions, which are discussed in more detail later on in this section.

Assessors make assumptions on the current (or future) system health based on the output of correlators, and invoke associated effectors if necessary.

Effectors are responsible to distribute contingency actions to the SCO of the various ACEs under supervision, where they are used to steer the execution of the ACE under supervision.

The application case described in Section VI uses simple arithmetic operations and pre-defined reaction patterns for analysis and actuation, but since all components mentioned above are generic and programmable, more complex correlation, assessment, and actuation approaches can be defined. For instance, the reactive part can be extended by additional components such as planners. A detailed discussion of such functionality is however outside the scope of this paper; the interested reader is referred to [12].

Note that because a supervision system is implemented by a set of ACEs, which are implemented in particular supervision organs; it is possible to extend the supervision activities to the supervision system itself by using the very same mechanisms as described above.

B. Long-term supervision

An important characteristic relevant for the autonomic self-evolution of pervasive services as well as their supervision mechanisms is that of prediction. If available at all levels, predictive capabilities could lead towards proactive ACE's and ACE interfaces that go some way towards the provision of calm environments, as envisaged in [13]. Such mechanisms provide the ability to predict the possible future contexts as well as interaction between stakeholders within and between ACE's. This means that, based on the observation and analysis of past behavior and the use of predictive reasoning, an ACE could predict its own future states for various aspects of e.g., its own operational environment to either guide itself to a more optimum state or, if necessary, to prevent unwanted or dangerous situations before their actually occur. Mechanisms of such supervision require a temporal aspect to be taken into account that can otherwise be discarded. That is that individual concepts and properties of a system under supervision need to be monitored over time. Similar, past behavior needs to be observed and analyzed in order to predict future situations. In relation to ACEs, relevant concepts to be analyzed include the detection of drift behavior as well as the modeling, monitoring and prediction of events, states or situations an ACE can step into or reach in the future. Thus, the general objective of long-term supervision components can be stated as to observe, model and analyze all available numerical and symbolic concepts over time, in order to predict future properties, behavior and situations of each ACE as well as any ACE ensemble. This would ultimately allow counteracting any form of behavior that could potentially lead to critical or undesired states before they are actually reached or before they occur. Additionally, it would, over time, identify the "best" execution plan for a given ACE or ACE ensemble, which can be actively used to guide new instances of a known service.

For that, three types of supervision components have been devised that are each capable of performing a long-term supervision task. Each service has been realized as an ACE

itself, following the self-similar design of the ACE platform, and can be requested by a supervisor in the same fashion as any of other supervision services, i.e., via GN/GA protocol.

Drift Analyzer (DA) allow facilitating flexible long-term supervision by analyzing and forecasting numerical concepts in relation to certain boundaries a system should operate in or, alternatively, an ideal state of operation that reflects the most optimum performance of a system or a systems component. Such numerical (or ordered symbolic) properties can refer to business goals or to operational parameters of a supervised ACE and its environment.

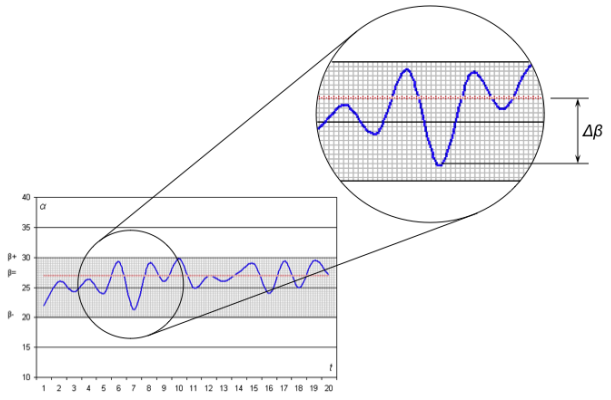

Figure 8: Drift Analyzer

The rationale of DA's is that the underlying concepts are volatile in nature and as such are likely to constantly change over time and that there is a strong desire to keep them within certain operational boundaries that reflect the correct or optimum behavior of the system under supervision. Thus, detecting if a parameter suddenly or slowly drifts towards its operational boundaries or away from its ideal state of operation would allow to deploy corresponding measures that counteract an observed effect before the system can reach a more serious, unwanted, state. As depicted in Figure 8, drift analyzers constantly monitors and analyses the numerical property it is attached too comparing it to the defined boundaries β, β^+ and the desired ideal state $\bar{\beta}$. If drift is detected a dedicated planner component may be notified to invoke actions to counteract the detected drift.

Event Predictors (EP) predict the time window in which a certain event is most likely to occur next. As depicted in Figure 9, an EP is monitoring the occurrence of past events and computes a static as well as dynamic prediction around which a given type of event may reoccur. The static service provides the mean distance between events as its prediction, whereas the dynamic service is based on the time of calculation/request, thus taking into account the time elapsed since that last event has been registered.

This service is of particular interest for periodic services and it would allow for both, the validation of correct behavior (e.g., an event should occur periodically) or, alternatively, for the detection of fraudulent behavior (e.g., if an event occurs outside of its predicted time window). Another useful application for such a service is the priming of ACE's, ACE ensembles or the allocation of resources in anticipation of a given event. For instance, if a frequently occurring service requires specific information or a certain amount of system resources then these could be acquired or reserved, respectively, in time for the event to occur.

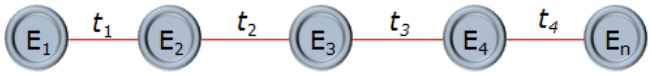

Figure 9: Event Predictor

State Predictors (SP) aim at observing and predicting the execution logic of ACEs as represented by their self-models. In particular, they allow (a) monitoring the execution of ACEs, (b) to build an execution model based on these observations and (c) based on an observed state change, to predict potential next states as well as the most likely transition(s) that lead to the predicted state(s). Note that, instead of a single possible "solution", a state predictor provides a ranked list of candidates as well as a ranked list of transitions that are associated to each candidate state. Thus, a given planner or executer component can evaluate the recommendation before executing them. Depending on their configuration, state predictors operate based on the observation of past and/or mass behavior as inspired by [14]. For instance an SP could monitor the execution of all ACEs (services) of a certain type and would, over time, construct a model that reflects how this particular type of service operates. In particular the constructed model would reveal the collective behavior of the type of service that is under supervision. If a new instance of this type of service is requested then the associated predictor component could provide recommendations of how the service should perform or behave, which would be based on the successful execution of past instances of the same service type. This would allow preventing illegal or dangerous behavior of an ACE and would also allow for the optimization of service execution in the long term. Based on the ACE concept and the associated self-model two distinct types of predictors have been devised.

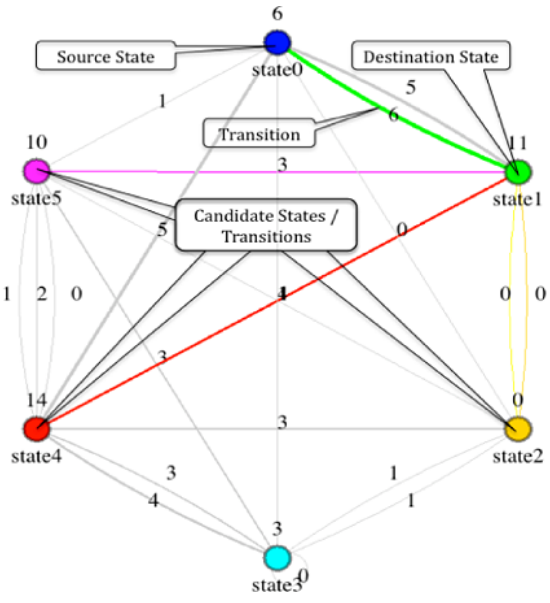

Figure 10: Meshed State Predictor (MSP)

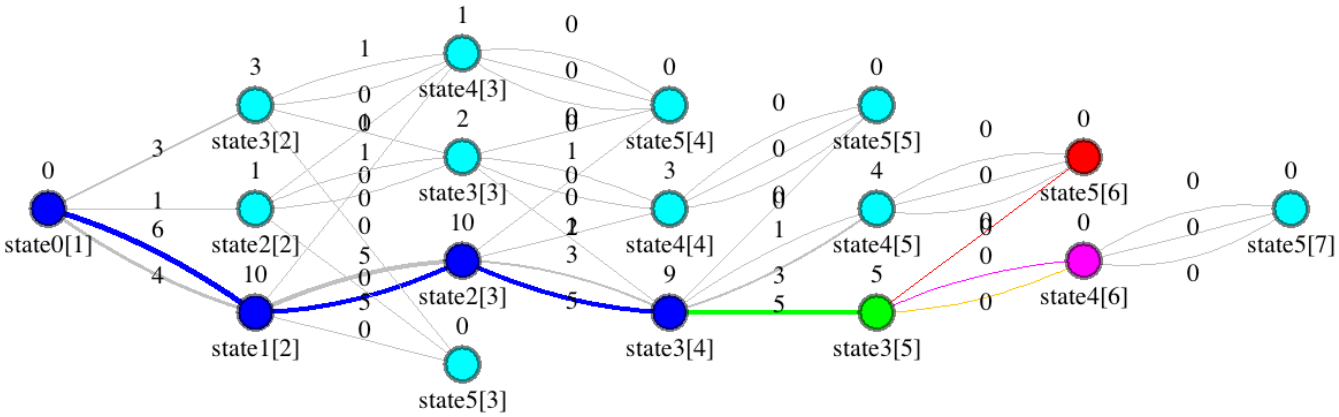

Figure 11: Directed State Predictor (DSP)

Firstly, a meshed state predictor (MSP) that only takes into account a single state change thus discarding all preceding activity and, secondly, a directed state predictor (DSP), which takes into account the entire execution path of a given service from a defined start to a defined end state. Based on this, the former is more suitable to validate stateless operational behavior as defined by individual states and transitions of a given self-model whereas the latter is more appropriate to model more specific behavior or cross ACE interactions, which are likely to be state dependent and, as such do require a more rigid model where the path of execution is relevant. For instance C can only occur if A → B has occurred first. In knowledge discovery terms this corresponds to associative pattern for the MSP and to sequential pattern for the DSP.

Figure 10 and Figure 11 show an example of the same execution model as constructed by the MSP and the DSP respectively. Each model contains a number of nodes, the transitions between them and the occurrence property that reflects how often a state has been assumed or how often a transition has been traversed. As can be seen, the full path of execution is maintained in Figure 10 whereas the model shown in Figure 10 discards this type of information. The rationale for this is based on the stateless method invocation of specific as well as common service functionality of ACE's and relates to an undirected graph in which a collection of states may form short sequences to reflect individual service execution rather than long-term business goals. In fact only the current state transition is of interest, independently of the current state of execution.

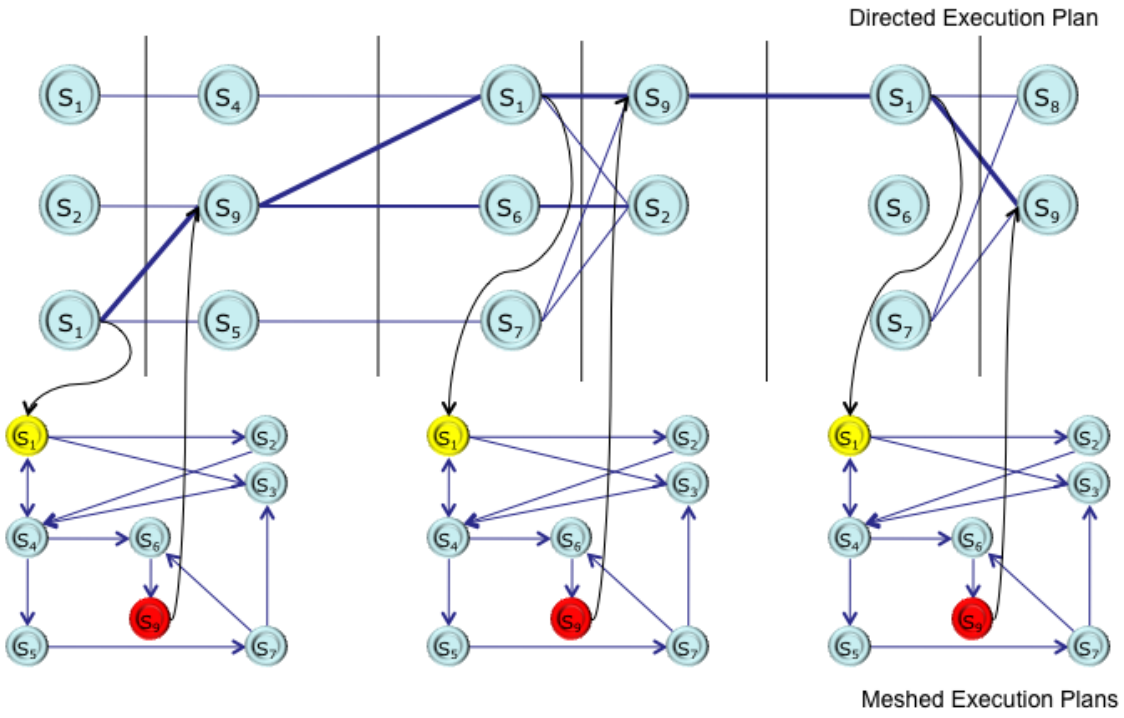

Figure 12: State Predictor Pervasion

Such a loose model of observation is ideal for short lived, stateless services where previous conditions are irrelevant. Independent of the path prefix that is maintained by the DSP, both models take into account only a single state change, which is reflected by the triple source state – destination state – transition traversed. Based on this and the properties of each state/transition, which reflect how often they have been visited or traversed in the past by the same service type, the likelihood of states / transitions to occur next can be computed. Thus an SP indicates how a service is most likely to continue based on its past behavior or based on the behavior of other instances of the same service. Such information can be used directly to, e.g., initialize subsequent states, provide system guidance, detect system violations etc.

For ACE ensembles that provide more complex services at runtime, a combination of both models is feasible that utilizes the meshed execution model at ACE level whereas the directed execution model is used for cross ACE interactions. This is depicted in Figure 12 where the top shows the execution plan of a directed state predictor, which, at certain states, steps into the execution plans of one or more meshed state predictor(s). Depending on configuration each execution plan can relate to the same or to different ACE's or ACE instances thus adapting to the distributed nature of the underling SUS. Notable for such a configuration is that no interaction between individual predictor components is required as this is embedded within the logic of the ACE ensemble that provides the overall service, which is equally reflected by the configuration of the supervision pervasion. However, how feasible such a configuration is within the context of very complex ACE configurations still needs to be explored in detail and is subject of future work.

Another aspect to be evaluated is to aid the prediction process with other parameters that relate to e.g., the current state of an ACE, its environment or its business goals. For example, during normal operation, the best path of execution of a given business process may be reflected by A→B→C with A, B and C referring to system nodes where a given service is executed on. If load on C is high, then this candidate could be demoted in favor of a candidate where load is low, e.g., A→B→D; thus ensuring that overall system load is evenly distributed. Such functionality could easily be incorporated into the state predictor by using the normalized and inverted load-factor of each node as a weight factor that influences the importance of each node within the execution model, which is normally only reflected by the occurrence property.

C. Automatic Configuration of Pervasion

The configuration of a supervision pervasion is done in a number of steps:

Contracting: Supervision is a supplementary service to be used by ACE ensembles that provide service(s) to a user (or another ACE ensemble). To facilitate supervision, the first step involves contracting all components (sensors, effectors, correlators, etc.) to be involved in the supervision pervasion. This is done via a special *controller* ACE, which commits a supervision contract with the SUS. Then the controller discovers, via GN/GA, the remaining ACEs, and sets up another contract for communication within the supervision pervasion. Finally, it obtains relevant configuration information, required to establish individual supervision checker components that provide a specific monitoring and control channel as discussed next. Note that the discovery of ACE's and the contracting is a service that is part of the common functionality of an ACE and as such is provided by default.

SC Deployment: Supervision checker objects (GCO and BCO) are deployed by sensors into individual ACEs that are to be supervised. To this end, a temporary contract is established between a sensor and an ACE at which an SC object is to be deployed. The SC object itself is sent as part of a specific message, which is handled by the supervision organ of the ACE to be supervised. After deployment, each SC object establishes a connection to a sensor as well as an effector ACE.

Subscription: A publish/subscribe based interaction mechanism is used as a general communication paradigm within the supervision pervasion. For instance, correlators as well as state predictors subscribe to information *published* by sensors, where the specific selection of topics obviously depends on the SUS and on the supervision task to be performed. Hence, the publish/subscribe protocol provides a data-flow driven group communication schema, where groups are defined by topics.

Re-configuration: Changes in the architectural structure of the SUS can be detected in several ways. The most generic approach is to use the BCO of an ACE to intercept events that steer the reconfiguration (contract cancellation, discovery, new contract establishments, etc.) on the internal communication bus, and to forward this information via sensors to a dedicated correlator. In some cases it is however easier to simply notify the supervisor ACEs about an ongoing reconfiguration, which in turn will adapt to this change.

The supervision pervasion reacts to the reconfiguration of the ACE ensemble under supervision by performing reconfiguration operation on itself. In particular it removes SC objects from ACEs that are not longer part of the SUS, and deploys new SCs to ACEs that are part of the new SUS ensemble. Moreover, it adapts its internal structure to reflect the new architecture of the supervised ensemble using the mechanisms (contracting, subscription) described above.

Termination: Supervision activities are terminated (or suspended in the case of long-term supervision) when the ACE ensemble under supervision decides to break the supervision contract, which is usually the case when the service contract grouping this ensemble is terminated. The controller ACE notifies all components of the supervision pervasion, and breaks the contract between them. As for the long-term supervision components, a contract can be re-instantiated to the same statefull supervision object thus allowing for the continuing observation of execution plans.

VI. EXPERIMENTAL FRAMEWORK FOR SUPERVISION PERVASION

A prototype of the proposed pervasive supervision mechanism has been implemented. It is integrated within the CASCADAS ACE Toolkit and it is available as open source [4]. Within the prototype, each of the supervision components is realized as a separate ACE. In particular, the supervision library includes a set of generic ACEs, one for each component thus providing basic as well as long-term supervision features (i.e., sensor, effector, correlator, assessor, drift analyzer, planner, state predictor, event predictor). In order to accommodate for systems specific requirements, these components can be further specialized by extending these components. More importantly, the supervision library provides relevant communication and interaction protocols (request/reply, notification, publish/subscribe) to set-up and (re-)configure a supervision pervasion and its components.

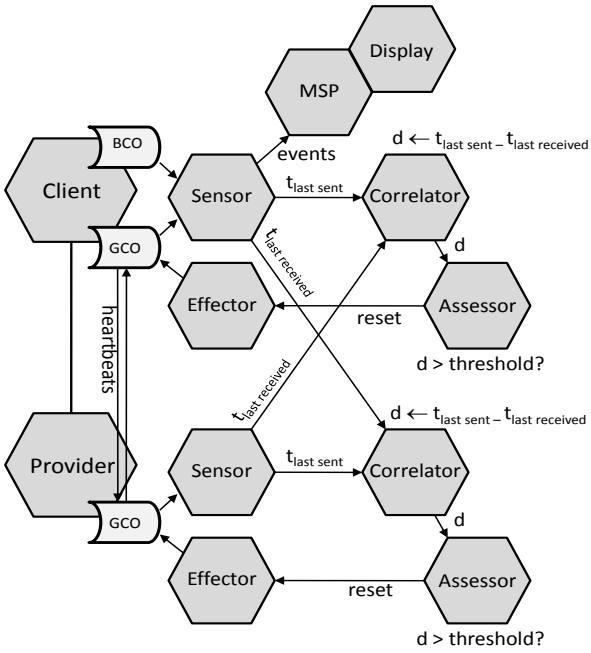

Figure 13: Dynamic Reconfiguration Scenario

The described supervision approach has been applied to supervise a video service implemented as a set of distributed ACEs. Pervasive supervision has been introduced in order to handle failures of ACEs implementing the video client and one of several available video providers as depicted in Figure 13. The goal for this scenario is to autonomously reconfigure the supervision pervasion (that is the SUS and the supervision components) if the Provider-Client relationship develops a fault.

Subject of such supervision activity is the liveliness of the contract between the client ACE and the provider ACE. Supervision is done by issuing an exchange of heartbeat signals between these two ACEs, hence, if the contract is malfunctioning in one or both directions, this fault can be detected by comparing the time stamps of sending and receiving a heartbeat signal. Heartbeats are handled by GCOs that are deployed into the supervised ACEs. Hence, the liveliness validation mechanism is transparent to the supervised ACEs where dynamic reconfiguration of the supervision pervasion takes place if the provider changes. In this case, the scenario depicted in Figure 13 is automatically adapted to work with the new service contracts. Relevant functions to inject specific GCOs / BCDs into the new SUS are provided via the ACE framework.

For long-term supervision, the MSP computes the probability of subsequent states based on observed state changes within the execution logic of the SUS (in this case, the video player ACE). Hence, in the above scenario predictions are related the probability that a certain channel will be selected and the probability that a fault during transmission will occur due to a contract problem. These probabilities are computed on the basis of sending the events of self-model transitions, which occur when the channel changes or timeouts for individual video channels occur, to the MSP ACE. The former is of particular interest, as it would allow a system to determine the channel that has been selected most in the past, which in turn could be selected if the currently selected channel becomes unavailable or if the selection procedure develops a fault.

We have not yet carried out a detailed analysis of the performance of the pervasive supervision approach. For general ACEs however, such an analysis is available, which allows inferring results also for the specific application described in this paper. Resource consumptions can be expressed in several terms:

Number of Thread: The number of threads an ACE runs in stand-by mode is around 11. Changing from an idle to a working mode the number of threads is increased to 18, with additional three threads for each new contract (which are needed to handle incoming and outgoing messages). Moreover, each parallel plan executed by an ACE adds another thread. An ACE involved in a supervision pervasion contributes to two to five contracts and runs two to six concurrent plans. Hence, in terms of threads, the performance burden seems to be significant. It should however be noted that a comparable supervision system, which is not based on ACEs would require resources too. In particular, the establishment of communication relations (corresponding to contracts) and working threads (corresponding to parallel plans) would be comparable. The main difference is that those threads are executed by ACEs and not by an external supervisor.

Memory Consumption: According to the experimental evaluations performed on the CASCADAS ACE Toolkit [15], the memory consumption turns out to be largely linear compared to the number of ACEs a system comprises and the number of established contracts (i.e., the memory needed for input queues). Hence, similar remarks on a comparison with supervision systems that are not ACE based apply.

A detailed analysis of response times in relation to the current application case is not yet available. Nevertheless, SCOs have been designed to infer with the internal event propagation of an ACE in the least possible way (monitoring is non-blocking). Hence, the performance burden added to an

ACE by deploying SCOs can be expected to be not significant.

To summarize, the impact of adding a supervision pervasion to an ACE configuration depends clearly on the architecture of this pervasion. Implementing feedback loops for each ACE (or pair of ACEs as in our application example) by means of a complete supervision configuration clearly increases the resources used in an unreasonable way. Supervision pervasions become (at least in term or resources) meaningful if larger configurations of ACEs are considered as SUS. For micro-level supervision the approach described in the following Section VII is more appropriate.

VII. ACE EMBEDDED SUPERSION

ACE embedded supervision is based on the self-adaption of behavior according to local "supervision" logic. It processes their internal state (e.g., active contracts, load information), and information received from their neighbors in "supervision" overlays. These overlays are used to communicate (through gossiping protocols) information about given ACEs that are relevant for distributed supervision algorithms. They are continuously adopted to efficiently achieve interactions among local supervision logic and as such guarantee scalability by keeping the interactions local thus avoiding information flooding.

As depicted in Figure 14 two types of "supervision" overlays are used to facilitate ACE embedded supervision:

- Achieving (T): interconnecting the ACEs, which provide a service of type T.
- Contracting (T): interconnecting the ACEs with an active contract to a service of type T.

The construction of the "contracting (T)" and "achieving (T)" clusters makes use of the rewiring algorithm described earlier in Section III.B.

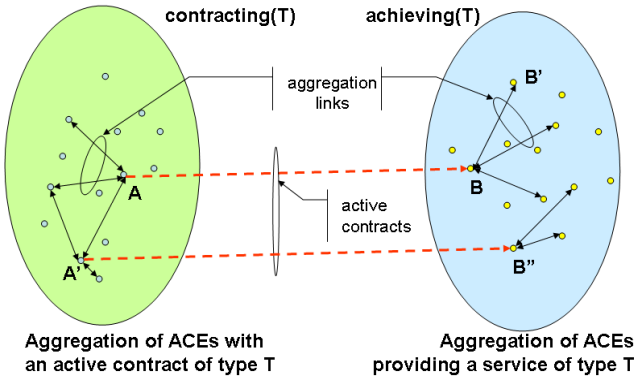

Figure 14: Self-organized overlays for service type T

This algorithm relies on the possibility to reconfigure the contracts of a given SUS and implies that services are state-less. If an execution context is needed to achieve multiple-request transactions then it can be passed as an argument within the request and response message. In this way, an ACE A with an active contract to another ACE B offering a service of type T, can replace B with another ACE that provides a service of the same type T, without losing the current context of execution. That is that information relevant for the current execution of a service are maintained, which allows such a service to continue execution instead of restarting it.

To achieve this, the ACE logic is enriched with specific supervision logic (described as a set of self-models), which process the ACE internal state and the information exchanged with the neighbors of the supervision overlays.

The following sections describe and evaluate ACE embedded supervision algorithms for load balancing, and power saving. An additional algorithm for handling contract failures outside the scope of this paper but is described in detail in [16].

A. ACE Load Balancing

The Load Balancing algorithm (LB) implements load distribution policies, in a fully distributed way. LB enables the migration of load, in terms of contracts, from ACEs that are overused to ACEs that are underused. Such underused ACEs "invite" their neighbors to the achieving (T) overlay in order to redirect some of their contracts, which is depicted in Figure 15 and facilitated as follows:

- when B' is underused, it informs all its neighbors in achieving(T);
- if B, one of B' neighbors, has a high load, it replies to B' by accepting the offer, and, negotiates with B' the load to be transferred. This negotiation mechanism effectively prevents B to transfer too much load to B';
- when A sends a request to B through one of the contracts that have been transferred to B', B informs A to redirect its contract to B'. Then A destroys the contract with B and send to B' the request to create a new contract.

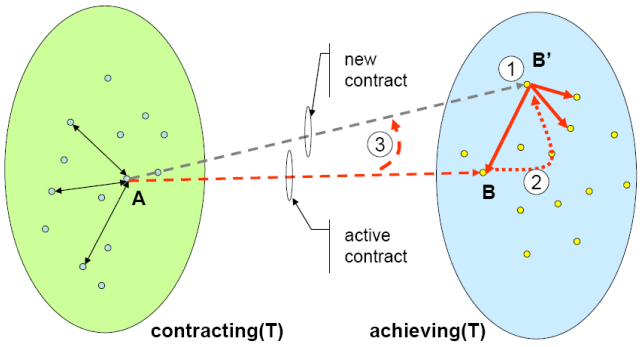

Figure 15: Load balancing supervision logic

B. Power Saving

The power saving algorithm extends the LB algorithm and assumes that each node in a distributed computing cloud is associated with an ACE that is in charge of its supervision,

and as such of its optimized use within the cloud. This supervisor ACE is able to monitor the load of the node and the contracts to the services it provides. The logic stems from the fact that a node in stand-by consumes much less energy than a node that is idle. Moreover, the energy consumed by an active node is proportional to its load, but with a small difference between an idle and a fully loaded state [17]. Thus, a group of servers with low utilization is a waste of energy considering that the same work could be executed by a smaller number of servers. In this case, the remaining servers could be put in stand-by, thus reducing energy consumption.

An ACE A supervising an underused node could contact its neighboring nodes in achieving (T) to take its entire load. If this succeeds, the node monitored by A can go in stand-by to save energy. Vice versa, it could be woken-up by a node that has a high workload if this node is not able to find any currently awake node with sufficient resources available.

Measures to force a node into stand-by logic may be executed by an ACE A that is monitoring an underused node, according to the following:

- if A gets a random neighbor B in the achieving (T) overlay, and
- if B is able to take all the load of A;
- then A transfers its load (contracts) to B and goes in stand-by.

Accordingly, a wake-up "call" is executed by an ACE B that is monitoring an overloaded node, which is, according to the set out LB policies, not able to transfer its load to any other node currently active. Then B selects a neighbor in stand-by mode, if any, and transfers part of its load to it.

To avoid node oscillation, a woken-up node has to wait for some period before it can go into stand-by again. Moreover, to reduce the number of failures in looking for a neighbor to wake-up, an overloaded node has to wait for a fixed time after a failed attempt in waking up a node, before performing a new one.

C. *Evaluation of the algorithms*

The algorithm for load balancing (LB) and its extension with power saving policies (LB+PS) have been implemented and evaluated through simulations by using the 'breve' simulation environment [18].

The simulations were executed on a set of 6400 nodes, each of which supervised by an ACE. Each node is initialized with a random number of queued tasks (in the range of 1 to 1000) and a number of contracts (ranging from 1 to 60). During each simulation cycle, each contract generated a random number of task requests (between 0 and 10), and each node executed 200 tasks.

Two thresholds were defined as follows: a node with less than 400 pending requests is considered underused, while a node with more than 2000 pending requests is considered overloaded. In order to avoid that an underused node immediately becomes overloaded, the total amount of contracts assigned to it should not exceed the number of 40 after receiving contracts from overloaded neighbors.

The energy consumption of a node is computed in "energy units", according to the following formula, which are aligned with the considerations set out in [17]:

$$\text{energy units} = \begin{cases} 8 & \text{if the node is in stand-by} \\ 120 & \text{if the node has less than 100 queued tasks} \\ 140 & \text{if the node has less than 150 queued tasks} \\ 160 & \text{Otherwise} \end{cases}$$

Due to the initial conditions, if a load balancing policy is not adopted, the system is instable, as some nodes become immediately overloaded. Moreover, in the interval [200 : 300] cycles, the experiments simulate a traffic peak with an increment of 50% of the rate of incoming tasks.

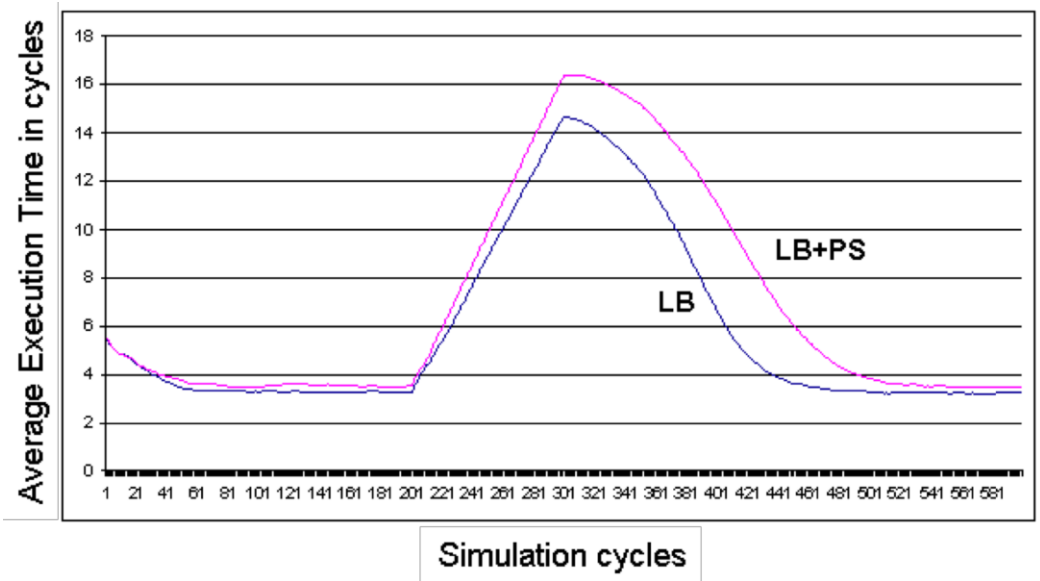

Figure 16: Comparison of task execution time

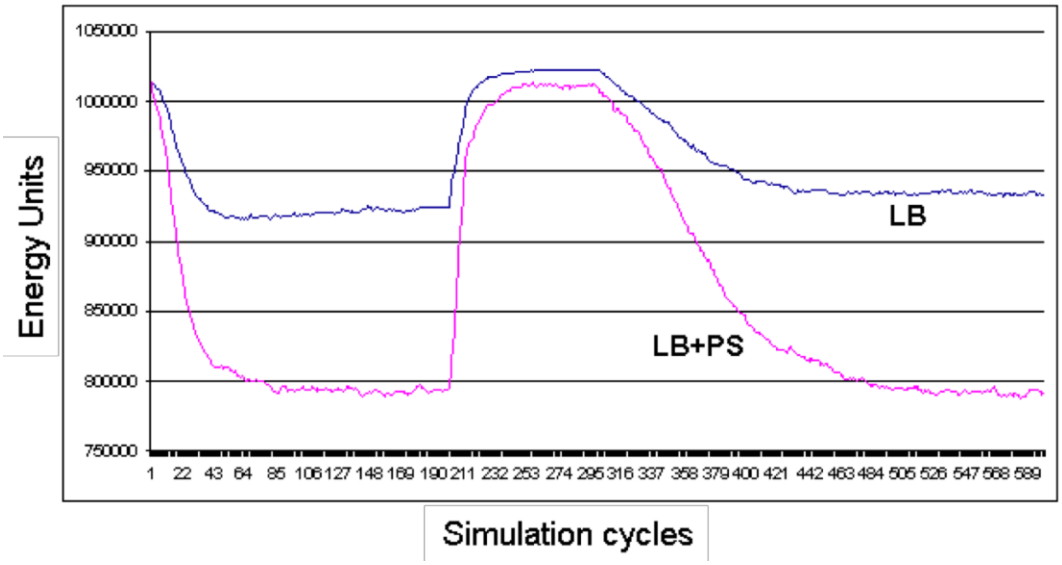

Figure 17: Comparison of energy consumption

Figure 16 and Figure 17 compare the average execution time and energy use for LB and LB+PS, respectively. As shown, the adoption of PS policies seems to introduce a benefit in the system that resulted in energy savings of about 14% (about 785000 energy units in stable state), which is achieved with a limited impact on execution time (about 5% in stable state). During the recovery phase from traffic peaks there is a maximum increment of 45% in the average execution time. This is due to a the higher delay of the system in returning to a stable state, which is based on the PS logic, which puts nodes into stand-by based on a single policy evaluation (see stand-by logic) disregarding the overall system state. Based on this some nodes may be forced into stand-by even if the overall system is still considered as overloaded. Although these nodes will be woken-up again, the system will perform inefficiently for a short time span.

LB+PS computes a quasi-optimal solution and the results were compared with the ones of a simulation of a system with an optimal distribution of load (e.g., a task is immediately assigned to an idle node). By considering the statistic variation of traffic, 5200 nodes are required to have a stable system with an average execution time of tasks of 1.95 cycles (instead of 2.15 cycles of LB+PS), and an average use of about 779000 energy units (instead of 785000). On the other hand, it is worth to point out that LB+PS requires about 25 cycles to reach a stable state in normal traffic and 160 cycles to recover from traffic peaks.

The described scenario shows that fully distributed algorithms with simple supervision logic are able to distribute load in a suitable and efficient way under normal load conditions as well as after load peaks. The efficient load balancing is also due to the fact that the proposed supervision algorithms do not move queued tasks, but contracts, i.e., the sources of requests. In this way, the algorithms balance the forthcoming load and limit the amount of information that is exchanged between ACEs. An extensive analysis of the LB and PS algorithms, and alternative options, is given in [19].

VIII. Combining ACE Embedded and Supervision Pervasion

ACE embedded supervision as well as supervision pervasions are considered to be complementary in both, the level of supervision tasks as well as their granularity. While the ACE embedded approach is more suitable for fine grained validation of system generic properties (e.g., self-repair, load distribution, and energy consumption / optimization); supervision pervasions are more suitable for service specific tasks and – due to the overhead resulting from the employment of a probably large number of supervision ACEs – applicable for tasks that can only be handled at a higher level of abstraction. For instance, the enforcement and validation of generic system management policies might turn out to be difficult using an embedded approach because it is not clear how to map those policies automatically into a local rule set or, in the context of ACEs, into the self-model. Vice versa, embedded supervision is applicable for the supervision of ACE internal properties. Nevertheless, a combination of both approaches is desirable for a number of application scenarios such as described next.

A. Scenarios

- Root cause analysis of faults in distributed systems is difficult because a fault might manifest itself at a completely different location of the system. While basic repair activities can be suitably handled by an embedded approach in many cases, root cause analysis requires a global view to a system, which contradicts the idea of embedded supervision to perform supervision activities on the basis of local information.
- Some problems require the coordinated effort of a number of distributed components to be solved. Consider a software update in a distributed computing network. One might want to apply a schedule that maintains a basic functionality while shutting down

some parts of the overall system to perform the update. While the execution of those coordinated activities can be performed based on local interactions and thus can be performed by a embedded supervision logic, the construction of an appropriate schedule and the distribution of sub-tasks requires again a global view. Similar, whilst updating such a system it may be important that the nodes and services from the "old" system are not mixed with the updated system. Again, this requires a more global viewpoint of the SUS.

- Supervision tasks based on statistical data concerning certain types of component requirements are derived from mass data (compare Section V.B on long-term supervision). For instance, load balancing requires the analysis of the current load situation in the neighborhood of a server. The same type of data can be used to do predictions of future load situations to e.g., identify bottlenecks. The unrestricted gossiping of such data to perform the necessary computations for such predictions is certainly not advisable; instead, the setup of dedicated event processing and correlation pipelines is required.

B. Supervisor Placement

In this section, we describe an approach to place supervisors "strategically" within a network of communicating nodes, and how to use it to distribute data amongst supervisors. Based on Figure 18, strategically means that:

- Each relevant cluster has to be connected to a particular supervisor. Recall that clusters are the result of the rewiring process and reflect, in a certain way, the functional structure of the SUS. Entities providing similar or connected sub-functions belong to the same clusters. In particular we are interested in those clusters that are dedicated to embedded supervision activities.
- As the system under supervision is organized according to various functional processes and considering that contracts between nodes define the interaction and relationships between interrelated processes, communication between supervisors that are related to overlapping node clusters is required for a large number of communication tasks. For instance, root cause analysis often requires the back tracing of a chain of faults until the initial problem is identified. For this, the communication between supervisors also has to back trace propagation of the fault. A similar observation applies to the software update scenario, where critical paths through the SUS need to be considered and where rollback procedures may need to be executed.

Consider an off-line computation approach where the dynamics of the underlying self-organized system, such as pre-computed distribution needs, need to be continuously updated on the basis of data obtained from the SUS. Changes in the contract structure within the SUS also require, the conceptually higher orientated, supervisor network to update itself. This shows that even off-line computation requires the existence of some kind of monitoring infrastructure that monitors and updates the current contract and clustering structure of the computing system. Self-organization mechanism could autonomously keep such a system up to date without the need for a centralized evaluation mechanism.

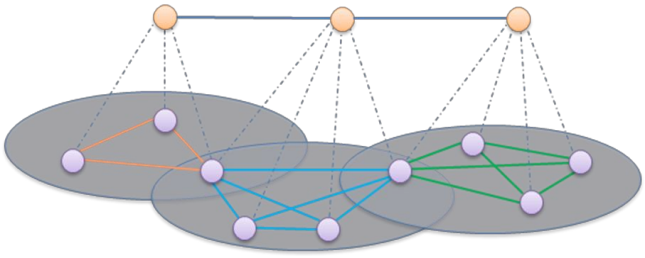

Figure 18: Strategic placement of supervisors (top nodes) in the SUS (bottom nodes)

For the initial construction of a supervisor network, we use the rewiring mechanism described in Section III.B. The network is constructed in three steps; steps 2) and 3) are continuously (concurrently) performed to update the network according to changes in the structure of the SUS. Each step is described as follows:

1. For the initial placement, a supervisor is determined for each cluster of the system. Using an ACE based system where nodes have unique identities; this can be achieved for each cluster by electing a leader (e.g., the ACE with the minimum address, or the highest computing power), and performing a GN/GA interaction to discover and to contract a supervisor ACE.
2. The supervisor discovers the other nodes within its associated cluster by using the rewiring mechanism. For that, recall that clusters are formed by the rewiring algorithm as well as by using a service dependent matching criteria. The matching criterion is that a supervisor s matches a node b if s is already connected to a node a matching b according to the matching criterion of the underlying cluster.
3. Finally, the connections between the supervisors that make up the supervisor network are achieved through the rewiring algorithm utilizing the following matching criterion: Supervisors s_1 and s_2 match if the sets of SUS nodes they are related to overlap.

Note that because steps 2) and 3) are performed continuously, each change in the contract structure of the SUS is detected. Thus, the supervisor network converges towards a state of strategic placement as described before.

C. Supervisor Communication

The gossiping algorithm described in Section III.A can be used to distribute global information about the state of the SUS. Thus, all supervisors have up to date information about the nodes in the clusters they belong too. Data related to parts of the system that are not directly connected to a supervisor are obtained through gossiping with adjacent supervisors. Naturally, the type of the data exchanged depends on the supervision task at hand. For instance, in the root cause analysis example data about "suspicious" observations can be exchanged, i.e., those information that indicate a possible propagation of the fault. In a second step,

more concrete analyses can be performed through the direct cooperation of supervisors on the potential propagation path.

Hence, the supervisors are able to maintain a global picture of the state and more importantly on the organization of the underlying SUS. A mathematical approach on this idea is given in [12] and [20].

IX. ADVANCES BEYOND THE STATE OF THE ART

ACE based systems provide services by means of interactions of a probably large distributed set of ACEs with a dynamically adapted interaction structure and task diversification [21][22]. Hence, the basic assumption underlying to traditional supervision approaches (see for instance [1][23][24][25][26]), which state that the SUS maintains a static architectural structure (i.e., it does not perform run-time architecture adaptations) is not longer valid for ACE based services.

The notion of a service providing system makes a novel approach for the formulation and deployment of autonomic control loops necessary, which does not require any a priori knowledge on the structure of the SUS. In order to address this need, the pervasive supervision approach includes a novel scheme to set-up those control loops that are based on the interaction of various ACEs that form a supervision ensemble. Evidently, the structure of the supervision pervasion adapts itself dynamically to the changes of the actual structure of the SUS.

Another novel achievement is the use of a common technological basis (namely the ACE software component) both for the SUS and the supervision system, which promotes self-similarity among components. This has a number of advantages as discussed next. The introduction of additional technologies does always increase the complexity of a system; hence a reduction of operational efforts by using a supervision system that is technologically different from the SUS is at least questionable. On the other hand, for the supervision system described in this paper, a number of basic functions that are necessary for supervision are already provided by the ACE component platform itself. Examples thereof include the service discovery and contracting mechanism based on the GN/GA protocol, which supports dynamic adaptation as described earlier; The separation between the process logic (provided by ACE self-models) and the function implementation (provided by ACE functional repositories); the built-in monitoring and control mechanisms the ACE supervision organ offers. Note that generic supervision tasks (such as liveliness validation as described in the case study in Section VI) can be applied to the components of a supervision pervasion as well. Thus, self-supervision can be performed through the proposed approach.

The temporal supervision of quantitative as well as symbolic based parameters and behavior is provided as a set of long-term supervision components. A more complex supervision ensemble can be enhanced through the flexible configuration / orchestration of these components with once that offer only basic supervision features. These components have been specialized to work with the ACE model and its declarative execution logic (i.e., based on self-models). For instance, state predictors have been specifically designed to address individual features of the ACE self-model / plan philosophy to model detailed ACE behavior over time and subsequently provide detailed predictions of potential future behavior.

The proposed approach for supervision of distributed autonomic systems introduces several novelties with respect to analogous solutions based on autonomic technologies. In fact, most solutions (e.g., [23]) rely only on the self-adaptation of the components, by elaborating changes in their internal state and in their execution environment / context. The proposed approach enhances the local self-adaptation features of autonomic components, with the possibility to exchange data in a peer-to-peer fashion. Supervision-related information is exchanged with neighbors through self-organized overlays. In this way the local supervision logic can work on a local vision of the whole system. In fact, through the overlay and the self-aggregation of information, the elements are able to collect and to diffuse data from / to their neighbors, to propagate them through gossiping protocols (e.g., the ones described in [14]), and combine them with locally available data.

Self-organization algorithms have already been adopted in defining supervision capabilities. For instance, [24] describes a load balancer based on these mechanisms. Nevertheless, the CASCADAS ACE Toolkit embedded supervision goes beyond this as it is fully integrated in the abstraction and communication model. Moreover, its implementation fully exploits the ACE model, its organs, self-models and interaction mechanisms. This would allow, for instance, that the load balancing is performed at the level of contracts, and not at the level of pending tasks.

Moreover, it is important to point out that, in contrast to centralized solutions that are designed to monitor a static cluster of computing resources [25], the proposed solutions are implemented in a pervasive and distributed way across the system to be supervised and that are able to supervise systems, which are dynamically changing in the number and in the configuration of their elements.

The proposed approach for embedding supervision logic in ACEs for managing distributed systems introduces several novelties with respect to analogous solutions based on autonomic technologies. In fact, most solutions, e.g., [27], rely only on the self-adaptation of components, which is achieved by elaborating changes in their internal state and in their execution environment / context. Instead, the proposed approach enhances the local self-adaptation features of ACEs, with the local exchange of information in a self-organized overlay through gossiping protocols such as described in [8][9][10]. In order to achieve decentralized supervision logic the algorithms can create, in a fully decentralized way, an approximated knowledge of (dynamically changing) global properties of the whole system, and use them in local supervision decisions.

X. APPLICATION SCENARIOS

In general, the supervision mechanisms proposed in this paper could be adopted to supervise any hardware and / or software system that are composed of a set of distributed and interacting components. As such, any resource of any given system can be associated to an ACE that is in charge of performing decentralized supervision logic or interacting with ACEs implementing specific supervision services.

This section elaborates briefly on some application scenarios, in the context of future telecommunications environments, where the supervision mechanisms proposed in this paper can be fruitfully exploited. For this, let's consider a simple architecture comprising the following three levels:

1. Level of resources where each of them is controlled by an ACE (or an aggregation of ACEs) through an interface, which allow its monitoring and affecting;
2. Level of ACEs supervising the resources according to embedded local supervision logic cooperating through gossiping protocols and overlays networks;
3. Level of self-organized ensembles of ACEs implementing self-adapting supervision services.

This simple architecture can be applied to tame the growing complication of supervision in future telecommunications networks that is characterized by the integration of several heterogeneous systems supporting the dynamic interconnection of huge amounts of small devices that simultaneously provide and consume services and data. The distributed supervision logic embedded in ACEs will allow the provision of supervision at infrastructure level, by coping with pervasiveness and the dynamic evolution of these environments. At the same time, the supervision pervasion would be able to self-configure and self-adapt its supervision capabilities in dependence to individual QoS and SLA specifications of specific end-to-end services.

For instance, as shown earlier, distributed supervision could improve the overall performance of pervasive clouds of computing resources by, e.g., shortening the response time through more effective load balancing policies or reducing the energy consumption by reallocating resources or putting the underused resources in stand-by. Furthermore, supervision pervasions can be adopted to cope with the management (e.g., QoS monitoring or SLA enforcement) of reliable content access and distribution services on such cloud of resources.

In fact, cloud computing is an area where the application of the proposed supervision principles can provide important benefits especially when considering that cloud computing involves a lot of disruption. For instance, the features that are essential for computing are within the network (processing and storage), the communication bus is the network itself and the input / output devices are the end user terminals. This reflects a fully distributed, highly heterogeneous system that needs to be supervised at various levels of granularity. Autonomic supervision can be used to support load balancing, dynamic configuration, fault tolerance, to enhance security, and to improve QoS in the presence of very dynamic conditions, which include resource availability and service requests. The basic idea consists in adopting ACEs to manage the high dynamicity of the cloud nodes in which users' may enter and exit the cloud in an unpredictable way increasing the dynamicity of the SUS even further. For example, each computing resource could be equipped with ACEs capable of exchanging and managing events coming from ACEs deployed on other resources and with ACEs implementing supervision services. In turn, supervision services could be used to cope with the problems of data synchronization whilst providing the proper number of duplications for the requested persistency.

Another application scenario is the supervision of distributed service provisioning platforms, where different actors can develop, provide, connect and interact, in a secure and reliable way, for selling, buying, negotiating, exchanging and trading any content, information, services and service components [28]. In such a context where components are dynamically negotiated and aggregated, supervision pervasions could be used by an actor creating a service by aggregating a cluster of components to enforce service-specific management policies, or by a provider of service components to supervise the instances of a service.

XI. CONCLUSIONS AND FUTURE WORK

One of the most serious technological challenges of future Telecommunication, ICT and Internet endeavors will be the interconnection and management of heterogeneous systems and the huge amounts of devices that are tied together in networks of networks. Autonomic Computing has already argued that, due to the increasing complexity of large-scale computing systems, both computers and applications need to learn how to manage themselves in accordance to high-level policies as specified by human operators. Nevertheless current autonomic solutions don't exploit the real pervasive nature of distributed systems.

This paper presented a novel approach for the supervision of highly dynamic and fully distributed systems structured as ensembles of autonomic components, based on two complementary and co-operating mechanisms: supervision pervasion, and embedded supervision.

The supervision pervasion is structured as an ensemble of distributed components that implement an autonomic control loop, which does not require any a-priori knowledge on the structure of the supervised system. The architecture devised is highly modular and can be configured towards individual needs. In addition, the supervision system is able to re-configure itself according to the changes of the SUS. This mechanism is mainly oriented to the supervision of clusters of ACEs implementing specific services in accordance to service-specific management policies. It was experimentally validated by the development of a prototype, which has been made available as open source. The performance overheads can be mostly neglected considering that advantages provided. This is based on the fact that the interaction between the SUS and the supervision system is asynchronous (i.e., the supervisor does not slow down the SUS), and performance bottlenecks resulting from the introduction of a supervisor are expected to be moderate.

Full quantitative evaluation of the approach is on all aspects is, however, subject of our ongoing work. A possible evolution of the prototype would be to include the definition of the management policies through a specific language. Also, the long-term supervision components could be enhanced to facilitate the dynamic orchestration into more advanced hierarchical supervision pervasions.

The embedded supervision consists of a set of supervision logic embedded in ACEs themselves. This mechanism enhances the local self-adaptation features of ACEs, with the local exchange of information in a self-organized overlay through gossiping protocols in order to implement decentralized supervision algorithms. This mechanism can supervise potentially huge amounts of components that are pervasively distributed and interconnected by offering the capability to manage ACEs at the level of their basic functions and at the level of their aggregations. Examples are fully distributed algorithms for handling binding failures, load balancing and for optimizing the utilization of resources. Some algorithms that have been implemented according to the embedded supervision approach were evaluated by means of simulations, which showed that a quasi-optimal behavior at system level could emerge from decisions that have been made by the cooperating local supervision logic. A full integration of this mechanism in the CASCADAS ACE Toolkit [5] is planned as future work.

The two mechanisms presented can fruitfully cooperate in order to provide a "cross-layer" supervision mechanism for distributed autonomic systems. In fact, a supervision pervasion must also be able to react to events generated by local supervision logic when it is not able to properly solve a given situation. These cooperation aspects will be considered in further investigations and in future experimental evaluations.

Security has not been taken into account yet. Obviously, any automated agent that is able to monitor and to impact the execution of some system comprises a considerable security threat. A solution would be to employ standard security mechanisms to authenticate and authorize a supervisor against the supervised system. Communication between supervisor and system under supervision can be encrypted using standard cryptographic approaches. More advanced approaches could use distributed schemes to establish trust relationships as outlined, for instance, in [31][32].

ACKNOWLEDGMENT

Authors would like to acknowledge European Commission for funding the IP CASCADAS (IST-027807) (FET Proactive Initiative, IST-2004-2.3.4 Situated and Autonomic Communications).

REFERENCES

[1] Deussen P., Baumgarten M., Mulvenna M., Manzalini A., and Moiso C., "Autonomic Re-configuration of Pervasive Supervision Services - The CASCADAS Approach," Proc. 1st International Conf. on Emerging Network Intelligence, 2009, pp. 33–38.

[2] Kephard J. O. and Chess D. M., "The Vision of Autonomic Computing," IEEE Computer, Vol. 36, No. 1, 2003, pp. 41–50.

[3] Horn, P. "Autonomic Computing Manifesto", http://www.research.ibm.com/autonomic/manifesto/, 2001.

[4] Manzalini A., Zambonelli F., Baresi L., and Di Ferdinando A., "The CASCADAS Framework for Autonomic Communications," in "Autonomic Communication", A. Vasilakos, M. Parashar, S. Karnouskos, W. Pedrycz (Eds.), Springer, 2009, pp. 147–168.

[5] CASCADAS Project, "ACE Toolkit open source," available at http://sourceforge.net/projects/acetoolkit/.

[6] Extended Finite State Machine, http://en.wikipedia.org/wiki/Extended_finite_state_machine

[7] Marrow P., Bonsma E., Wang F., and Hoile C., "DIET — A Scalable, Robust and Adaptable Multi-Agent Platform for Information Management," BT Technology Journal, Vol. 21, No. 4, 2003, pp. 130–137.

[8] Jelasity M., Montresor A., and Babaoglu O., "Gossip-based Aggregation in Large Dynamic Networks," ACM Transactions on Computer Systems, Vol. 23, No. 3, 2005, 219–252.

[9] Babaoglu O., Canright G., Deutsch A., Caro G. A. D., Ducatelle F., Gambardella L. M., Ganguly N., Jelasity M., Montemanni R., Montresor A., and Urnes T., "Design patterns from biology for distributed computing," ACM Transactions on Autonomous and Adaptive Systems, Vol. 1, No. 1, 2006, pp.:26–66.

[10] Babaoglu O. and Jelasity M., "Self-* Properties through Gossiping," Philosophical Transactions of the Royal Society, Vol. 366, No. 1881, October 2008.

[11] Saffre F., Tateson R., Halloy J., Shackleton M., and Deneubourg J. L., "Aggregation Dynamics in Overlay Networks and Their Implications for Self-Organized Distributed Applications," The Computer Journal, 2008.

[12] Deussen P. H., "Model Based Reactive Planning and Prediction for Autonomic Systems," Proc. Workshop on INnovative SERvice Technologies (INSERTech07), 2007, pp. 1–10.

[13] Weiser M., "The Computer for the 21st Century," Scientific American, Vol. 265, No. 3, 1991.

[14] Han J. and Pei J., "Mining Frequent Patterns by Pattern-Growth: Methodology and Implications," ACM SIGKDD Explorations Newsletter, Vol. 2, No. 2, 2000, pp. 14–20.

[15] Manzalini A., Minerva, R., and Moiso C., " Autonomic Clouds of Components for self-managed Service Ecosystems," in Journal of Telecommunications Management, Vol. 3, No. 2, 2010, to appear.

[16] Deussen P. H., Ferrari L., Manzalini A., and Moiso C., "Highly Distributed Supervision for Autonomic Networks and Services," Proc. 5th Advanced International Conference on Telecommunications (AICT2009), 2009.

[17] Chen B., Jamieson K., Balakrishnan H., and Morris R., "Span: An Energy Efficient Coordination Algorithm for Topology Maintenance in Ad Hoc Wireless Networks," Journal Wireless Networks, Vol. 8, No. 5, 2002, pp. 481–494.

[18] Klein J., "breve: a 3d environment for the simulation of decentralized systems and artificial life," Proc. 8th International Conference on the Simulation and Synthesis of Living Systems (Artificial Life VIII), The MIT Press, 2002.

[19] Manzalini A., Minerva R., and Moiso C., "Supervision of decentralized systems: the magic of self-organization," Telecom Italia Internal Report TFC0900009, 2009.

[20] Deussen P. H., "Supervision of Autonomic Systems – Tutorial," Proc. Budapest Tutorial and Workshop on Autonomic Communications and Component-ware, 2008, Published on CD.

[21] Devescovi D., Di Nitto E., Dubois D., and Mirandola R., "Self-organization algorithms for autonomic systems in the SelfLet approach," Proc. 1st Conference on Autonomic Computing and Communication Systems, 2007, pp. 1–10.

[22] Shackleton M., Saffre F., Tateson R., Bonsma E., and Roadknight C., "Autonomic Computing for Pervasive ICT – A Whole-System Perspective," BT Technology Journal, Vol. 22, No. 3, 2004, pp.191–199.

[23] Deussen P. H., Valetto G., Din G., Kivimaki T., Heikkinen S., and Rocha A., "Continuous On-Line Validation for Optimized Service Management," Proc. EURESCOM Summit 2002, 2002.

[24] Garlan D., Cheng S., Huang A., Schmerl B., and Steenkiste P., "Rainbow: Architecture-based Self-adaptation with Reusable Infrastructure", IEEE Computer, Vol. 37, No. 10, 2004, pp. 46–54.

[25] Kaiser G., Parekh J., Gross P., and Valetto G., "Retrofitting Autonomic Capabilities onto Legacy Systems," Journal of Cluster Computing, Vol. 9, No. 2, 2006, pp. 141–159.

[26] Knight J. C., Sullivan K. J., Elder M. C., and Wang C., "Survivability Architectures: Issues and Approaches," Proc. DARPA Information Survivability Conference and Exposition, 2000, pp. 157–171.

[27] McCann J. and Huebscher M., "Evaluation Issues in Autonomic Computing," Proc. Grid and Cooperative Computing Workshops (GCC), 2004, pp. 597–608.

[28] Deussen P. H., Höfig E., and Manzalini A., "An Ecological Perspective on Future Service Environments," Proc. 2nd IEEE International Conference on Self-Adaptive and Self-Organizing Systems Workshops, 2008, pp. 37–42.

[29] Di Nitto E., Dubois D. J., Mirandola R., Saffre F., and Tateson, R. "Applying Self-Aggregation to Load Balancing: Experimental Results," Proc. Bionetics2008, 2008, pp. 1–8.

[30] Pinheiro E., Bianchini R., Carrera E., and Heath T., "Load Balancing and Unbalancing for Power and Performance in Cluster-Based Systems," Proc. Workshop on Compilers and Operating Systems for Low Power, 2001.

[31] Zubair, I., and M. H. Islam, "Adaptive Trust Management in P2P Networks using Gossip Protocol", Proc. 4th Int. Conf. on Emerging Technologies, Rawalpindi, Pakistan, 2008, pp. 176 – 181.

[32] Cascella, R., "Enabling Fast Bootstrap of Reputation in P2P Mobile Networks", Proc. IEEE 23rd International Conference on Advanced Information Networking and Applications (AINA-09), Bradford, UK, 2009, pp. 371 – 378.