An Architecture-Based Approach for Compute-Intensive Pervasive Systems in Dynamic Environments

Rima Al Ali, Ilias Gerostathopoulos, Inti Gonzalez-Herrera, Adrian Juan-Verdejo, Michal Kit, Bholanathsingh Surajbali

An Architecture-Based Approach for Compute-Intensive Pervasive Systems in Dynamic Environments

Rima Al Ali[1], Ilias Gerostathopoulos[1], Inti Gonzalez-Herrera[2], Adrian Juan-Verdejo[3,4], Michal Kit[1], Bholanathsingh Surajbali[3]

[1]Charles University in Prague, Faculty of Mathematics and Physics, Czech Republic
[2]IRISA, University of Rennes 1, Rennes, France
[3]CAS Software A.G., Karlsruhe, Germany
[4]Chair of Information Systems I, University of Stuttgart, Stuttgart, Germany

alali@d3s.mff.cuni.cz, iliasg@d3s.mff.cuni.cz, inti.glez@irisa.fr, adrian.juan@cas.de, kit@d3s.mff.cuni.cz, b.surajbali@cas.de

ABSTRACT

Distributed systems have continued to evolve and we note two important trends: the dramatically increasing level of dynamism in contemporary distributed systems and the convergence of mobile computing with cloud computing. The end result is that it is very difficult to achieve the required level of scalability and dependability in a systematic way when considering pervasive systems that are software- and compute-intensive and whose functionality is typically augmented by static cloud infrastructure resources. This work discusses relevant challenges and requirements for integrating cloud computing with pervasive systems operating in dynamic environments. We present a set of requirements using a holistic case study and describe a reference architecture to address these requirements.

1. INTRODUCTION

Rapid advancements in mobile devices, wireless sensors and mobile communications have led to the development of cyber-physical systems, pervasive computing and smart environments with important applications in environmental, civilian, military, industry, and government sectors. However, the ability of mobile devices and pervasive systems to perform intense computation is limited due to small battery life and computing power. To alleviate this limitation many research works such as Fog computing [8], cloudlets [17], or client-edge-server ecosystem [3] have emerged to bring the computing power of cloud data centers closer to the end devices. The objective is to offload part of the computation, needed to deliver the right quality of experience to the end user, to scalable server infrastructure while ensuring low latency and small delays.

However, these research works do not address problems of dynamic and open-ended environments, such as unreliable communication, intermittent connections and physical mobility, which are typical in modern cyber-physical systems. Such types of systems require a significant level of self-awareness and autonomy so that they can dynamically adapt to changes in their operational environment and recover from potential failures. In this context research initiatives, such as the EU FP-7 projects ASCENS[1] and QUANTICOL[2], have been leading efforts of featuring the appropriate computation models and development processes to develop large-scale distributed autonomic and adaptive systems. Specifically, ASCENS approach is based on the novel concept of grouped, attributed-based communication between service-components [7].

What is missing is a common reference architecture that will combine the advantages and promises of high-bandwidth, low-latency cloud-based systems with that of distributed adaptive systems. Using such a reference architecture would allow for building Compute-Intensive Pervasive Systems (CIPS). This paper proposes the following contributions towards reaching this goal:

- *A CIPS cloud reference architecture*. In CIPS resource-consuming jobs of pervasive devices operating in dynamic environments are delegated to external infrastructure with sufficient computing capabilities, while the system is still able to provide basic services in a best-effort manner when no infrastructure is available or fail gracefully in a controlled manner.
- *A CIPS model-based validation*. We validate our approach using a model-based approach through a holistic case study from the domain of emergency coordination using Kevoree [5] and DEECo [4] component systems.

The paper is structured as follows. First, in Section 2, we describe the main challenges required by CIPS using a fire-fighters use-case scenario. Then, in Section 3 we describe the proposed reference architecture approach, followed by Section 4, where we describe the instantiation of the reference architecture using DEECo and Kevoree. Next, in Section 5 we analyse the applicability of our approach at different scenarios and discuss the implementation aspects of our approach, followed by Section 6 providing an analysis against related work. Finally in Section 7 we present our conclusions and identify areas of future work.

2. CASE STUDY

Let us consider that an emergency such as a fire or a hurricane breaks out. A firefighting department deploys a team of firefighters and a team leader within the emergency field. The team leader organizes, commands, and sends feedback to teams. Nowadays, firefighting departments can provide their teams with sensing and actuating devices due to the lower cost of technology and the need to improve safety and decision making during an emergency. As a result, firefighting departments and team leaders are able to obtain a lot of data gathered in real-time about firefighters' position, health condition, or surrounding environment. As firefighters can now be better tracked, they can be more effectively commanded by their team leader.

In our case study, team members are provided with battery-powered mobile sensing and computing devices with wireless

[1] http://www.ascens-ist.eu

[2] http://blog.inf.ed.ac.uk/quanticol/

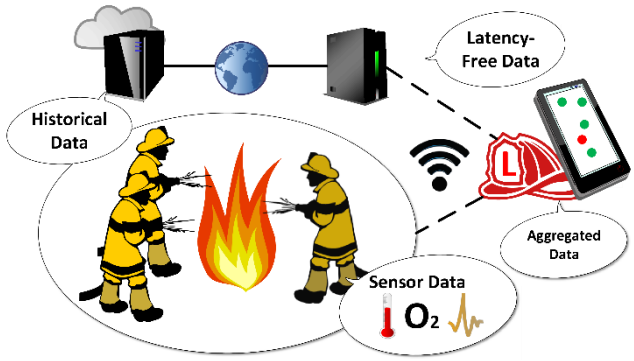

Figure 1. Firefighter coordination case study.

capabilities – see Figure 1. Those devices capture information about their environment – for example humidity, noise, location, air quality, or temperature – and communicate through a mobile ad-hoc network. The devices carried by firefighters' could lose at times wireless connectivity but are always able to independently sense, compute and actuate. However, mobile devices are not necessary able of doing compute-intensive tasks due to battery, or CPU and memory restrictions. Team leaders carry devices with better computation and battery capabilities that aggregate data so that useful global information can be extracted. Additionally, there are data centers in the vicinity of these sensing and/or actuating devices, which are not restricted in terms of battery or computing power. In such setting, mobile devices should be able to offload compute-intensive tasks with real-time requirements to data centers in the vicinity so that the overall system overcomes its inherent risks – increased latencies, slow response times, increased delays, disconnection from the network, or battery drainage – that might hinder the real-time and resource-demanding scenarios. Additionally, cloud environments accessed through the internet could be employed to execute non real-time tasks such as Big Data analysis, data aggregation that allows coordination with other firefighting departments, or training of prediction models.

We list here some application scenarios where the available technology can greatly benefit the firefighters in their job.

1. *Image and video recognition.* Thermal images captured by the firefighter's device can be processed to identify potential risks the firefighter is unaware of. Mobile devices carried by the firefighters do not have the necessary resources to carry out this compute-intensive processing [12].
2. *Augmented reality.* In an augmented reality scenario, the system sends live video stream to firefighters to notify them of what they cannot see due to smoke, obstacles, or their high levels of stress. This scenario demands low communication latencies and fast response times.
3. *Real-time decision making.* Data are sensed via spatially distributed sensors, pre-processed and aggregated in order to extract meaningful information, which is then available to the team leader. This in turn enhances the leader's decision making process due to his or her "virtual sensing" capability.
4. *Timely actuation based on sensed information.* Sensed data based on humidity, temperature, or oxygen concentration are aggregated. If these data are timely processed, the output can trigger a response sent to devices carried by firefighters to let them know of a dangerous situation a fellow firefighter might be facing – due to, e.g., high temperatures in his or her surroundings.

3. APPROACH

In this section the main components of our architectural solution are presented. The solution is inspired by, but not restricted to, the specific challenges outlined in the case study. Indeed, it is envisioned that the proposed architecture can act as a template, in the form of a reference software architecture [1] that will guide the structuring of applications deployed on CIPS.

3.1 Nodes and their dependencies

A target application instance comprises a number of entities with distinct roles and dependencies. At the early requirements stage, their dependencies can be depicted using the Strategic Dependency diagram [14] as the one depicted in Figure 2.

We differentiate between four types of nodes:

i. *Low-power nodes.* Typically mobile sensor nodes scattered around in the physical environment. They are responsible for acquiring the sensor data and disseminating them to the other nodes in their vicinity, which are either low-power or resource-poor nodes (described next). For that, they need to possess hardware sensing capabilities. These nodes can also act as actuators, in which case they get to have a direct impact on their physical environment or on the end-user experience.

In the case study, low-power nodes are small wearable devices that can measure a firefighter's acceleration, position, oxygen level and external temperature. In a more sophisticated scenario, the low-power nodes include image/video recording devices and augmented reality glasses, which both sense, by means of capturing the firefighters' vision as a constant video stream, and actuate, by providing visual aid to the firefighters (e.g., pointing to the exit of a building in case of limited visibility because of smoke).

ii. *Resource-poor nodes.* Mobile nodes with limited computation power and with energy constraints. They are responsible for aggregating the sensor data from the low-power nodes. The aggregated data allow for a number of resource- and compute-intensive tasks, such as online image/video analysis. To prevent possible short-term resource starvation, they delegate the task of online data analysis (together with other resource-demanding tasks) to the more powerful, resource-rich nodes.

In the case study, resource-poor nodes are ragged hand-held devices such as PDAs and tablet PCs. They are used by the leaders of the firefighter teams, who require gathering and acting upon data sensed by the firefighters on the field, while being themselves nearby and on the move. Applications running on these nodes are able to perform simple forms of data pre-processing and reasoning (e.g., when a firefighter's breathing apparatus stops working while the firefighter is on the field, then he or she is probably in danger).

iii. *Resource-rich nodes.* Nodes with ample computational power and memory. They are not energy-constrained. They are responsible for performing some kind of online data analysis, such as image/video analysis (performed e.g., via computer vision algorithms), online trend prediction and immersive graphical visualizations [2]. They only temporarily store the data required for these tasks; for long-term data storage and offline analysis they rely on traditional cloud infrastructure nodes.

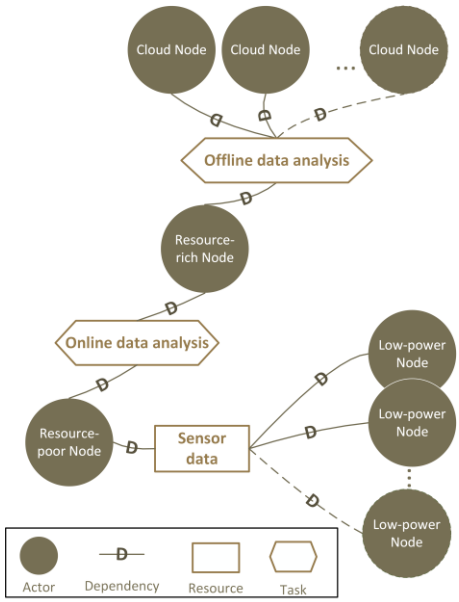

Figure 2. Nodes and their dependencies in an application deployed on CIPS.

In the case study, resource-rich nodes are deployed in micro data centers on powerful servers residing in firefighting departments and other organizations that traditionally belong to the public sector (e.g., schools, hospitals, police departments) or to private companies/individuals that voluntarily offer their spare computation power.

iv. Cloud nodes. Dynamically-provisioned nodes deployed on conventional data centers of one or more cloud infrastructure providers. They are responsible for storing historical data and performing offline analysis on them (e.g., Big Data analytics, postmortem documentation). They have the ability to scale out on demand and efficiently process large chunks of data using massively-parallel data processing platforms, such as MapReduce.

In the case study, cloud nodes are pre-configured virtual machines provisioned by an IaaS provider (e.g., Rackspace, Amazon) producing firefighter mission reports and analytics and providing long-term reliable storage of the analyzed data.

3.2 Reference architecture

The reference architecture shows the target system as a composition of software components (Figure 3). The components reify the identified nodes and their links reify the nodes' dependencies. A key observation is that different component interactions have different requirements on the underlying end-to-end communication links. Notably, the links can be grouped into three types as shown in Figure 3 which highlights the different link types.

(a) low-power ⇔ resource-poor nodes. The main requirement of this type of interaction is that it has to be robust in face of disconnections, meaning that intermittent connections and temporary disconnections should be tolerated. Taking into consideration the physical distribution of sensors in our system, and the fact that they are mobile, the low-power nodes may dynamically join and leave ad-hoc networks created around the resource-poor nodes. To deal with such dynamic behavior, this part of the reference architecture can

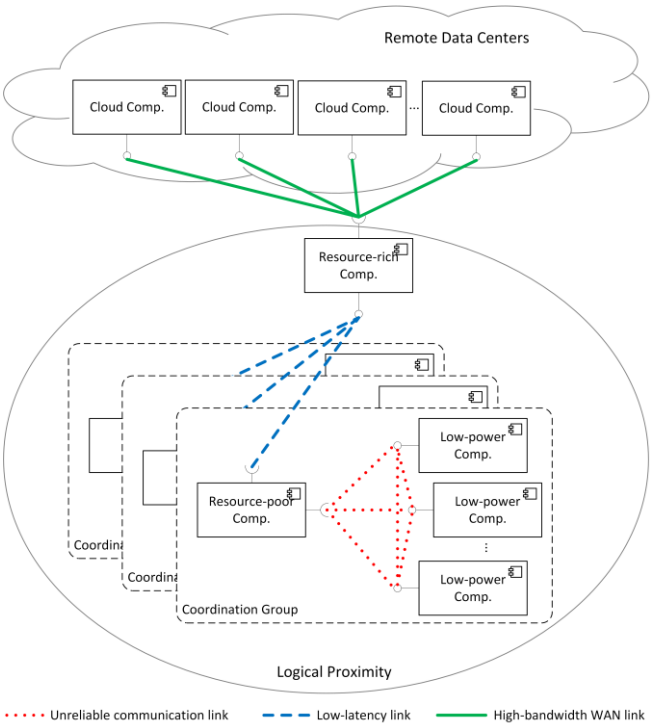

Figure 3. Proposed components and their links in the reference architecture.

be instantiated using a component framework tailored for dynamic cyber-physical systems. We provide such an example in Section 4.2.

(b) resource-poor ⇔ resource-rich nodes. Low communication latency is of paramount importance in this type of interaction. Considering that most of the resource-demanding applications that run on the resource-rich nodes also have soft real-time constraints (e.g., video streaming), network delays can impair user experience and can have safety implications if the network latency is not kept minimal. As a promising technique to implement this link type, we advocate a cloudlet-based approach as described in the next Section (Section 3.3).

(c) resource-rich ⇔ cloud nodes. This type of interaction does not impose any real-time constraints. High bandwidth is more important than low latency here, since typically large amounts of data have to be transmitted upstream. Since the cloud nodes reside in remote data centers accessible through the internet backbone, this type of link type should be realized via traditional IP protocols.

3.3 Cloudlet approach to reducing communication latency

In order to achieve low end-to-end latency between the resource-poor and resource-rich nodes (dashed link in Figure 3) it is necessary to bring the deployed components "close" enough. Research on offloading mobile multimedia applications to remote data centers has quantified this constraint to the maximal acceptable logical proximity of 1 WiFi [6]. A promising direction in satisfying such requirement on low latency is to employ the hierarchical architecture of *cloudlets* [17, 18].

A cloudlet is essentially a data center in-a-box, which resides close to the edge of the network with the goal of reducing

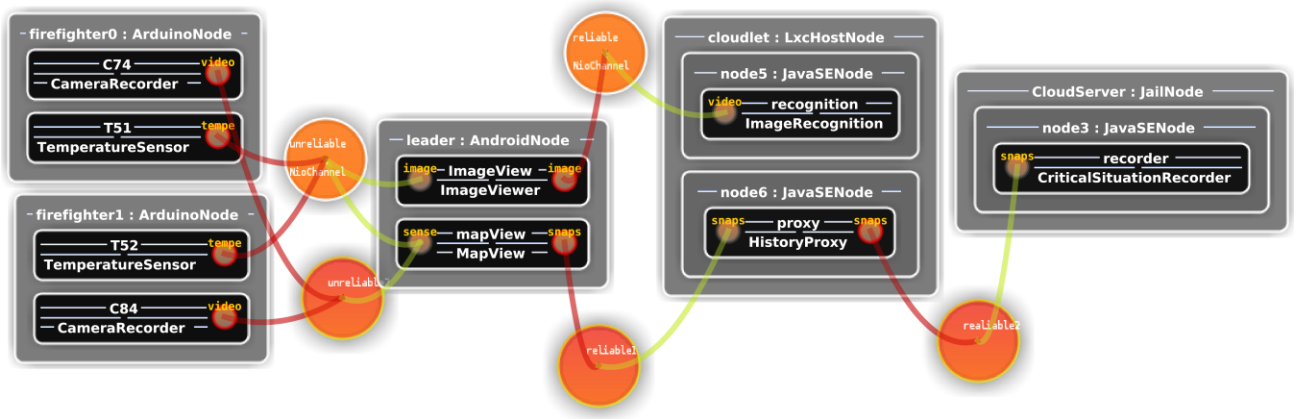

Figure 4. Modeling of the running example in Kevoree.

communication latency. A cloudlet resembles a cluster of multicore computers with gigabit internal connectivity. The main difference from traditional data centers is that a cloudlet has only soft state: virtual machine images and files are cached on local storage from one or more remote data centers. This means that management burden is kept considerably low. The idea is that, once configured, a cloudlet can dynamically self-provision from remote data centers. An approach to cloudlet customization, when following a VM-based approach, is termed *dynamic VM synthesis* [17]: a mobile client delivers a VM overlay (delta between the application-specific and base VM) to the cloudlet infrastructure where the base VM is already installed. This approach not only speeds up customization compared to the alternative of VM migration from remote data centers to cloudlets, but also works when no remote cloud is accessible.

Cloudlets stand as a possible realization of the resource-rich nodes of the reference architecture. In the case study, the utilization of cloudlets is envisioned in the following way. Cloudlet containers reside in micro data centers of organizations (e.g., firefighting and police departments), which are typically scattered around in a city. Once a fire breaks out in a district covered by a specific firefighting department, several teams are dispatched from the nearby departments. The firefighter leaders bring along on their mobile devices the pre-computed VM overlays, which correspond to data analysis tasks relevant to the mission in hand. The overlays are then delivered to the nearest cloudlet-based micro data center, which possesses a number of standard base VMs, and the pre-configured complete VMs are reassembled. This way, the resource-rich components deployed in the cloudlet VMs are able to respond to requests coming from the resource-poor nodes in a timely manner.

4. INSTANTIATIONS OF REFERENCE ARCHITECTURE

In this section we present two possible instantiations of our reference architecture. Both examples are grounded in standard component-based software engineering techniques and well-defined component models, i.e., collection of rules for component definition, composition and interaction.

To exemplify them, we use as running example a specific scenario from our case study. In the scenario, firefighter (low-power) nodes with thermal image capturing capabilities cooperate with team leader (resource-poor) nodes in order to augment their real-time risk detection capability. Team leaders aggregate image data from their team members but do not directly analyze them; for that they rely on the cloudlets (resource-rich) nodes.

4.1 Kevoree model

Kevoree is an example of a dynamic component platform [5] which can be used to realize all the nodes and links of our reference architecture. Kevoree relies on Models@runtime approach [15] to provide model-based dynamic adaptation in form of re-configuration of the running system. It does so by extending the idea of reflection to consider (architectural, but also behavioral) models which can be extracted from the running system (for reasoning, transformation, validation, and simulation purposes) and then automatically resynchronized.

Kevoree provides a number of concepts to structure distributed systems. The Node concept models an infrastructure node, the Component node models a reusable subsystem with well-defined functionality, and the Channel concept models different communication semantics between remote Components. As an example, the Kevoree model of our running example depicted in Figure 4, comprises five instances of Nodes and five instances of Channels. This architecture models the simple case where two firefighter nodes (team members) cooperate with one team leader, whose capability is augmented by one cloudlet and one cloud node. Figure 5 depicts the same configuration in the textual DSL of the Kevoree. Note that bindings between individual components in Kevoree are explicitly modeled (e.g., lines 19-21), which essentially renders it possible to visualize the system in the graphical DSL used in Figure 4.

4.2 DEECo model

When structuring the part of the system which is heavily influenced by the dynamicity in the physical world, i.e., low-power to resource-poor links (dotted links in Figure 3, left part of the architecture in Figure 4), it is beneficial to employ component models that specifically target environments with evolving physical substratum, such as DEECo (*Dependable Emergent Ensembles of Components*) component model [4, 10].

In DEECo, a *component* is an autonomous unit of computation and deployment, which comprises knowledge and processes. An example of a DEECo component modeling the low-power node of our running example is depicted in form of DEECo DSL in Figure 6, lines 7-25. Processes in DEECo operate upon the knowledge of a component and feature cyclic execution based on the concept of feedback loop, similar to processes in real-time systems. DEECo components are not bound to other components neither do they

```
1.  …
2.  # components
3.  addComponent T52@firefighter1 : TemperatureSensor
4.  addComponent C84@firefighter1 : CameraRecorder
5.  addComponent C74@firefighter0 : CameraRecorder
6.  addComponent T51@firefighter0 : TemperatureSensor
7.  addComponent ImageView@leader : ImageViewer
8.  addComponent mapView@leader : MapView
9.  addComponent recognition@node5 : ImageRecognition
10. addComponent recorder@node3 : CriticalSituationRecorder
11. …
12. #channels
13. addChannel unreliableChannel0 : NioChannel
14. bind T52.temperature@firefighter1 => unreliableChannel0
15. bind T51.temperature@firefighter0 => unreliableChannel0
16. bind mapView.sensedData@leader => unreliableChannel0
17. …
18. addChannel unreliableChannel1 : NioChannel
19. bind C74.videoStream@firefighter0 => unreliableChannel1
20. bind C84.videoStream@firefighter1 => unreliableChannel1
21. bind ImageView.image@leader => unreliableChannel1
```

Figure 5. Kevoree DSL example.

explicitly communicate with each other. Instead, interaction among components is determined by their composition according to their *roles* (e.g., lines 1-2, 4-5) into groups – called *ensembles* – cooperating to achieve a particular goal. Ensembles are dynamically formed based on the state of components and external situation (e.g., when a group of firefighters are physically close together, they form an ensemble). Within an ensemble, communication takes the form of knowledge exchange between the selected components, performed by the runtime framework of DEECo. As an example, consider the ImageUpdate ensemble definition (Figure 6, lines 49-57). It captures the fact that, when two components belong to the same team and the firefighter component has a new image to propagate (*membership* condition, lines 52-54) the image knowledge gets propagated to the knowledge of the team leader (*knowledge exchange* function, lines 55-56).

Obviously, DEECo is a viable choice in highly dynamic and unpredictable settings where direct bindings à la Kevoree do not scale, as it provides a notion of built-in robustness through periodic execution. It also copes well with the dynamicity of opportunistic networks [9], which typically underlies type (a) link type (red links) of our reference architecture by relying on stateless interaction templates for component linking. However, DEECo is not the best choice for realizing the non-dynamic parts of the reference architecture (in particular, data replication used to achieve robustness in DEECo can harm performance); for them, more "traditional" component models, such as Kevoree, are the preferred choice.

5. DISCUSSION

In this section, we first discuss the implications of our approach by performing a what-if analysis of the situations that can arise in the target system (Section 5.1). Then, in Section 5.2 we elaborate on the implementation aspects of our approach by focusing on both parts of the target system: the mobile ad hoc system part (MANET) and the Ethernet backbone part.

5.1 What-if analysis

With the aim of showing how our approach optimizes resources and data provisioning and availability, we explain five levels of system availability. Each level reflects on the potential connectivity issues of the use case presented in Section 2.

```
1.  role ImageProducer:
2.    missionID, image
3.
4.  role TemperatureAggregator:
5.    missionID, images
6.
7.  component Firefighter7 features ImageProducer
8.    knowledge:
9.      ID = 7
10.     missionID = 5622
11.     image = Image("#20-02-2014_18:52:11")
12.     oxygenLevel = 55%
13.     temperature = 33.1
14.     position = {36.90103, 27.284229}
15.     … /* other knowledge field definitions */
16.   process captureImage
17.     out image
18.     function:
19.       image ← Camera.capture()
20.     scheduling: triggered( CameraCaptureSignal )
21.   process measureOxygenLevel
22.     out oxygenLevel
23.     function:
24.       oxygenLevel ← OxygenLevelSensor.read()
25.     scheduling: periodic( 1000ms )
26.   … /* other process definitions */
27. … /* other firefighter definitions */
28.
29. component TeamLeader features ImageAggregator
30.   knowledge:
31.     ID = 2
32.     missionID =5622
33.     images = {{7, Image("#20-02-2014_18:52:11")},
34.               {8, Image("#20-02-2014_18:41:06")}…}
35.     results = …
36.     … /* other knowledge field definitions */
37.   process uploadImagesForProcessing:
38.     in images
39.     function:
40.       TCPconnectionToRemoteServer.upload(images)
41.     scheduling: periodic( 500ms )
42.   process receiveImageProcessingResults:
43.     out results
44.     function:
45.       TCPconnectionToRemoteServer.get(results)
46.     scheduling: periodic( 10000ms )
47.   … /* other process definitions */
48.
49. ensemble ImageUpdate:
50.   coordinator: ImageAggregator
51.   member: ImageProducer
52.   membership:
53.     member.missionID == coordinator.missionID &&
54.       coordinator.images.get(member.ID) != member.image
55.   knowledge exchange:
56.     coordinator.images ← { (m.ID, m.image) | m ∈ members }
57.   scheduling: periodic( 500ms )
```

Figure 6. Modeling the MANET part of the running example in DEECo.

Level-4 – all components can communicate with each other. All the optimization and resource harvesting features are enabled. As a result, type 2 and 3 devices (see Section 3) benefit from the offloading of computation-intensive tasks. The offloading enables us to support the decision making with resource-greedy techniques. Additionally, if cloud platforms provide big data capabilities the nodes profit from additional data processing possibilities. From the firefighters' perspective, they have access to all the information that could possibly support their actions in the field. These data include latency-free results of highly intensive computation performed by the cloudlet such as image processing or voice recognition. Due to the use of resource-unconstrained computational power — i.e., cloudlet and cloud

platforms — team leaders get instant access to analyses that could increase the correctness of a strategy selection when managing team members.

Level-3 – the cloudlet constitutes a communication edge and the system is not able to leverage the cloud back end. This could possibly degrade the offline analysis support to team leaders' decision-making. In this case, we assume that the information provided by the cloud infrastructure is not critical for the currently ongoing operation in the field but it is useful for analysis performed afterwards.

Level-2 – we assume here no cloudlet connectivity, which results in degradation of system performance and data availability. Team leaders and firefighters — who benefit directly from the cloudlet latency-free resources — need to adapt to the situation by reducing their resource utilization. In turn, less information is available which constraints the decision support.

Level-1 – team leaders are disconnected from the rest of the team. Hence, team members can no longer rely on their leader's support and they depend on each other to optimize the access to information. We assume that each firefighter is able to communicate with at least one other team member, so that minimal data exchange exists.

Level-0 – no communication is available. Team members execute in isolation and depend only on their own sensor data. Nevertheless, the system remains available and provides minimally required information — such as oxygen level, temperature, or exit direction — to a firefighter. Therefore, the information available satisfies the aforementioned requirement regarding system availability.

Our hybrid DEECo-Kevoree approach (described in Section 4.2) aims to address all of these levels. In particular, the system's operability is sustained by employing DEECo at Level-2, Level-1, and Level-0. DEECo components are autonomous and able to execute in isolation. Hence, knowledge about the remote components is not required all the times. Moreover, their processes implement adaptation techniques that provide reaction mechanisms depending on the level of data and resources availability. Those characteristics of the DEECo component model imply higher system resilience, which subsequently prolongs the system's operability and availability. These properties are of course critical for the end users in our case study.

Figure 7 summarizes this section by depicting our 5-level model of system availability.

5.2 Implementation Aspects

In this section we discuss the technical aspects of the target system, elaborating on the MANET and Ethernet subsystems separately.

5.2.1 MANET jDEECo

As a part of the discussed system realization, we envision to employ the jDEECo framework[3]. It is a Java realization of the DEECo component model that provides the necessary programming abstractions and runtime environment for building and running DEECo-based applications. The reference architecture can be mapped directly to components and ensembles in jDEECo, constituting the basic building blocks of the framework. Considering our case study scenario, we presume that

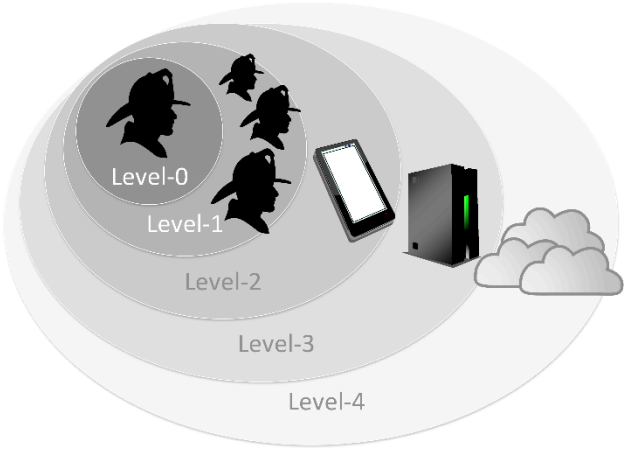

Figure 7. The five levels of system availability.

jDEECo will be deployed on the low-power and resource-poor nodes. In principle ensembles in DEECo are ubiquitous, and as such they are supposed to be evaluated globally, which requires access to all component data. In order to achieve this and, at the same time, sustain both distribution and scalability, ensembles are evaluated locally by each of the jDEECo (runtime and component host) node. For that each jDEECo node periodically distributes (broadcasts to a radio channel) its local component data, which is then received by firefighter nodes in the communication range. Apart from publishing its own local data, jDEECo nodes rebroadcast other nodes' data (called *replicas*), taking the role of relays. In order to minimize the network traffic overhead produced by periodic broadcasts, a concept of communication boundary is used, which considerably reduces the amount of data generated by jDEECo runtimes. An important characteristic in case of data dissemination in jDEECo is the absence of the network layer (from the OSI model), as no routing is required. This approach fits the semantics of ensemble, which, as previously mentioned, needs to be ubiquitous.

In terms of jDEECo targeted communication standards (i.e. IEEE 802.15.4) the data being sent in a single packet is limited to approx. 100 bytes.

5.2.2 Ethernet Backbone

On the other hand, the case study scenario and the proposed architecture assume reliable communication links between resource-poor and resource-rich nodes. In such settings jDEECo would introduce unnecessary overhead that comes mainly from data replication techniques employed internally by the framework implementation. As such, for this part of the architecture we aim to build on existing VM-based approaches suggested by Satyanarayanan et al. [17] and Ha et al. [6]. By that, we leverage on both flexibility with respect to software stack used inside the VM being offloaded to the cloudlet, and generic mechanisms (dealing only with the VM images and overlays) employed in the cloudlet management. Those characteristics enable further sharing of latency-free infrastructure with other (emergency) service providers, using proprietary software solutions.

Nevertheless, we can simplify the offloading process with the execution of Kevoree framework[4] in the cloudlet server. Since,

[3] https://github.com/d3scomp/JDEECo

[4] http://kevoree.org

Kevoree supports the concept of continuous deployment based on models@run.time and it also allows the definition of new semantic for its fundamental concepts (Node, Channel, Group, Component), we can envision an architecture with a master node in charge of deploying applications on demand. Indeed, such an idea has been partially implemented by the Kevoree team to support the deployment of components in LXC containers[5]. A Kevoree component running within a LXC Node type is waiting for messages that trigger the execution of a new Java-based Kevoree Runtime within a new LXC container. Such a runtime can deploy a set of components under additional request. We envision that both the process of executing a new Container and the process of deploying the new model in Kevoree can be atomic.

The application of this method to other Operating-System virtualization solutions as Jail is straightforward. It is also applicable with other virtualization providers as Amazon since those services provide a public interface to deploy VMs. The implementation of a Node Type must use such an interface to deploy a new instance of a VM-image with Kevoree pre-installed.

Using Kevoree decreases the network traffic. This is because, using a Kevoree model only the components in the application need to be specified and for each component the associated dependencies are then downloaded from the network. Thus, the time to offload an application to the cloudlet or the cloud is only the time to execute the new VM plus the time of downloading the components. This approach has two advantages: first, on using a VM-overlay this requires more bandwidth as it requires the deployment of all the associated components; and second, the intensive communication is no longer between low-power devices and the cloudlet, but between the cloudlet and Internet. On the other hand, this requires the pre-installation of Kevoree on each cloudlet instance. Nonetheless, assuming the pre-installation of Java and a standardized model@runtime (based for instance on OSGi) is a plausible solution at mid-term.

6. RELATED WORK

Our approach tries to bridge the gap between models of composition and execution tailored to the domain of cyber-physical systems, and computation offload from mobile devices to the cloud. The goal is to augment the capabilities of constrained devices while keeping power consumption low. Related areas of research are Mobile Cyber-Physical systems [11] and Edge/Fog computing [3].

Satyanarayanan et al. [17] were the first to introduce the term "cloudlet" as an approach to bring the cloud closer to the edges of the network and to support the non-disruptive execution of both interactive and resource-demanding applications in mobile devices. In [6] the authors evaluate the approach of delegating intensive tasks to the cloud infrastructure. The results show the need of a resource-rich-layer physically closer to end-user devices since it is impossible to satisfy all non-functional requirements with the cloud due to high latency. Likewise, the authors of [16] propose the combination of cloud and cloudlet infrastructures as targets for resource-intensive jobs. Mobile nodes execute a heuristic, based on the trade-off between minimal delay and higher throughputs, to decide where to offload a job, either to the cloud or the cloudlet. The work also presents a mechanism to route answers to the source mobile when the nearest cloudlet changes. Preliminary results suggest under which conditions the cloudlet approach outperforms the cloud approach. However, such approaches cannot provide service when neither connection to the cloud nor to the cloudlet is available. In our framework, it is essential to maintain system operation even when no resource-rich infrastructure is available. Moreover, such approaches consider that both the cloudlet and the cloud execute the same jobs. Instead, we propose to separate at design time the tasks with real-time response needs that are executed in the cloudlet from those that are executed offline in the cloud.

Simanta et al. [18] propose approaches to decrease start-up time of offloaded applications in cloudlets. To avoid the offload of a full VM-Image, they propose the construction of a VM-Overlay, which is the difference between a VM-Image with specific applications and a VM-Base-Image within the cloudlet infrastructure. Mobile devices send such an overlay to a cloudlet server, which in turn synthesizes the image and instantiates the fully-customized VM. The usability in highly mobile environments where the overlay must be sent to different cloudlet servers over and over is limited because the size of overlays is not negligible. On the other hand, Unikernel [13] proposes a solution based on the observation that in many cases a traditional software stack is not required on a hosted virtual machine because few features of the underlying system are used. Hence, the authors advocate the construction of a specialized operating system with small software stack that comprises only essential functionality for each application. The resulting VM-Image is on the order of kilobytes in many cases, which eases the rapid deployment to remote locations. Nevertheless, we intend to use the former approach in our framework implementation because it allows access to legacy code, which is not possible when following the Unikernel approach.

In [16], the authors also face the problem of task delegation and data sharing for mobile devices in a dynamic network environment. They present a framework to use the energy of mobile devices in a more efficient way, by combining execution of tasks both on traditional cloud infrastructure and on ad-hoc mobile cloud. The framework uses a service discovery mechanism and offload decision algorithms to find the lowest offloading cost. Nodes rely on the ad-hoc mobile cloud if the cloud infrastructure is not available. A device accesses a service either through peers of connected nodes in the ad-hoc mobile cloud or directly via the cloud. The decision of choosing the offload target depends on two factors: the historical cost of the computation and the cost of communication with the nodes providing the service. However, the authors only consider two classes of infrastructure for doing computation and neither distinguish between mobile devices nor consider resource-rich infrastructures other than the cloud.

The authors of [3] and [8] try to solve the problem of performing resource-intensive jobs by delegating the execution to as many devices as possible and not only to a dedicated infrastructure as the cloud or the cloudlet. Their thesis is that, nowadays, many installed devices are capable of executing offloaded jobs; but there is a lack of proper software frameworks and protocols to achieve this goal. Their idea is to define a framework with different logical layers, which reifies a hierarchical relationship within the network (e.g., sensor/actuator nodes, mobile nodes, routing devices in the network and cloud infrastructure). Each node executes processes as part of the application and communicates with any reachable node in the network since the aim is to provide a framework for distributed computing. Nonetheless, the framework encourages communication that follows the hierarchy due to performance advantages. Within this scheme, mobile devices offload tasks to a resource-rich node

[5] http://linuxcontainers.org/

responsible for its region and they change the target-region node while on the move. Simulation-based evaluations show low-latency in large-scale systems with wide geospatial distribution. While their work focuses on finding an efficient programming model for the so-called "Mobile Fog" paradigm, we adopt an architecture-based approach, which is platform and language agnostic.

7. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a reference architecture that can act as a template for the structuring of applications deployed on Compute-Intensive Pervasive Systems. Our reference architecture approach is designed to handle high dynamicity and low latency requirements of CIPS. Furthermore, we have used a model-driven approach to validate the proposed reference architecture using two component-based frameworks, DEECo and Kevoree. We strongly believe our approach improves on availability of CIPS and using a cloudlet approach reduces the latency, thereby enhancing the operability of CIPS.

As future work we plan to fulfill the design view and validate it by real experiments. A couple of points we are considering as next steps:

1. **Different experimentations**. We plan to do different experiments using different reference architecture instantiation (i.e. Kevoree, DEECo) of our reference architecture to evaluate their performance and latencies.
2. **Focus on the physical attributes**. We will focus on extending the reference architecture to correlate with attributes of the physical world, and consider them in data propagation part. For example, we plan to instantiate our reference architecture with models and techniques that build on opportunities offered by cyber-physical systems (e.g., geospatial routing in data propagation, gossip-based communication between components in MANETs, etc.).

8. ACKNOWLEDGMENTS

This work has been supported by the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840. The authors would like to thank Petr Hnetynka and Jaroslav Keznikl for the constructive feedback and discussions.

9. REFERENCES

[1] Bachmann, F., Bass, L., Clements, P.C., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R. and Stafford, J.A. 2010. *Documenting Software Architectures: Views and Beyond, Second Edition*. Addison-Wesley Professional.

[2] Behzadan, A.H. and Kamat, V.R. 2005. Visualization of construction graphics in outdoor augmented reality. *Proc. of WSC'14* (Orlando, FL, USA, Dec. 2005), 1914–1920.

[3] Bonomi, F., Milito, R., Zhu, J. and Addepalli, S. 2012. Fog Computing and Its Role in the Internet of Things. *Proc. of MCC'12* (Helsinki, Finland, Aug. 2012), ACM, 13–16.

[4] Bures, T., Gerostathopoulos, I., Hnetynka, P., Keznikl, J., Kit, M. and Plasil, F. 2013. DEECo - an Ensemble-Based Component System. *Proc. of CBSE'13* (Vancouver, Canada, Jun. 2013), ACM, 91–100.

[5] Fouquet, F., Barais, O., Plouzeau, N., Jezequel, J., Morin, B. and Fleurey, F. 2012. A Dynamic Component Model for Cyber Physical Systems. *Proc. of CBSE'12* (Bertinoro, Italy, Jun. 2012), 135–144.

[6] Ha, K., Pillai, P., Lewis, G., Simanta, S., Clinch, S., Davies, N. and Satyanarayanan, M. 2013. The Impact of Mobile Multimedia Applications on Data Center Consolidation. *Proc. of IC2E'13* (San Fransisco, CA, USA, Mar. 2013), 166–176.

[7] Hölzl, M., Rauschmayer, A. and Wirsing, M. 2008. Software Engineering for Ensembles. *Software-Intensive Systems and New Computing Paradigms*. M. Wirsing, J.-P. Banâtre, M. Hölzl, and A. Rauschmayer, eds. Springer Berlin Heidelberg. 45–63.

[8] Hong, K., Lillethun, D., Ramachandran, U., Ottenwälder, B. and Koldehofe, B. 2013. Mobile Fog: A Programming Model for Large-scale Applications on the Internet of Things. *Proc. of MCC'13* (Hong Kong, China, Aug. 2013), ACM, 15–20.

[9] Huang, C.-M., Lan, K. and Tsai, C.-Z. 2008. A Survey of Opportunistic Networks. *Proc. of AINAW'08* (GinoWan, Japan, Mar. 2008), 1672–1677.

[10] Keznikl, J., Bures, T., Plasil, F. and Kit, M. 2012. Towards Dependable Emergent Ensembles of Components: The DEECo Component Model. *Proc. of WICSA'12* (Helsinki, Finland, Aug. 2012), 249–252.

[11] Kim, B.K. and Kumar, P.R. 2012. Cyber–Physical Systems: A Perspective at the Centennial. *Proceedings of the IEEE*. 100, Special Centennial (2012), 1287–1308.

[12] Kumar, K. and Lu, Y.-H. 2010. Cloud computing for mobile users: Can offloading computation save energy? *Computer*. 43, 4 (Apr. 2010), 51–56.

[13] Madhavapeddy, A., Mortier, R., Rotsos, C., Scott, D., Singh, B., Gazagnaire, T., Smith, S., Hand, S. and Crowcroft, J. 2013. Unikernels: Library Operating Systems for the Cloud. *SIGPLAN Not.* 48, 4 (Mar. 2013), 461–472.

[14] Maiden, N. a. M., Manning, S., Jones, S. and Greenwood, J. 2005. Generating requirements from systems models using patterns: a case study. *Requirements Engineering*. 10, 4 (Oct. 2005), 276–288.

[15] Morin, B., Barais, O., Jezequel, J.-M., Fleurey, F. and Solberg, A. 2009. Models at Runtime to Support Dynamic Adaptation. *Computer*. 42, 10 (2009), 44–51.

[16] Ravi, A. and Peddoju, S.K. 2013. Energy Efficient Seamless Service Provisioning in Mobile Cloud Computing. *Proc. of SOSE'13* (San Francisco, USA, Mar. 2013), IEEE, 463–471.

[17] Satyanarayanan, M., Bahl, P., Caceres, R. and Davies, N. 2009. The Case for VM-Based Cloudlets in Mobile Computing. *IEEE Pervasive Computing*. 8, 4 (Oct. 2009), 14–23.

[18] Simanta, S., Ha, K., Lewis, G., Morris, E. and Satyanarayanan, M. 2013. A Reference Architecture for Mobile Code Offload in Hostile Environments. *Mobile Computing, Applications, and Services*. D. Uhler, K. Mehta, and J. Wong, eds. Springer Berlin Heidelberg. 274–293.