The ARCHADE: Ubiquitous Supercomputing for robotics. Part I: Philosophy

Leonardo Camargo-Forero · Pablo Royo · Xavier Prats[1]

Universitat Politècnica de Catalunya - ICARUS Research group
Esteve Terradas, 5. 08860, Castelldefels, Catalonia - Spain
E-mail: leonardo.camargo@upc.edu
[1] Serra Hunter Fellow

Abstract

In this work, we introduce *Ubiquitous Supercomputing for robotics* with the objective of opening our imagination to the development of new powerful heterogeneous multi-robot systems able to perform all kind of missions. Supercomputing, also known as *High Performance computing* (HPC) is the tool that allows us to predict the weather, understand the origins of the universe, create incredibly realistic fantasy movies, send personalized advertisement to millions of users worldwide and much more. Robotics has been mostly absent in its use of HPC but some previous works have lightly flirted with it. With the findings presented in here, we propose a ubiquitous supercomputing ontology, which allows describing systems made up of robots, traditional HPC infrastructures, sensors, actuators and people and exhibiting scalability, user-transparency and ultimately higher computing efficiency. Moreover, we present a technology called *The ARCHADE*, which facilitates the development, implementation and operation of such systems, and we propose a mechanism to define and automatize missions carried out by ubiquitous supercomputing systems. As a proof of concept, we present a system depicted as *Tigers VS Hunters*, which illustrates the potential of this technology. The results presented in here are part of a two series work introducing The ARCHADE. This first delivery presents its philosophy and main features. Correspondingly the second part will present a set of use cases and a complete performance benchmark. Supercomputing is part of our lives and it can be found in many research and industrial endeavors. With the ubiquitous supercomputing ontology and The ARCHADE, supercomputing will become part of robotics as well, bringing it therefore everywhere.

Keywords

Ubiquitous Supercomputing · The ARCHADE · Ubiquitous Supercomputing ontology · High Performance Robotic Computing - HPRC · HPRC cluster · Parallel Robotic Computing Node - PRCN · General-purpose computing mission

1. Introduction

While it is not entirely accurate to say that computing is everywhere, it is fair stating that the bases and mechanisms for *Ubiquitous Computing* have been proposed and implemented. Starting from the Internet and up to all kind of existing protocols for the integration of different-purpose applications, computing is part of our lives all the time. Within the computing realm, a particular type of systems, which are of interest for this work, are those that exhibit *High Performance Computing (HPC)*, also known as *Supercomputing*.

HPC consists of a set of infrastructures, techniques and parallel computing libraries designed specifically to speed up the computation of problems that cannot be solved in reasonable time using standard software or individual personal computers. Examples of such problems are forecasting the weather, simulating physics phenomena, analyzing DNA

samples, drug manufacturing, aircraft design, movie rendering, sending personalized advertisement to millions of users, etc.

HPC software is such that it is written using a parallel computing library, e.g. Message Passing Interface (MPI) [1] or Compute Unified Device Architecture (CUDA) [2], which allows it to be executed correspondingly on a multi-core (e.g. a *cluster of computers*) or many-core (e.g. Graphics Processing Unit - GPU) computing infrastructure.

A cluster of computers is the *de facto* HPC infrastructure; it consists of a set of computers (nodes) connected through a Local Area Network (LAN), where a set of *HPC software* layers, are installed and configured in each of the computing nodes. These software layers allow the HPC cluster to be scalable (i.e. to allow the inclusion of new nodes) and to look, from the end-user perspective, as a single computing unit (user-transparency). Such features extend to HPC software as well.

In robotics, works such as [3–11], showed at research and industrial level the impact of HPC within robotics. However, its purpose has been solely to exploit HPC computing efficiency and while this is the main objective of HPC, the scalability and user-transparency features are useful for the building of systems that behave and act as a single unit.

In our previous work [12], we introduce the concept of *High Performance Robotic Computing (HPRC)*, an abstraction of the ideas behind HPC for multi-robot systems. In these systems, a robot is a *Parallel Robotic Computing Node (PRCN)* and a set of robots is a HPRC cluster, able not only to process data or execute robotic tasks more efficiently on board, but more importantly to resemble a scalable single robotic unit. To prove that HPRC can be more than a simply speeding up tool in robotics context, we presented, in [12], MPI software able to control the motion of any quantity of robots (e.g. Unmanned Aerial Vehicles – UAVs, Unmanned Ground Vehicles – UGVs, etc.).

Multi-robot motion is not by default a task requiring HPC in its traditional sense (i.e. as a speeding up tool), but by using parallel libraries such as MPI, the parallel motion software provides scalability (i.e. able to be used in one up to N robots) and user-transparency since only one ground station is required to control a multi-robot system.

Following these ideas, in this work, we present the concept of *Ubiquitous Supercomputing* in the context of robotics. As its name suggests, Ubiquitous Supercomputing aims at having HPC everywhere. Previous works, such as [13] or [14], proposed a general idea of the concept; however, these works did not consider robots and they focused only on the efficiency feature of HPC.

Given that a robot, coupled with a companion computer, can be set as a PRCN able to execute on-board software, as we have shown in [12] and [15], in this work, we propose an ontology for the definition and implementation of *ubiquitous supercomputing systems*. Such ontology allows the integration of HPRC entities, traditional HPC entities, computing-less devices and people into single systems, which exhibit scalability, user-transparency and computing efficiency. In the following sections, we define the *ubiquitous supercomputing ontology* and introduce a novel framework / middleware, depicted as *The ARCHADE* for the deployment, implementation and operation of *ubiquitous supercomputing Systems*.

As a test case of The ARCHADE, we present a system example, called *Tigers VS Hunters*, simulated on a *ubiquitous supercomputing infrastructure* composed of a set of HPC and HPRC nodes. The system example is simulated but The ARCHADE is designed to facilitate a transparent transition from experiments to real world applications.

Several solutions, providing deployment and creation of multi robot systems exist at research level as presented in [16], where a very complete set of ubiquitous robotics testbeds was explored. However, ubiquitous robotics differs from the concepts exposed in here. *Ubiquitous supercomputing aims at reusing HPC technologies, strategies, etc., in the context of robotics*. Conversely, ubiquitous robotics consists on deploying robots everywhere. While both aspire to similar objectives, our approach differs by the means of achieving such objectives.

The main difference between the analyzed solutions in [16] and The ARCHADE is that none of them provides, nor it uses HPC as its foundation. In The ARCHADE, the framework and the middleware use and interact with the subjacent HPC software layers. However, [16] showed a remarkable review and identified not only existing solutions, but also provided guidelines to be taken into consideration for the development of new solutions.

The conclusions and guidelines drawn in [16] and The ARCHADE's approach towards them is presented in table 1.

Table 1. The ARCHADE VS. Current robotics testbeds and guidelines for new solutions.

.

Conclusions drawn by [16]	**The ARCHADE's approach**
Significant number of non-integrated testbeds and few highly-integrated testbeds	There exist several testbeds for multi-robot systems but integration between systems, developed with different testbeds, is scarce. The ARCHADE uses common Linux technologies and communications are based on TCP / IP protocols. This favors possible integration with all kind of systems.
Multi-robot testbeds tendency to be ad-hoc	The majority of testbeds use ad-hoc approaches, i.e. designed for specific purpose. The ARCHADE is fully general purpose. Multipurpose is achieve via the tasks' software. Information about mission, tasks and related software will be introduced in section 2. In [16], WSN (Wireless Sensor Networks) testbeds were analyzed as well. There exist more WSN options for general-purpose missions than multi-robot testbeds. The Tigers VS Hunters systems example shows that The ARCHADE can be used for any kind of purpose.
Lack of remote access testbeds.	The ARCHADE is fully based on traditional HPC technologies and Python software, which uses communications via TCP/IP in distributed systems. Such communications are based on an IP / Port tuple, which allows remote access to every entity.
Multi-agent coordination is the most common approach with multi-robot testbeds	The ARCHADE belongs to this category

Tendency to create higher integration and heterogeneity	The ARCHADE supports all kind of entities that can embed Linux operating systems. Furthermore, it allows using drones, rovers and planes as HPRC entities.
Increasing adoption of reusable software	The ARCHADE uses standard well-tested HPC software for Linux distributions
Guidelines provided by [16]	**The ARCHADE's approach**
General purpose	HPC standard technologies, multipurpose / multiuser HPC / HPRC infrastructure. Framework, middleware and general purpose computing mission
Modular and flexible architectures	The ARCHADE inheritable classes. Independent and modular middleware and framework services, TCP / IP communications
Openness	The ARCHADE is not open-source but cooperation is encouraged.
APIs, reusable code and standardized interfaces	The ARCHADE's python libraries (API). In addition, *agents* can be modified to add extra functionalities. Furthermore, communications mechanisms can be integrated or replaced with other technologies, e.g. ROS [17], Data Distribution Services (DDS) [18] i.e. for real time systems, etc.
Availability of suitable usability tools	Future work
Remote execution	TCP / IP communications
Experiments to real applications	The ARCHADE facilitates transition from experimentation to real- world applications. It transparently allows connection with simulated or real autopilots.

Furthermore, we propose a methodology to define a robotic mission as a directed network with interactions between nodes, based on *events* and *actions*. This methodology offers easiness for the development of general-purpose applications, one of the most desired features as exposed in [16].

The ontology here might open the door to the creation of complex distributed and heterogeneous systems able to perform all kind of missions. We believe that *Ubiquitous Supercomputing*, i.e. bringing HPC everywhere, even robotics, and considering not only its proven efficiency but also the rest of its features, holds an enormous potential. Therefore, in this work, we present the following main contributions:

- The formal definition of the concept of *ubiquitous supercomputing* in the context of robotics. Following the results of our previous work [12], where we introduced High Performance Robotic Computing as a set of strategies to create multi-robot systems resembling a scalable single robotic unit, we propose in here *ubiquitous supercomputing, as the joining of HPC, HPRC, computing-less devices and people*. As a clarification, while HPC focuses on computing efficiency, conversely HPRC and its extension, ubiquitous supercomputing focuses rather on other HPC features. Because of this reason, computing efficiency indicators are not presented.

- ❖ The ubiquitous supercomputing ontology, which defines the necessary terminology and concepts for creating and operating ubiquitous supercomputing systems. Furthermore, the ontology includes a mechanism to define and automatize missions carried out by ubiquitous supercomputing systems.

- ❖ The ARCHADE, a framework / middleware, which implements the ubiquitous supercomputing ontology and provides strategies, services, etc., to facilitate the creation and operation of ubiquitous supercomputing systems used for any kind of mission.

- ❖ The Tigers VS Hunters system example (simulator), a proof of concept of The ARCHADE.

The rest of the paper is organized as follows. Section 2 introduces the ubiquitous supercomputing general concepts, the ontology and the ideas behind ubiquitous supercomputing missions. Section 3 introduces The ARCHADE, section 4 the system example and conclusions are drawn in section 5.

2. Ubiquitous Supercomputing

In this work, we define *Ubiquitous Supercomputing* as the union of HPC, HPRC, computing-less devices (CLD) and humans as depicted in figure 1.

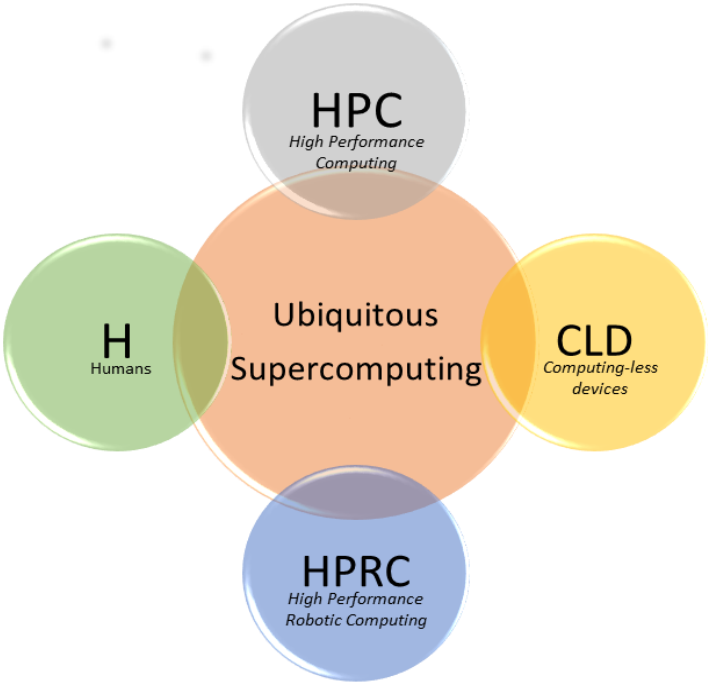


Fig. 1. Ubiquitous Supercomputing definition. Ubiquitous Supercomputing is the joining of traditional High Performance Computing, High Performance Robotic Computing, computing-less devices such as sensor networks and humans such as end-users, operators, managers, etc.

In a ubiquitous supercomputing system, the human plays an important role as the entity with the highest hierarchy. Conversely, computing-less devices are those that do not exercise computing but rather produce or consume data. While a PRCN could embed this type of devices, e.g. sensors or actuators, in this case, such devices are considered part of the HPRC

setting. CLD refers therefore to independent computing-less devices, e.g. sensor networks, vehicle networks, etc.

Before introducing the ubiquitous supercomputing ontology, we summarize relevant HPRC concepts in table 2. For detailed information, please refer to [12].

Table 2. High Performance Robotic Computing main concepts. Information taken from [12]

HPRC concept	Definition
General-purpose computing robot	A robot embedded with a general-purpose companion computer. A robot can be used or reused for different missions by simply replacing the software associated to it.
Parallel Robotic Computing Node (PRCN)	A robot in which parallel software libraries (e.g. MPI, CUDA) are installed in its companion computer or embedded computing cards (e.g. GPUs).
High Performance Robotic Computing	Set of strategies that allow the deployment of multi-robot systems that behave and act as a single distributed unit able to exploit HPC features such as scalability, user-transparency, centralization / distribution and ultimately computing efficiency.
HPRC cluster	A set of PRCNs connected via a wireless LAN and installed with the HPC software layers adapted to the specific conditions of the world of robotics.
Task	A software linked to a robotic task, requiring a set of resources during a specific time (walltime)
Resources	Companion computers' cores, embedded computing cards (GPUs, FPGAs, etc.), cameras, sensors (e.g. GPS, LIDAR, etc.), actuators (gimbals, weaponry, robotic hand, etc.) and batteries, etc. Resources could be also software-based e.g. licenses, etc.
General-purpose computing mission or simply mission	A HPRC mission is a directed graph of HPRC tasks. The direction implies that a task must be done before the next task can be started.

In a HPC or HPRC cluster, computing nodes are classified as *Master or Slaves*. The Master node is responsible of coordinating the entire infrastructure and implements the server side of each of the HPC software layers. The slave nodes are used for computation and implement the client side of each of the HPC software layers. Furthermore, the master node acts as a frontend for the end-user, providing therefore user-transparency. The HPC software layers are the operating system, the file system, the user system, the batch system and the applications layer. Furthermore, an implementation of the MPI standard, e.g. Open MPI

[19], must be installed in order to run parallel software. For a detailed description of the HPC software layers and their specific adaptation to robotics settings (HPRC clusters), please refer to [12].

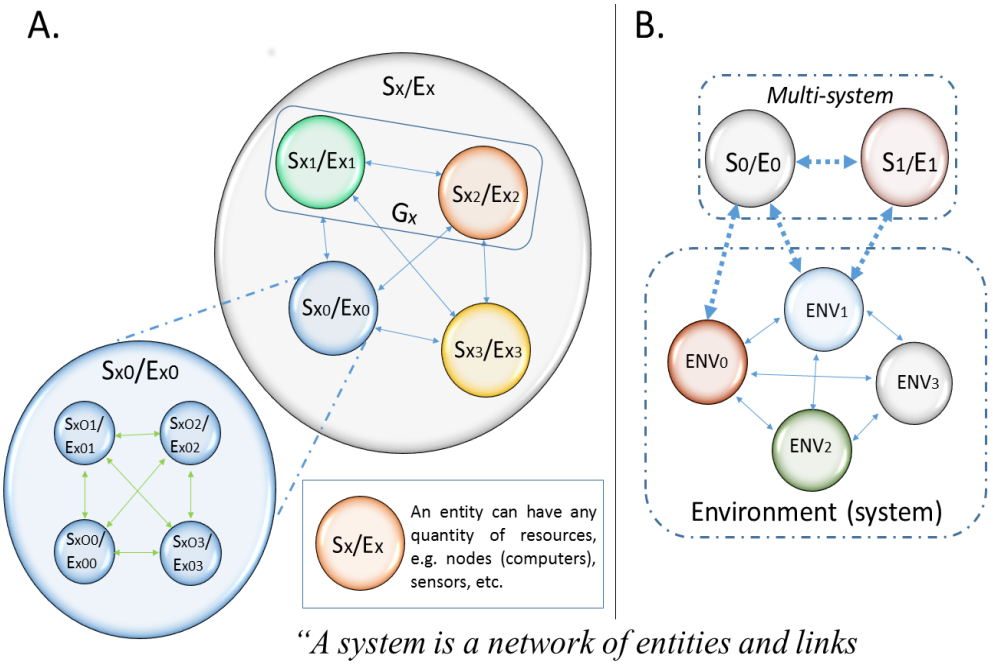
Since HPRC and ubiquitous supercomputing for robotics are new concepts, in the next section we proposed an ontology to describe systems belonging to such classifications.

2.1.Ubiquitous supercomputing ontology

Ubiquitous Supercomputing systems or simply systems, as we refer for the rest of this paper are described accordingly to the following main statements:

"A system is a network of entities and links

A System is an Entity"



"A system is a network of entities and links

A System is an Entity"

Fig. 2. Ubiquitous supercomputing ontology. (**A**) *A system is a network of entities and links. A System is an Entity*. A system is composed of entities where each entity can be a system as well. Consequently, a system is an entity. (**B**) A system of type *multi,* interacting with an *environment* via interfaces (blue thick dotted lines). Systems *0* and *1* communicate using interfaces as well. In addition, each system can have a different environment to which it interacts with. *Sx* = System X, *Ex* = Entity X, *Gx* = Group X

Entities are the bricks of a system and correspond to elements interacting with each other. Examples of entities are robots, servers, people, sensors, etc. An entity is by default a computing-element, however entities that do not exercise computing, e.g. a sensor, are defined as *dummies* in the ontology.

Furthermore, entities can be grouped as desired. The *group* concept represents that idea. For example, consider a mission carried out by a set of UAVs and a set of UGVs. The UAVs can be grouped as an *air-group*, while the UGVs as a *ground-group*. In addition, a HPC

cluster is a group of servers. Entities hold specific *roles* within the groups they belong to; such roles can be of any type or represent any classification but a particular important role is the *system controller*. This role represents the system frontend. For example, in a multi-robot system, the system controller role can be assigned to the ground station controlling the robots. Roles can change dynamically, e.g. during the execution of a mission and an entity can hold different roles within the same group or between different groups.

A *link* represents the connection between entities i.e. the ability of an entity to communicate with other entities. More specifically, a link represents the communications technology between a pair of entities. The nature of the systems proposed in here is by default dynamic e.g. the links in the network could appear or disappear during the execution of a mission. While a traditional HPC cluster is a wired-based infrastructure, which usually is not subjected to sudden disconnection, an HPRC setting could experience entities' disconnections given its moving essence and the wireless connectivity inherit issues, which leads to a dynamic network topology. However, a system, in its ideal condition, is represented by a full graph.

The first statement, "*A system is a network of entities and links",* uses terminology from the field of network science (e.g. *node*, *link*). However, the term *node* from network science is replaced for *entity* in the ontology. This way, we preserve HPC terminology. The term *node* refers to a computing unit, e.g. a single computer, an embedded computing card, etc. Entities can have as many nodes as desired. Even an entity could embed a full HPC cluster. Same as with the entities, nodes hold roles. For example, a *master* node and the *slave* nodes.

The advantages of understanding a system as a network are many, for example:

- If a network represents a system, mathematical and conceptual approaches from graph theory and network science can be applied to the system's modeling, understanding and improvement.

- A network is scalable because it allows the adding of new nodes (entities) and new links, maintaining the autonomy of each of the entities.

- A network can be robust and resilient depending of its specific topology.

The second statement, "*A System is an Entity*", allows the creation of *systems of systems*, as portrayed in figure 2 – A. Furthermore, the ontology describes system equipped with the following features:

- *User-transparency / cohesion*: From a computer science point of view, user-transparency relates to the capacity of a system of being perceived, from the user's perspective, as a single element, rather than a set of interconnected elements. Under the scope of ubiquitous supercomputing, we use the term user-transparency interchangeably with *cohesion* to state that a system is a cohesive and *independent* group of entities.

- *Centralization/distribution*: A cohesive system is *centralized* because it exhibits cohesion and it is *distributed* because it is composed of independent systems and entities, interacting with each other. For example, a multi-robot system is distributed because each of its robots can be given sufficient autonomy to execute a particular mission and it is centralized because a single ground station can control all the robots.

❖ *Scalability*: Understanding a system as an entity and therefore an entity as a system, where each element is independent, allows easy inclusion and removal of entities, nodes, etc., leading to scalable systems.

Systems are classified as *single* or *multi*. In a *multi-system* setting (system of type *multi*), systems communicate with each other via an *interface*. An interface is a communication strategy, which can involve a subset of the entities within two systems. It is defined by a communications technology (link idem), a specific IP / Port tuple or any other approach. Moreover, a system interacts with an *environment* while performing a mission. The environment is a system as well, according to the ontology here presented, as shown in figure 2 - B.

Systems can have as many systems, entities and groups as desired but a single system should be cohesive, i.e. all entities should focus in a particular mission or a related group of tasks within a mission.

Furthermore, a system is a *hierarchical network,* where entities can have a different hierarchy. However, the default hierarchy approach is, from top to button: human, system controller and other entities. An entity's hierarchy represents the level of authority that such entity exercises over other entities. Therefore, an entity can accept *orders* from entities with higher hierarchy. The ubiquitous supercomputing ontology includes a complete hierarchy scheme based on groups and tasks. However, this scheme is outside the scope of this work.

A ubiquitous supercomputing system can be used for different missions considering that a robot, a server, etc., are *general-purpose computing devices*, as presented in [12]. Therefore, in the next subsection, we present the concept of *general-purpose computing mission*.

2.2.General-purpose computing mission

A *mission*, being carried out by a system, consists of a set of interconnected tasks or a directed network as shown by figure 3. The direction of the links indicates the task execution order.

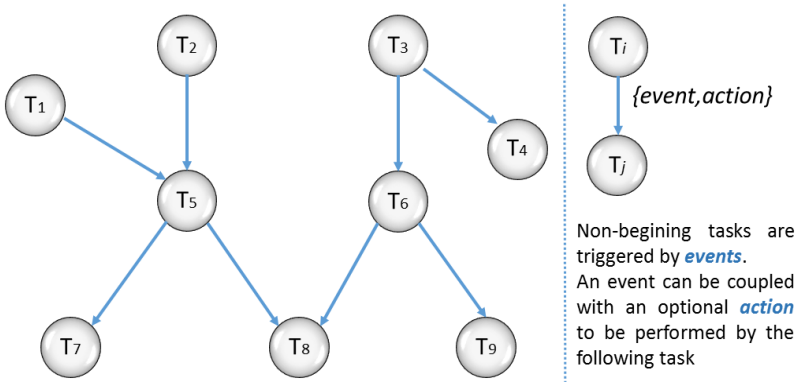


Fig. 3. General-purpose computing mission. A Mission is a directed network of tasks where each task (T1, T2… TN) is a software that requires a set of resources (e.g. computing cores, cameras, sensors, actuators, etc.) in order to be executed

A task is defined as a *software* that requires a set of *resources* for its execution. Nodes, computing cores, robotics sensors and actuators, software licenses, etc., are all type of

resources within the scope of ubiquitous supercomputing. A task resembles the concept of *job* in traditional HPC.

Tasks are assigned to entities, holding the task's required resources. Moreover, tasks can be assigned to a subset of the system's entities. For example, a task could consist of a MPI software requiring a set of CPUs not available in a single entity's nodes, therefore the software will be executed in different CPUs that are distributed among different entities.

Tasks are classified according to the types:

- *Blocking*: Tasks' blocking types are *go* and *lock*. A task is of type *go* if the entity executing the task can proceed with its tasks' children. A task if of type *lock* if the entity executing the task cannot proceed until the task's parents are completed.

- *Execution*: Tasks' execution types are *none*, *exclusive*, *parallel* and *distributed*. A task is of type *none* if there is not associated software to it. A task is of type *exclusive* if it is performed by one single entity. A task is of type *parallel* if the associated software is either Single Instruction Multiple Data (SIMD) or Multiple Instruction Single Data (MISD) according to Flynn's Taxonomy [20]. A task is also parallel if the entities are performing the same software on a different geographic area, e.g. a sub area of the mission's area. A task is of type *distributed* if its associated software is of type Multiple Instruction Single Data (MIMD) (e.g. using MPI). Parallel and distributed tasks are used for computationally demanding problems. This way, ubiquitous supercomputing systems offer *computing efficiency*.

- Necessity: Tasks' necessity types are *mandatory* and *optional*.

- Topology: Tasks' topology types are *begin*, *path* and *end*

- Priority: Tasks' priority types are *low*, *medium* and *high*.

Missions should be divided in independent tasks, where each task is linked to a different software. Non-beginning tasks (i.e. *path* and *end*) are triggered by *events*, this is: one of the outputs of each software is an event. Optionally, an output event can be coupled with an *action*. Consequently, an action is performed by the triggered task.

The objective of the ontology, including the missions' definition as presented in this section, is to propose a set of guidelines for the implementation of systems offering scalability, user transparency, computing-efficiency and other traditional HPC features.

In the next subsection, we introduce a technology called *The ARCHADE*, a framework / middleware, which facilitates the creation of such systems

3. The ARCHADE: Ubiquitous Supercomputing framework and middleware

The ARCHADE is a component-based *ubiquitous supercomputing framework / middleware*, written in Python 2.7, designed for the *automatic* deployment, implementation and operation of ubiquitous supercomputing systems. It consists of a set of classes, a set of templates to describe the system and its mission and a set of services to be used for the building and operation of systems described according to the ontology.

Figure 4 introduces The ARCHADE framework, which consists of a set of class layers (ontology, things and robotics), templates and services, which allow the creation of systems that can be described according to the ontology.

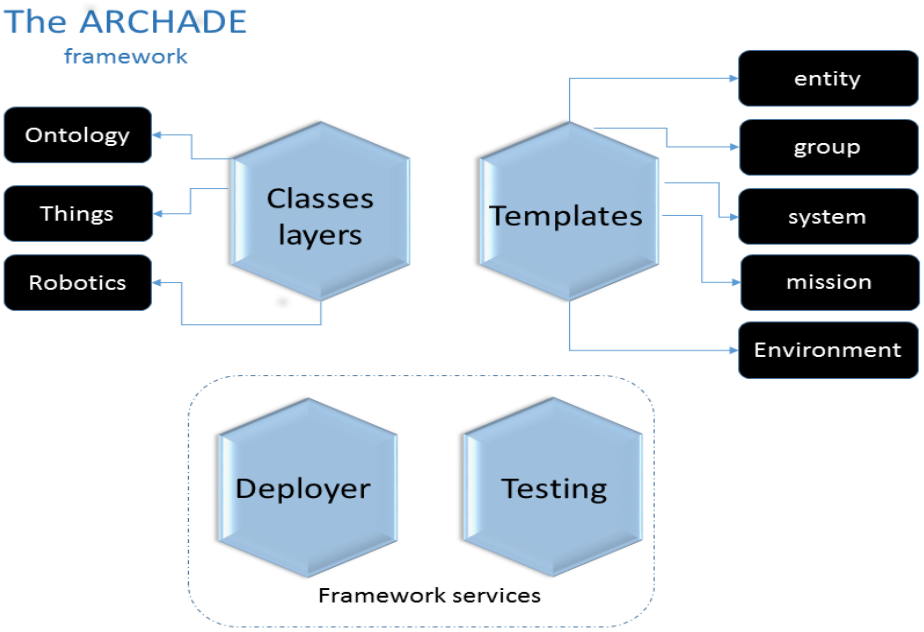


Fig. 4. The ARCHADE framework. The framework consists of a set of classes and templates that allow the description and implementation of a ubiquitous supercomputing system. Furthermore, the framework provides services for deployment of HPC or HPRC settings, testing and monitoring.

The ARCHADE allows the creation of any kind of systems, used for any kind of missions, by using the framework classes and templates. The classes were designed to allow the creation of new classes in each layer or even new layers by using the *inheritance principle* from OOP (Object-Oriented Programming). While ubiquitous supercomputing differs from the ideas behind the Internet of Things, given that the systems, described in here, are fundamentally HPC-based, the layer class *thing* opens up the spectrum of possibilities. For example, a sensor or an actuator (CLD) can be described using the default classes, or using classes inheriting from the framework.

The framework also includes a set of XML templates to describe a system:

- Entity: The *entity template* is used to describe entities within the system, resources per entity, etc.

- Group: The *group template* is used to describe groups of entities and corresponding roles within

- System: The *system template* is used to describe the system i.e. entities, groups, etc.

- Mission: The *mission template* is used to describe the tasks and its interdependence (network of tasks)

- Environment: The *environment template* is used to describe the system's environment

The templates are used to set the *system's description*. The ARCHADE is designed to allow a system to be modifiable, i.e. to include new entities, new groups, etc., by simply changing the system' description, providing therefore scalability. Furthermore, the system is reusable for different missions.

Moreover, the framework includes a set of services:

- Deployer: The *deployer service* allows the automatic deployment of a HPRC cluster of computers, i.e. the automatic installation and configuration of the HPC software layers, in computing units using Ubuntu, Raspbian or Kali Linux. The service consists of a ROS (Robotic Operating System) package called the *HPC-ROS package* [15]. The package has been tested on Raspberry Pi computing cards and common desktop computers.

- Testing: The *testing service* allows simple MPI testing of a HPC / HPRC setting. However, the official HPC performance test Linpack [21] is currently being automatized and included in the service. Linpack test estimates the amount of FLOPS (FLoating point Operations Per Second), a quantity used to estimate the computing power of a HPC setting. These tests allow the user(s) to find out the computing power they can exploit while using the system and use such knowledge to revise their needs.

By using the framework, The ARCHADE's user, i.e. the one building systems, need to focus only on creating the software associated to each task i.e. *the mission software*. This way, the technology is very flexible and abstracts the *HPC / HPRC system platform* (e.g. the robots, servers, etc.) from the mission software, by means of its middleware component services.

The ARCHADE middleware services are presented in figure 5. Such services are *communications*, *initialization*, *live*, *matching* and *synchronization*. All the services were designed within a component-based / modular approach, which allows the inclusion of more services that can interact or reuse existing services. In fact, currently more services are being developed.

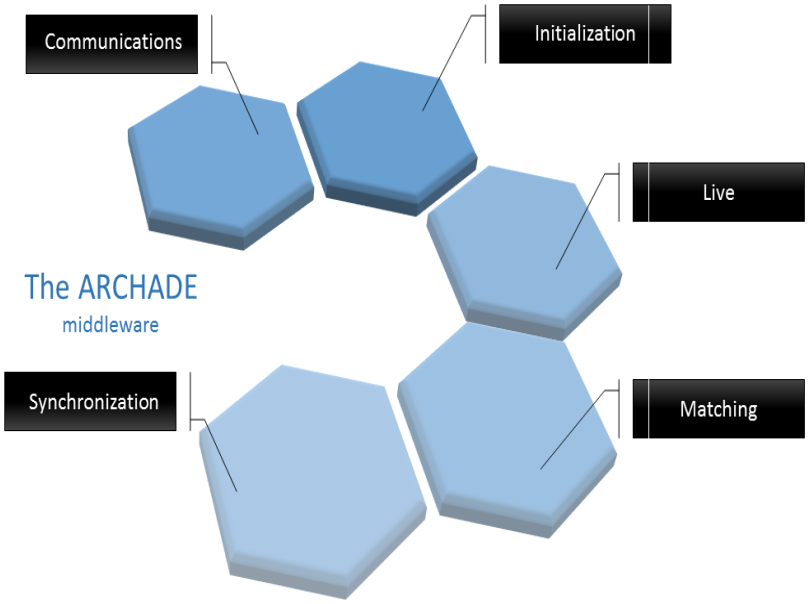


Fig. 5. The ARCHADE middleware. The middleware includes the services: *live*, *synchronization*, *communications*, *initialization* and *matching*.

In a HPC setting, such as a HPC cluster of computers, the middleware layer is the *Batch System* because it provides user-transparency by monitoring all the distributed resources. Moreover, it exercises job scheduling policies; both features lead the user to perceive the cluster as a single unit. In this sense, The ARCHADE middleware performs a similar approach in the context of ubiquitous supercomputing but extends its features and adapts in order to support multi-robot settings satisfying the ontology presented in the previous section. The middleware services are:

- Communications: The *communications* service provides methods for message exchange between entities, specifically for events and orders exchange.

- Initialization: The *initialization* service uses the framework templates to create entities, groups, systems, etc.,

- Live: The *live* service is used during the execution of a mission. Each entity executes a *live service agent, which* controls the entity. The agents aim at providing autonomous behavior by implementing methods for autopilot control, motion, mission execution, etc., for robotic entities. Furthermore, the live service includes methods for the creation of events and orders. Events can include data related to the event's occurrence. The user can make use of such methods within the software associated to tasks (mission software).

- Matching: The *matching* service compares the tasks' requirements, in the mission template, with the available resources in the system platform, assigning tasks to those entities satisfying the requirements. This process is automatically executed based on the entities and mission templates.

- Synchronization: The *synchronization* service provides user-transparency and centralization/distribution by synchronizing data (e.g. images, etc.), events, orders, etc., between all entities requiring it.

There are two types of live service agents, *system controller agent* and *entity agent*. The system controller agent has a centralized image of the entire distributed system platform, including all the data collected during the mission and has the highest hierarchy after the user, which allows it to command orders to all the entity agents. It also distributes the tasks among the entities and monitor the mission execution. The entity agents are in charge of controlling its corresponding entity, i.e. to execute the assigned tasks, interact with the autopilot, in case of a robotic entity, etc.

Both type of agents can be fully automatic or controlled by the user. This provides centralization / distribution to the ubiquitous supercomputing system. Furthermore, the agents can be modified or enhance to adapt the agent to the specific mission or system. All the middleware services, including the agent's functionalities, require the existence of a HPRC cluster deployed upon the entities' nodes.

The ARCHADE includes a complete set of general functions for interaction with Linux operating systems, the HPC software layers, network analysis and monitoring, etc. Furthermore, it includes MPI software for the motion control of a multi-robot system as introduced in [12].

Finally, a ubiquitous supercomputing system is *a multipurpose / multiuser system*. This means that different users, i.e. the ones interacting with the system, can make use of the resources for different purposes (i.e. missions) at the same time. For example, a pilot can control de motion of a robot, while another user makes use of the cameras, etc.

Next sub section presents a proof of concept of The ARCHADE, a system example called *Tigers VS. Hunters*. The system was built using the framework and the middleware and it consists of a Software in the loop (SITL) simulator.

4. Tigers VS. Hunters' system example

In this subsection, we present a system built with The ARCHADE. The system uses specific classes and mission software developed for it. The system example, depicted as *Tigers VS. Hunters*, consists of one human operator, a set of simulated robots (two UAVs and two UGVs), a ground station and a HPC cluster of computers, used for a mission described as:

- Localization of a group of hunters and a group of tigers distributed on a geographic area
- The protection of the tigers from the hunters and
- The identification of the hunters.

The ARCHADE framework allows the creation of new classes by inheriting from the three main framework layers: core, things and robotics. For the system example, we have inherited from the robotics layer. Three classes were created, *dkrobot*, *dkdrone* and *dkrover*. The last two inherit from the first (dkrobot).

The parent classes include general and specific methods than can be overwritten in children classes. The children classes (dkrobot, dkdrone and dkrover) use The ARCHADE and the Python library DroneKit [22]. DroneKit is a framework that facilitates interaction with an automatic pilot supporting Micro Air Vehicle Communication Protocol (MAVLink) [23] and it can be installed in different computing embedded cards [24], including the Raspberry Pi, which is used to simulate one of the UGVs in the tigers VS hunters' system.

The system is a SITL simulator because in each of the robot entities, we deployed the ArduPilot SITL [25]. The SITL software allows simulating the use of a real autopilot without having the actual hardware. This way the dkrobot, dkdrone and dkrover objects interact with the autopilot by using DroneKit.

In each of the entities, its agent interacts with the autopilot. The agents receive orders from the system controller agent running in the ground station, which is controlled by the pilot user. Examples of orders are *start autopilot*, *connect to vehicle*, *execute mission*, etc. However, the system controller and entity agents could act fully automatically if desired (i.e. without human intervention).

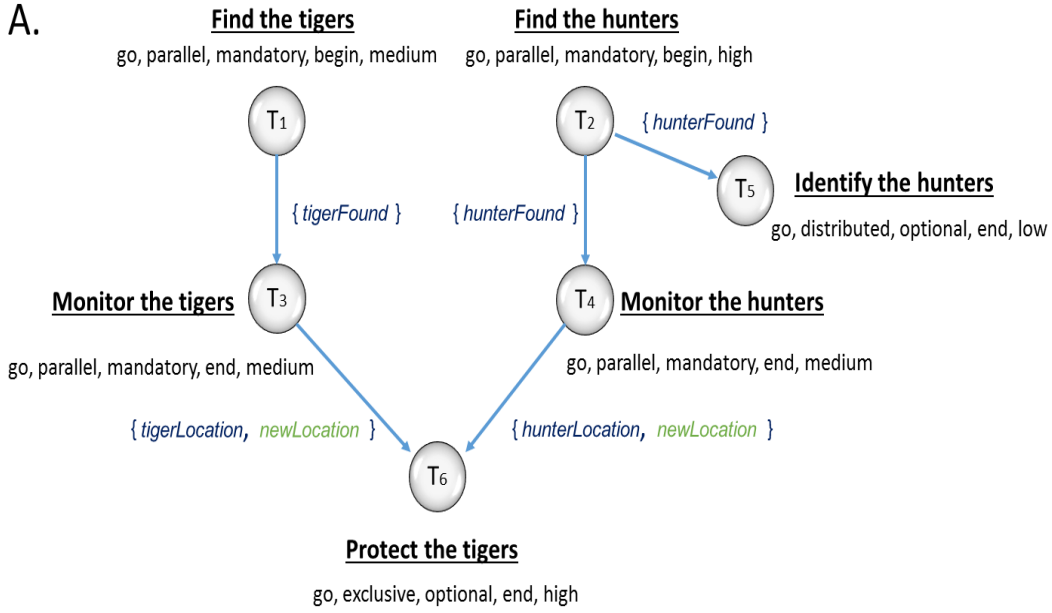
The HPC / HPRC system platform consists of a laptop, three virtual machines, a Raspberry Pi 3 model B and a HPC cluster of computers as described in table 3.

Table 3. Tigers VS Hunters system entities information. Notations: SC = System Controller, M = Master, S = Slave, PM = Physical Machine, VM = Virtual Machine, RPI 3B = Raspberry PI 3 Model B.

Entity	nodes	OS	Type	*Agent* & roles	CPU cores	RAM [GB]	ArduPilot vehicle	Devices
Demo	1	Ubuntu 14.04	PM - Laptop	SC, M, S	4	24	None	None
drone1	1	Ubuntu 16.04	VM	E, S	2	2	ArduCopter - dkdrone	Camera, thermal camera, GPU, LIDAR
drone2	1	Ubuntu 16.04	VM	E,S	2	2	ArduCopter - dkdrone	Camera, thermal camera, GPU, LIDAR
rover1	1	Ubuntu 16.04	VM	E, S	2	2	APMrover2 - dkrover	Camera, thermal camera, weaponry, LIDAR
rover2	1	Raspbian Stretch	PM - RPI 3B	E,S	4	1	APMrover2 - dkrover	Camera, thermal camera, weaponry, LIDAR
HPC cluster	7	Ubuntu 16.04	HPC cluster of computers	No agent, Interface	26	N/A	None	None

The devices in each entity are not actually real for the simulation but they are used by the matching service for task assignment. Moreover, figure 6 – A presents the mission tasks' network. Links between tasks represent event/action dependencies. After using the matching service, which is automatically done by the system controller agent, tasks are assigned as shown in figure 6 – B.

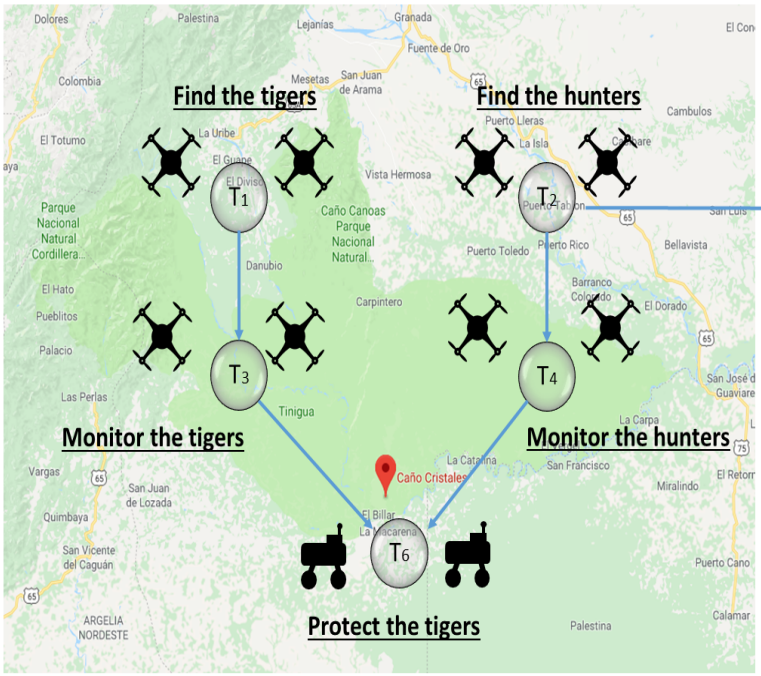
A.



Task Requirements

T1: Camera, GPU, thermal camera
T2: Camera, GPU, thermal camera
T3: Camera, GPU, thermal camera, LIDAR
T4: Camera, GPU, thermal camera, LIDAR
T5: HPC cluster of computers
T6: Camera, GPU, thermal camera, LIDAR, weaponry

B.



Task	Task software
Find the tigers	*findTigers*
Find the hunters	*findHunters*
Monitor the tigers	*monitorTigers*
Monitor the hunters	*monitorHunters*
Identity the hunters	*identifyHunters*
Protect the tigers	*protectTigers*

Area name: Caño Cristales, Meta, Colombia, **Latitude**: 2.264897, **Longitude**: -73.794225, **Altitude**: 276.90

Fig. 6. Tigers VS Hunters mission network. (**A**) Events / actions triggering forward tasks are contained within curved brackets. (**B**) Matched tasks after using the matching service. Map image taken from google maps.

The *matching* service checks the available resources and assign the tasks to the corresponding entities taking into consideration the tasks' classification (e.g. blocking, execution, etc.) Furthermore, the robotic entities are grouped as *air group* (UAVs) and *ground group* (UGVs). The environment area is equally distributed within the groups i.e. each of the UAVs in the air group and each of the UGVs in the ground group are assigned half of the area. For the simulation, a Colombian area known as *Caño Cristales* was chosen. It is colloquially known as *El río de siete colores*, the seven-color river. While there are not actual tigers in such location, we chose it to inspire the reader to visit the place and experience its beauty.

The process of defining a mission consists simply on filling up the mission template and once the agents are running, the mission execution is automatic. However, the mission software, i.e. the software for each task should make use of The ARCHADE live service to create events and actions.

Consequently, the framework and the middleware allow the creation of automatic ubiquitous supercomputing systems for multi-purpose missions that only require from the user, creating the system, to write the software for each of its designated tasks. Therefore, for the Tigers VS. Hunters system, the following tasks software were developed:

- *findTigers*: The software *findTigers* simulates image processing for tiger detection over the designated area. It uses the live service to create a *tigerFound* event in an entities' shared folder using the HPC file system layer. In a real software, i.e. use for the actual finding of tigers, the event can be coupled with the tigers' location data, which can be used by the next task (monitorTigers) to set its initial monitoring coordinates. For the purpose of the simulator, such locations are randomly fabricated.

- *findHunters*: The software *findHunters* simulates image processing for hunter detection over the designated area. It uses the live service to create a *hunterFound* event in an entities' shared folder using the HPC file system layer.

- *monitorTigers*: The software *monitorTigers* simulates the use of a LIDAR for the monitoring of the tigers found by the *findTigers* software. It uses the live service to create a *tigerLocation* event coupled with a *newLocation* action in an entities' shared folder using the HPC file system layer.

- *monitorHunters*: The software *monitorHunters* simulates the use of a LIDAR for the monitoring of the hunters found by the *findHunters* software. It uses the live service to create a *hunterLocation* event coupled with a *newLocation* action in an entities' shared folder using the HPC file system layer

- *protectTigers*: The software *protectTigers* reads events and actions from the software *monitorTigers* and *monitorHunters*, commands the motion of the rovers towards the tigers and calculates the distance to the hunters. If the distance approaches to a predefined value, the rovers are authorized to scare the hunters by using gas bombs. This is a simulated process and the *protectTigers* software is only triggered if the hunters and the tigers are found.

❖ *identifyHunters*: The software *identifyHunters* is executed by the ground station, which interfaces with a HPC cluster of computers, by executing MPI parallel software, which simulates the identification of the hunters.

Since the system is a simulator, the tasks' software does not actually do what would be necessary if the system were to be used for the real mission. However, the mission software does use the live service methods for events and actions creation, thus simulating the entire mission and it serves well as a proof of concept of The ARCHADE and the ubiquitous supercomputing ontology. Figure 7 shows the mission workflow.

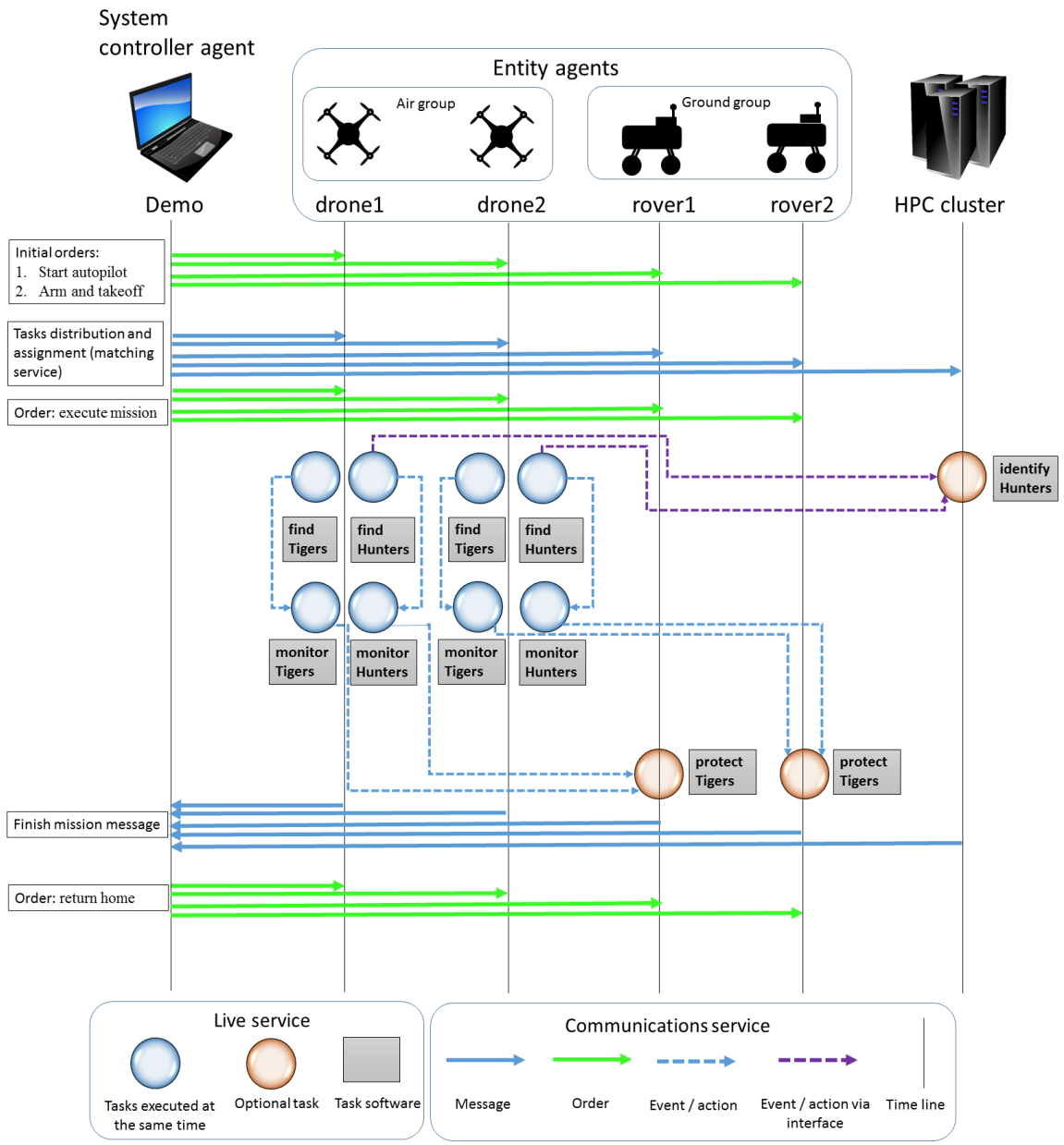


Fig. 7. Tigers VS Hunters mission workflow. There is corresponding software for each task. Events should be named in the same way that are inputted in the mission template. Correspondingly, the agents running in each entity constantly polls the shared file system to detect in-events in order to continue with its assigned tasks. The area is distributed equally between entities within a group. Consequently, events / actions are exchanged between entities in the same sub area in the case of protectTigers software. IdentifyHunters is a MPI parallel software.

Figure 8 shows a set of pictures of the implemented simulator.

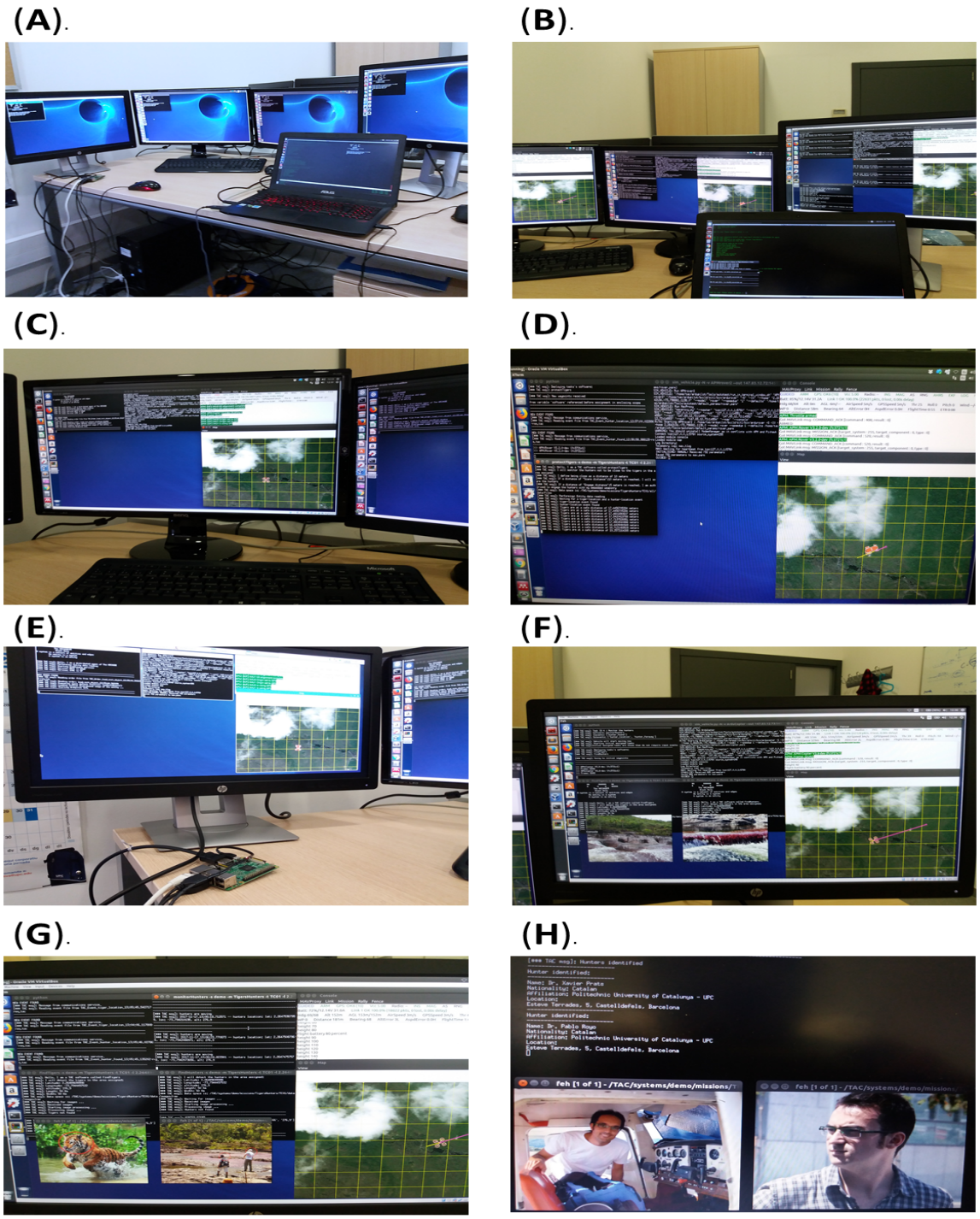


Fig. 8. Tigers VS Hunters simulator. (**A**) HPC / HPRC system platform with agents running in each entity (black boxes xterm in the left corner of the screen). The cluster is not shown in the picture (**B**) Running simulator. (**C**) Simulated drone. (**D**) Simulated rover. (**E**) Raspberry PI 3 model B used for simulated rover. (**F**) Simulated drone carrying out tasks *findTigers* and *findHunters*. (**G**) Simulated drone finishing *findTigers* and *findHunters* tasks and starting tasks *monitorTigers* and *monitorHunters* tasks. (**H**) Ground station finalizing task *identifyHunters*; the displayed pictures show the second and third author of this work. For clarification, they are not tigers' hunters in real life.

5. Conclusions

High Performance Computing (HPC) is a very useful tool for the solving of complex problems that humanity only dared dreaming before HPC appearance. Large corporations, vendors, engineers and scientists develop, improve and use HPC constantly worldwide. With several decades of existence, currently the only HPC competitor is the unstable Quantum Computing, which still need more years to be implementable and usable. However, even then, we can aspire encountering problems needing High Performance Quantum Computing.

While HPC can be found within several research and industrial endeavors, robotics has been reluctant to its utilization. Perhaps, because many robotic tasks do not require higher computing power or are ultimately executed on ground stations. However, with newly embedded computing cards, able to be installed with traditional operating systems, our imagination can run freely proposing highly automatic robotic systems performing tasks that would not be possible years ago.

Systems exhibiting ubiquitous supercomputing hold enormous potential, not only because they offer higher computing efficiency, as traditionally provided by HPC infrastructures and parallel libraries, but also because they bestow centralization/distribution, scalability and user transparency / cohesion as provided by using the same technologies and software used in traditional HPC for decades.

The ontology presented in this work can be used to build any kind of system, composed of any kind and quantity of heterogeneous entities and for any kind of mission. Furthermore, the current version of The ARCHADE facilitates the deployment, building and operation of such systems. While we focused on unmanned vehicles, The ARCHADE could be set upon any kind of robots or Internet of Things devices allowing the installation of Linux-based operating systems, opening possibilities for domotics, human augmentation, social robotics and many more applications. Within this frame of mind, in our previous work [26], we introduced the concept of *HPRC cluster of aircraft*, a set of civilian aircraft that, while flying in cruise phase, temporarily join to cooperatively execute its trajectories resulting into fuel savings as found by our work or to improve air-traffic flow management as we plan for future work. Both missions, fuel saving and air-traffic flow management are not computationally expensive but highly benefit from HPRC features such as scalability and user-transparency.

Ubiquitous supercomputing, as introduced in this work, relates to Edge computing, Fog computing and fits well within the Industry 4.0 revolution. Aiming at reducing latencies associated with the use of external computing infrastructures such as cloud computing and facilitating real-time systems, Edge computing proposes to use local / on-board computing power for specific type of applications while Fog computing proposes as well to use LAN connected devices. In this sense, HPRC could be understood as *HPC edge/fog computing*. However, ubiquitous supercomputing goes beyond by proposing integration between edge computing devices and centralized architectures. By using the ideas of a general-purpose computing mission, the user can decide or use the matching service to allocate specific tasks, either locally at the robots or in an external computing infrastructure, such as is the case of the identifyHunters task. While we use a HPC cluster for this task, computing resources over the cloud are an alternative as well. Furthermore, tasks type SIMD or MIMD (execution type) can make use of multiple nodes, embedded on different entities, given its

LAN connectivity. Nevertheless, for future work, we plan to explore a possible integration with the RoboEarth [27] platform, integrating The ARCHADE with cloud robotics as well. This flexibility is given by the ontology and more specifically by the interface concept.

The ARCHADE is currently is its infant stages and more middleware services are currently being developed. Such services allow a deeper interaction with the batch system software layer and provide failure tolerance and online task redistribution. Moreover, DroneKit classes are part of The ARCHADE and they allow using real vehicles, not only simulated as the ones in the Tigers VS Hunters system, by simply changing a field in the entities templates. This facilitates an easy transition from experimentation to real operation systems. In addition, advances in communications technologies, i.e. 5G [28] will provide eventually sufficient features, e.g. bandwidth, low latency, etc., for fully distributed performance-based High Performance Computing to be implementable with robotic units.

In [16], ROS was mentioned as the most suitable testbed nowadays. The most significant advantage of The ARCHADE over ROS is its default use of HPC technologies, which allows a system to exhibit high computing efficiency. However, we have used ROS for the HPC-ROS package as a part of the framework deployer service and furthermore the communications service from the middleware methods can be unplugged and instead use ROS or others solutions if desired, maintaining still the HPC features.

The ARCHADE goes further from existing testbeds, which focus mostly on the same type of entity (e.g. robots, WSNs, etc.), by allowing to integrate multi-robot systems with HPC traditional infrastructures. This is possible because of a change of mind of what a robot actually could be nowadays. Both in HPRC and in ubiquitous supercomputing for robotics, a robot is a general-purpose computing device and as such, it can be used within all kind of missions, purposes, etc.

The ARCHADE is a very flexible plug-and-play technology and we believe it to be consistent with a logical transition from single embedded computing units, such as current robots, into HPRC clusters interacting with traditional HPC clusters, computing-less devices and humans. Furthermore, the technology can be used to create either fully automatic, partial (user holding the highest hierarchy within the network system) or non-automatic systems (full control by the user).

The Tigers VS Hunters system was developed with the objective of showing The ARCHADE's general features and potential. While other systems could have been proposed, we used such system to incite the reader to imagine the possibilities that could emerge from the implementation of ubiquitous supercomputing and the use of The ARCHADE. However, a performance indicators demonstration, e.g. CPU load, RAM consumption, communications latencies, etc., is out of the scope of this work. Nevertheless, in a following work: *The ARCHADE: Ubiquitous Supercomputing for robotics. Part II: Experiments*, we will present results of using The ARCHADE for three different missions (simple motion, follow me and automatic WPA security cracking) both in simulated and real mode (i.e. using real robots). Moreover, we will show the results of a complete performance benchmark, both of The ARCHADE and the three missions' software.

In order to replicate the findings presented in this work, it is necessary to use The ARCHADE and the Tigers VS Hunters software. Additionally, for the system example, it

would be necessary deploying the HPC / HPRC setting depicted in section 4, installed with Ubuntu, Raspbian or Kali, as currently supported by The ARCHADE.

While the software is not open-source, we are interested in cooperation and we encourage the reader into contacting the corresponding author with the objective of establishing such cooperation, both at research and industrial level.

6. References

1. D. Walker, J. Dongarra. MPI: a standard message passing interface. *Supercomputer*. 12, 56–68 (1996).
2. M. Garland, S. Le Grand, J. Nickolls, J. Anderson, J. Hardwick, S. Morton, E. Phillips, Y. Zhang, V. Volkov. Parallel computing experiences with CUDA. *IEEE micro*. 28(4) (2008).
3. F. Cocchioni, E. Frontoni, G. Ippoliti, S. Longhi, A. Mancini, P. Zingaretti. Visual based landing for an unmanned quadrotor. *Journal of Intelligent & Robotic Systems*. 84(1), 511–528 (2016).
4. J. Dongwoon, K. Doo-Hyun, H. Young-Guk, T. Vladimir. Image processing acceleration for intelligent unmanned aerial vehicle on mobile GPU. *Soft Computing*. 20(5), 1713–1720 (2016).
5. O. Cetin, G. Yilmaz. Real-time autonomous UAV formation flight with collision and obstacle avoidance in unknown environment. *Journal of Intelligent & Robotic Systems*. 84(1), 415–433 (2016).
6. C. Kanellakis, G. Nikolakopoulos. Survey on computer vision for uavs: Current developments and trends. *Journal of Intelligent & Robotic Systems*. 87(1), 141–168 (2017).
7. E. Salami, J. Soler, R. Cuadrado, C. Barrado, E. Pastor. Virtualizing supercomputation on-board UAS. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*. 40(7), 1291 (2015).
8. NVIDIA Jetson solutions for Drones and UAVs. https://www.nvidia.com/en-us/autonomous-machines/uavs-drones-technology/. Online. Accessed December-2017.
9. C. Gonçalves, M. Dutra, A. Rabelo, F. Oliveira, A. Barbosa, L. Frinhani, D. Porto, R. Milanez. A robotic flying crane controlled by an embedded computer cluster. *2015 12th Latin American Robotics Symposium and 2015 3rd Brazilian Symposium on Robotics (LARS-SBR)*. 91–96 (2015).
10. O. Holland, J. Woods, R. De Nardi, A. Clark. Beyond swarm intelligence: the ultraswarm. *IEEE Swarm Intelligence Symposium*. 217–224 (2005).
11. A. Marjovi, S. Choobdar, L. Marques. Robotic clusters: Multi-robot systems as computer clusters: A topological map merging demonstration. *Robotics and Autonomous Systems*. 60(9), 1191–1204 (2012).
12. L. Camargo-Forero, P. Royo, X. Prats. Towards High Performance Robotic Computing. *Robotics and Autonomous Systems*. 107, 167-181 (2018).
13. I. Foster, S. Tuecke. Enabling technologies for web-based ubiquitous supercomputing. *Proceedings of 5th IEEE International Symposium on High Performance Distributed Computing*. 112-119 (1996).
14. S. L. Chu, C. C. Hsiao. OpenCL: Make ubiquitous supercomputing possible. *Twelfth IEEE International Conference on High Performance Computing and Communications*. 556-561 (2010).

15. L. Camargo-Forero, P. Royo, X. Prats, in "[On-board high-performance computing for multi-robot aerial systems]" Aerial Robots - Aerodynamics, Control and Applications, O. D. Lopez-Mejia, Ed. (InTech, 2017), chap. 7.
16. A. Jiménez-González, J.R. Martinez-de Dios, A. Ollero. Testbeds for ubiquitous robotics: A survey. *Robotics and Autonomous Systems*. 61(12), 1487-1501 (2013).
17. M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng. ROS: an open-source Robot Operating System. *ICRA workshop on open source software*. 3, 5 (2009).
18. G. Pardo-Castellote, D. D. Chairman. OMG data distribution service: Real-time publish/subscribe becomes a standard. *RTC Magazine*. 14 (2005).
19. E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, T. S. Woodall, in "[Open mpi: Goals, concept, and design of a next generation mpi implementation]" European Parallel Virtual Machine/Message Passing Interface Users Group Meeting (Springer, 2004), 97–104.
20. M. J. Flynn. Some computer organizations and their effectiveness. *IEEE transactions on computers*. 100(9), 948-960 (1972).
21. HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers. www.netlib.org/benchmark/hpl/. Online. Accessed December-2017.
22. DroneKit by 3D Robotics. http://dronekit.io/. Online. Accessed December-2017.
23. MAVLink Micro Air Vehicle Communication Protocol. QGroundControl. http://qgroundcontrol.org/mavlink/start. Online. Accessed December-2017.
24. DroneKit companion computers. http://python.dronekit.io/develop/companion-computers.html. Online. Accessed December-2017.
25. ArduPilot Software In The Loop - SITL. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html. Online. Accessed December-2017.
26. L. Camargo-Forero, P. Royo, X. Prats. High Performance Robotic Computing as an enabler for cooperative flights. *Proceedings of 37th AIAA/IEEE Digital Avionics Systems Conference*. 869 – 878 (2018).
27. M. Waibel, M. Beetz, J. Civera, R. D 'Andrea, J. Elfring, D. Gálvez-López, K. Häussermann, R. Janssen, J.M.M. Montiel, A. Perzylo, B. Schießle, M. Tenorth, O. Zweigle, R. Van De Molengraft. RoboEarth. *IEEE Robotics and Automation Magazine*. 18(2), 69 - 82 (2011).
28. International Telecommunications Union (ITU). Minimum requirements related to technical performance for IMT-2020 radio interface(s). www.itu.int/md/R15-SG05-C-0040/en. Online. Accessed February-2018.

Acknowledgments

The authors would like to thank the Colombian government, in particular to the Colombian Administrative Department of Science, Technology and Innovation Colciencias for the funding of the Ph. D studies of the first author, which include the work presented hereby.