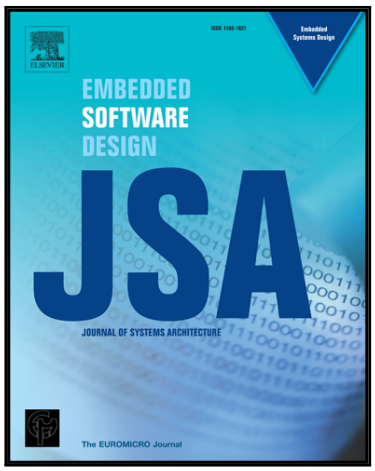


Kassandra: A Framework for Distributed Simulation of Heterogeneous Cooperating Objects

Richard Figura, Chia-Yen Shih, Matteo Ceriotti, Songwei Fu, Falk Brockmann, Héctor Nebot, Francisco Alarcón, Andrea Kropp, Konstantin Kondak, Marc Schwarzbach, Antidio Viguria Jiménez, Margarita Mulero-Pázmány, Gianluca Dini, Jesús Capitán, Pedro José Marrón

PII: S1383-7621(16)30234-X
DOI: 10.1016/j.sysarc.2016.11.012
Reference: SYSARC 1399

To appear in: *Journal of Systems Architecture*

Received date: 7 May 2016
Revised date: 21 November 2016
Accepted date: 30 November 2016

Please cite this article as: Richard Figura, Chia-Yen Shih, Matteo Ceriotti, Songwei Fu, Falk Brockmann, Héctor Nebot, Francisco Alarcón, Andrea Kropp, Konstantin Kondak, Marc Schwarzbach, Antidio Viguria Jiménez, Margarita Mulero-Pázmány, Gianluca Dini, Jesús Capitán, Pedro José Marrón, Kassandra: A Framework for Distributed Simulation of Heterogeneous Cooperating Objects, *Journal of Systems Architecture* (2016), doi: 10.1016/j.sysarc.2016.11.012

HIGHLIGHTS

- The distributed simulation of heterogeneous cooperating objects is addressed.
- The proposed framework allows the interaction between real and simulated hardware.
- Parts of the application logic can be simulated during the debugging of live systems.
- The framework allows to plan and evaluate system changes before deploying them.

KASSANDRA: A Framework for Distributed Simulation of Heterogeneous Cooperating Objects

Richard Figura*, Chia-Yen Shih*, Matteo Ceriotti*, Songwei Fu*, Falk Brockmann*, Héctor Nebot†, Francisco Alarcón‡, Andrea Kropp§ Konstantin Kondak¶, Marc Schwarzbach¶ Antidio Viguria Jiménez‡, Margarita Mulero-Pázmány‖, Gianluca Dini**, Jesús Capitán††, Pedro José Marrón*
*Networked Embedded Systems Group, Faculty of Business Administration & Economics, Dept. of Computer Science & Business Information Systems (ICB), University of Duisburg-Essen, Schützenbahn 70, 45117, Essen, Germany
richard.figura,chia-yen.shih,matteo.ceriotti,songwei.fu,falk.brockmann,pjmarron@uni-due.de
†ETRA Research and Development, Tres Forques, no. 147 46014 Valencia
hnebot.etraid@grupoetra.com
‡Center for Advanced Aerospace Technologies (CATEC), C/Wilbur y Orville Wright 17-19-21 41309 La Rinconada, Seville, Spain
falarcon,aviguria@catec.aero
§Leonardo - Finmeccanica Spa, Piazza Monte Grappa n. 4 - 00195 Roma, Italy
andrea.kropp@finmeccanica.com
¶German Aerospace Center (DLR), Institute of Robotics and Mechatronics Robotics and Mechatronics Center (RMC), Oberpfaffenhofen 82234, Wessling, Germany
konstantin.kondak,marc.schwarzbach@dlr.de
‖Estación Biológica de Doñana CSIC c/ Americo Vespucio, s/n, 41092 Sevilla, Spain; 2- 2 Department of Natural Sciences, Universidad Tcnica Particular de Loja, San Cayetano Alto, Loja, Ecuador
muleromara@hotmail.com
**Department of Ingegneria dell'Informazione, University of Pisa, Via Girolamo Caruso, 16, 56122, Italy
gianluca.dini@ing.unipi.it
††Departamento de Ingeniera de Sistemas y Automática Escuela Superior de Ingeniera. Universidad de Sevilla Camino de los Descubrimientos, s/n, 41.092 Sevilla (Spain).
jcapitan@us.es

Abstract—To address the heterogeneity and scalability issues of simulating Cooperating Objects (COs) systems, we propose KASSANDRA, a conceptual framework for enabling distributed COs simulation by integrating existing simulation tools. Moreover, KASSANDRA exploits the communication middleware used by real-world COs as underlying communication mechanism for integrating KASSANDRA-enabled simulation tools. In this way, real-world COs can be included with simulated objects in a seamless way to perform more accurate system performance evaluation. Moreover, such a hardware-in-the-loop approach is not limited to pre-deployment performance analysis, and can offer possibilities to analyse performance at different phases of CO applications. The concept of KASSANDRA has been carried out in the EU PLANET project. In this paper, we introduce the KASSANDRA framework components and show their interactions at different phases for node deployments in PLANET use cases. The result demonstrates the applicability of KASSANDRA to facilitate the development of CO applications.

Index Terms—heterogeneous simulation, distributed simulation, hardware in the loop simulation, live debugging of distributed systems, cooperating objects

While working on the paper, Marc Schwarzbach was associated with the German Aerospace Center (DLR), marc.schwarzbach@dlr.de. Nowadays, he is working for Autel Europe GmbH, Friedrichshafener Str. 2, 82205 Gilching, Germany, marc@autel.com

1 INTRODUCTION

Orchestrating heterogeneous Cooperating Objects (COs) can enrich application scenarios by exploiting different characteristics specific to each type of COs, e.g., wireless sensors, Unmanned Ground- and Aerial Vehicles (UGVs and UAVs) [1]. Stationary Wireless Sensor Networks (WSNs) are typically used for providing physical parameter measurements in a monitored area [2] . UGVs can complement

a static WSN with their mobility to act as mobile sensing element, allowing to have precise measurements at various locations [3], to collect data from a stationary WSN [4] or to maintain an existing deployment [5]. Rotary- or fixed wing UAVs are not limited by the topography of the area and can have similar mobility capability regarding surveying wide areas [6], data collection [7] and node deployment [8], [9], but with different limitations on accuracy, data collection bandwidth and operating ranges.

While clear advantages can be foreseen with Cooperating Objects, developing such CO systems is not a trivial task, specially if the CO deployment environment is hostile or unattended [10]. The system complexity and deployment cost can be dramatically unaffordable with increasing heterogeneity and scalability. Thus, performance evaluation of system design and implementation is required throughout the lifecycle of CO applications. Simulation offers a simple and pragmatic solution for pre-deployment system analysis. However, traditional simulator tools are often tailored to specific systems and thus do not match the requirements for heterogeneous COs Networks. It is not possible and infeasible to develop a monotheistic simulation environment that comprises all necessary models to meet different types of simulation requirements. Moreover, it is extremely complicate to evaluate the efficiency of deployment process and deployed CO application performance in a simulated environment in contrast to dynamic environment situations. Furthermore, such performance analysis is performed typically prior to the actual deployment. However, with vulnerable embedded COs and dynamic deployment environments, performance analysis can be required during the deployment or post-deployment phase in order to flexibly reconfigure the CO deployment to be resilient from dynamic changes.

To address these issues, we propose a conceptual framework, KASSANDRA for enabling distributed heterogeneous CO simulations by integrating different simulators for their specific COs. The integration of these KASSANDRA-enabled simulators is achieved by using the very same communication middleware that is used by the real-world COs. The simulators are modified to use the actual communication mechanism, e.g., the same application messages. Therefore, the real COs can interact with the simulated COs to evaluate the system performance during different phases of the application lifecyle. Such a hardware-in-the-loop distributed simulation environment, with operating COs in a real-world environment, can greatly increase accuracy of system performance evaluation. Moreover, this can significantly reduce the time and cost for CO deployments. In summary, our contribution is three-fold: First, the KASSANDRA framework outlines a flexible mechanism to integrate existing simulators for imitating interactions among heterogeneous COs. The simulation capability of KASSANDRA can be easily extended with a newly integrated simulator to meet more simulation requirements on application performance analysis. Second, the integration of the simulators is achieved by exploiting the communication middleware used by the real-world COs. This enables hardware-in-the-loop simulation for more accurate performance analysis during different phases of CO deployments. Finally, KASSANDRA defines a set of simulation-specific modules, such as simulation control and simulation logging, which ensure a controlled simulation environment for all simulated COs.

The design of KASSANDRA was motivated by the EU project PLANET, a framework "to create an integrated platform that supports the efficient deployment, maintenance and operation of large-scale deployments of heterogeneous Cooperating Objects" [11]. A prototype implementation of KASSANDRA has been carried out as a proof of concept with the deployment framework developed in PLANET. Particularity, the PLANET communication middleware is used to achieve distributed simulation with real- and simulated COs. With the PLANET application scenarios, we show that the KASSANDRA approach is applicable in different phases of the CO application lifecycle.

The remainder of the paper is organised as follows. Section 2 discusses existing simulation tools related to KASSANDRA for simulation of various COs. Section 3 presents an overview of the KASSANDRA framework and describes how KASSANDRA components work in the lifecycle of CO applications; a prototype implementation of KASSANDRA is detailed in Section 4; Section 5 demonstrates the applicability of KASSANDRA in PLANET application use cases; finally we conclude our work in Section 6 .

2 Related Work

Simulation has been widely performed to imitate the physical characteristics of COs or the operations of COs systems, and to estimate the approximated performance of real-world CO systems. Numerous simulation tools have been developed to serve various simulation purposes for different levels of system components. We focus in this section on the related work regarding the simulation tools for COs including wireless sensors and UAVs.

For simulating networks and protocols, ns2 [12] and OMNeT++ [13] are widely accepted simulation tools that can be adopted for testing WSN applications. Castalia [14], which is based on OMNeT++, can be used to test node behaviour, algorithms or networking protocols in realistic wireless channels with different radio models. The common issue with these tools is that the native application cannot directly be deployed without prior modifications like NesCT [15]. Moreover, the system and environment models are typically generic, lacking the credibility for simulating WSNs in dynamic environments.

The above limitations motivate the development of WSN-specific simulators. TOSSIM [16], Avrora [17] and COOJA [18] are well-known simulation tools for WSNs. TOSSIM allows to run native TinyOS [19] applications by emulating the operating system behaviour for each node. Avrora provides cycle-accurate simulation by emulating the hardware of the node and thus can run the code image for a specific sensor node platform. COOJA enables the simulation of networks at different abstraction levels and thus allows to trade off speed for accuracy. These solutions perform simulations on one single machine, which may incur a scalability issue as the number of simulated nodes increases. To address this, DiSenS [20] introduces a distributed architecture to simulate WSN applications on several hosts. Another evaluation framework, WISEBED [21], allows to run simulated and emulated nodes from several testbeds

in a shared environment, and thus enables hardware-in-the-loop simulation for WSNs.

While the aforementioned simulators offer useful environments for developing WSN applications, they do not support simulating the interaction between sensor nodes and external vehicles. Even though some tools can be augmented with mobility models, these models merely simulate the moving patterns instead of the physical behaviour of real vehicles. Thus, they are not fully applicable for supporting practical and effective evaluation of COs applications.

For simulating vehicle physics, simulink [22], integrated with MATLAB, was developed to model and simulate dynamic systems. It has been widely used in automatic control and digital signal processing for a broad range of autonomous aerial- and ground vehicles [23], [24]. Regarding autonomous aerial vehicles (UAVs), there exist many Flight Dynamics Models (FDMs) which allow to estimate the physical forces (e.g., thrust, lift and drag) on simulated aircraft. JSBSim [25] and YASim can be used in many higher-level simulation engines. Both differ in the kind of parameters required to produce a realistic outcome. While JSBSim requires parameters about the aircraft itself (e.g., derived from a wind tunnel test), YASim focuses on the physical characteristics of the aircraft (e.g., wings, engines) to simulates the airflow on the different parts of the plane.

FlightGear [26] and Openeagles [27] are sophisticated simulation environments that target realistic aircraft simulation. They allow simulating a large-size personal aircraft and offer a huge set of real planes and airfield models. They are also employed as training tools for pilots. Both support hardware-in-the-loop as well as distributed simulation with each aircraft being simulated on a different PC host. Both environments allow integrating JSBSim and YaSim as FDMs.

Other solutions, such as Stage Gazebo [28] and V-Rep [29] in the robotics field, support simulating the interactions of different vehicles among themselves and with their environment. Program debugging is available by using on-board sensor inputs like GPS, cameras, 2D/3D laser scanners. While Stage supports UGVs in a 2D environment, Gazebo and V-Rep support also UAVs in 3D environments. The Robotic Operating System (ROS) [30] enables the interaction between different devices and an application orchestrating their behaviour. However, it is only specific to robotic platforms, which limits the scope of applicability.

Similarly to WSN simulators, these tools do not support simulating the interaction with the external wireless sensors. That is, it is impossible to process data coming from an external sensor network and to physically interact with single sensor devices for node deployment or data collection. KASSANDRA is developed to bridge this gap and to enable such CO interaction. This is achieved by integrating existing simulation tools in a simple and yet effective way. Furthermore, KASSANDRA emphasizes hardware-in-the-loop, allowing more simulation scenarios to assist CO system development at different phases.

3 APPROACH

While employing heterogeneous Cooperating Objects (COs) has clear advantages, it makes deployment planning and debugging an extremely difficult task. KASSANDRA aims to outline a framework for enabling distributed simulations for Cooperating Objects (COs) Systems. Moreover, to estimate more realistic application performance, KASSANDRA adopts the concept of hardware-in-the-loop for interacting with real COs at different phases of the application development lifecycle. We first elaborate the KASSANDRA framework and then describe its role in the lifecycle of COs applications.

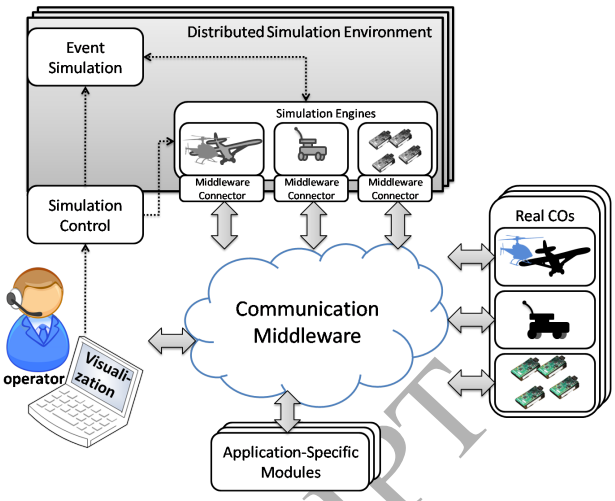


Fig. 1: The Conceptual framework of KASSANDRA with Real- and Simulated COs.

3.1 KASSANDRA Framework

KASSANDRA introduces a framework architecture to achieve distributed COs simulations with real hardware-in-the-loop. The framework specifies the integration of existing simulators and the communication mechanism required for the interaction between real- and simulated COs. An overview of KASSANDRA is illustrated in Figure 1. For integrating the simulators, KASSANDRA employs the very same communication middleware used for the interactions among real devices. In that way, further simulators can be seamlessly incorporated like their real counterparts, as long as the corresponding simulation models guarantee a real-time execution. As a result, it becomes possible to concurrently execute multiple simulated and real sub-systems if their interactions happen exclusively through the middleware.

In order to enable the analysis of system behaviours in a controlled environment, KASSANDRA defines simulation contexts for the different types of Cooperating Objects (the gray area in Figure 1). In addition, it can link the simulation contexts with the real-world COs, allowing the realistic evaluation of possible changes to an operational system before deploying them, thus saving time and resources. To address these, the KASSANDRA framework specifies several components including: *Communication Middleware*, *Simulation Engines*, *Middleware Connectors*, *Application Modules*, *Simulation Control* and *Event-specific Modules*.

For a typical CO application, KASSANDRA assumes that heterogeneous COs are interacting with each other via a *Communication Middleware*. To integrate a normal simulator (represented by a *Simulation Engine*) for distributed CO simulation, KASSANDRA requires the simulator to implement a Middleware Connector module. This module adopts the

same communication protocols of their real counterparts, in order to enable simulated COs interacting with real ones. This module also specifies application-specific messages used by the real COs for carrying on application tasks and thus enforces the simulator to simulate the COs with the same messaging scheme. Therefore, no adaptation is required for the real components for hardware-in-the-loop simulation. Furthermore, in this way, the application logic (represented as an *Application Module*) can be applied to both real- and simulated COs depending on the specific needs.

All the simulation engines are controlled by the *Simulation Control* component. It specifies the management registration procedure and the control commands (e.g, start/stop of the simulation or the simulator configuration) for the user to interact with the KASSANDRA-enabled simulation engines. This provides higher flexibility and more accurate control over the simulation sequence. This module also specifies logging or replay mechanisms for tracking the progress of the simulation. While being controlled centrally, the interaction is realised through the communication middleware, allowing a direct exchange of messages between all components during a simulation.

To simulate environment properties or sensing events, KASSANDRA also defines *Event-specific Modules*, which specify the interaction between the COs and a phenomena.

The proposed design of KASSANDRA has the following three advantages: First, it provides the *flexibility* to support a wide range of applications by orchestrating simulated modules together with real COs. Second, its *modular* design simplifies the extension with further simulation engines. Third, it simplifies the analysis of the system with real *software, hardware and humans in the loop*.

3.2 KASSANDRA in the Lifecycle of a Heterogeneous Network Deployment

The development and deployment of CO applications normally undergoes the following phases: In the *pre-deployment phase*, user requirement specification and deployment planning are the main tasks; in the *deployment phase*, the actual deployment operation is performed; finally in the *post-deployment phase*, the deployed CO network is monitored and recovery actions are required in response to CO or network failures. For these phases, different CO simulations are required to meet different evaluation objectives. We describe below how KASSANDRA can serve different simulation purposes in the lifecyle of CO applications.

The vision of KASSANDRA is to enable the analysis of the target system (or parts of it) in a controlled environment where the behaviour of the different Cooperating Objects can be investigated. During the *pre-deployment phase*, KASSANDRA allows to test alternative CO candidate configurations prior to deployment and to identify the COs satisfying the application requirements. This is achieved by orchestrating an arbitrary set of simulated COs with the real application logic. In this phase, potential simulated WSN deployments will be tested, together with the simulated autonomous vehicles as mobile data collectors, or mobile sinks. In order to increase the simulation fidelity, hardware-in-the-loop simulation of single COs can also be considered in this phase. During the actual *deployment phase*, KASSANDRA can estimate the energy consumption involved in the process and evaluate the most effective deployment procedure also with respect to possible restrictions regarding the use of unmanned vehicles. Simulated and real vehicles are used as deployment tools for a simulated sensor network, to evaluate the deployment procedure itself. In this phase KASSANDRA-based CO subsystems can be integrated into a partly deployed network to estimate its performance with different resource configurations to achieve higher deployment efficiency. Finally, when the network is deployed, the KASSANDRA-based CO systems can be integrated into the network for two different purposes: First, it enables the validation of the monitored performance against the predicted one, identifying and evaluating maintenance actions. Second, it allows to evaluate potential reconfiguration plans, by adding additional resources in simulation.

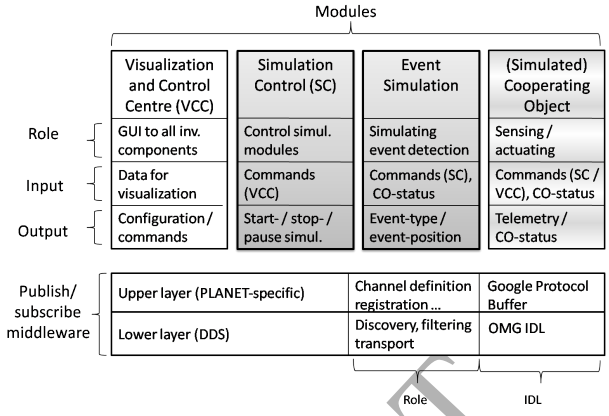


Fig. 2: KASSANDRA modules (top) and PLANET communication middleware (bottom). Modules in grey are specific to the simulation, while the other modules are used also in real-life scenarios.

4 IMPLEMENTATION

We have carried out a prototype of the KASSANDRA framework in the PLANET project. The implementation of KASSANDRA uses as communication medium the same communication middleware used in PLANET, which was developed for inter-connecting heterogeneous COs including wireless sensors and UAVs/UGVs.

Figure 2 summarizes the KASSANDRA modules and their interactions through the PLANET middleware. Modules in grey are specific to the simulation environment, while the others are used also in real-life scenarios. In KASSANDRA, each individual CO can be instantiated as either simulated components, real components or various combinations of both. We elaborate the used PLANET middleware, the implemented KASSANDRA modules and their interactions in the following subsection.

4.1 Communication Middleware

The main purpose of the communication middleware in a real scenario is to enable all involved devices to exchange information and synchronize on common tasks. In the context of KASSANDRA, the same middleware also integrates the simulation-specific modules into the network of real-life

devices, in order to allow real and simulated devices to co-operate with each other. The following paragraph describes the communication middleware employed in the PLANET project, which was used also for the implementation of KASSANDRA.

In PLANET, a publish/subscribe middleware was developed to enable the inter-communication between the different sub-systems and Cooperating Objects , implemented in C++, C# and Java. Typical data exchanged includes gathered sensing data provided by the COs or delivered CO status information. Additional commands are used to manage COs based on the application logic or the PLANET CO control policies.

The implementation of the middleware follows a two layered approach (as shown in the bottom part of Figure 2). The upper layer defines the CO registration routine, the PLANET messaging scheme and the default PLANET message channels, which could be extended on demand. Default message channels were defined to separate between different kinds of messages, e.g., for registration, commands and status information. On top of this, a further channel for the synchronization of the specific simulation modules was defined for KASSANDRA. The platform already included a set of pre-defined message types, e.g., for sending telemetry information. Further customized messages can be defined in Google Protocol Buffer format. The registration was implemented through a PLANET component to provide unique IDs to each participating device. During the registration, a module provides information about its type, status and capabilities. The user could request the list of active modules to plan operations involving any combination of real and simulated devices.

The lower communication layer provides a set of communication APIs for underlying communication protocols. This layer is especially responsible for discovery as well as message filtering and transportation. Currently, the middleware is built on the Data Distributed Service (DDS) implemented by RTI [31], but does not depend on it. DDS is an open standard, defined by the Object Management Group (OMG) [32], it provides efficient discovery, message filtering and transportation. It is highly customisable though a variety of QoS parameters. The RTI implementations is available for many platforms and languages, such as C, C++, C#, Java. Commercial alternatives to DDS are the Advanced Message Queuing Protocol (AMQP) [33], Java Message Service (JMS) [34] and Message Queue Telemetry Transport (MQTT); instead, related research projects are, e.g., Amigo [35], RUNES [36] and iLand [37]. While all of these middleware solutions fulfil the basic requirements from PLANET for communication, the choice of DDS based on two reasons: 1) the RTI implementation of DDS is available for all required programming languages (C++, C# and Java), while other solutions like Amigo, RUNES, iLand and JMS do not provide implementations in all these languages. 2) DDS provides the richest set of configuration and thus was the most flexible option for PLANET. For example AMQP and its implementations like RabbitMQ always utilize TCP/IP and thus does not offer best effort communication. MQTT as a standard defined by OASIS, is an appropriate choice for IoT but it requires a central component (Broker), unlike DDS that acts in a fully distributed fashion. Furthermore, using DDS as lower layer enabled the PLANET middleware to efficiently filter out messages, through DDS topics and domains.

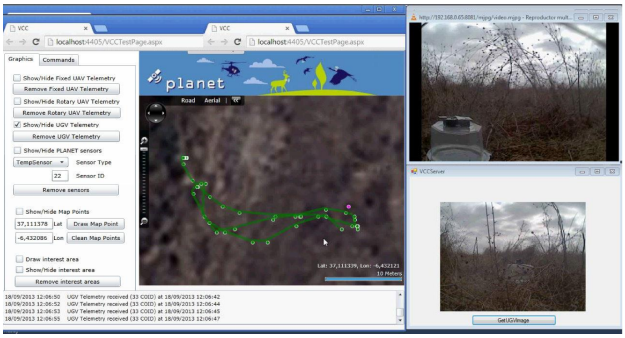


Fig. 3: Visualisation and Command Centre (VCC), used to control the waypoints for a UGV and for displaying images from the onboard camera.

4.2 Real COs, Application Modules and Visualizer

In addition to the PLANET middleware, other PLANET components are used as non-simulation specific KASSANDRA modules, including real COs, application modules and the visualization tool (Visualisation and Command Centre, VCC). The VCC acts as the user interface and it allows to configure the other components and to view the status of a simulated or real operation. The COs are controlled by the VCC to fulfil a common task. If included into a simulation, the real COs behave in the exact same way as in real life. For these modules, the interaction is independent of whether they are utilised in a real or a simulated scenario.

In PLANET, three types of CO platforms are employed to support application scenarios: sensor nodes, rotary/fixed-wing UAVs and UGVs. The sensor nodes are the Prospeckz-5 board based on the EFM32 microcontroller with the NRF24L01 radio module, and they are mainly built for pollution detection and animal monitoring. UAVs (Viewers, X-Vision, etc.) and UGVs (Jaguar-4x4-wheel Mobile Robotic Platform) are used as deployment tools or mobile data collection stations for sensor networks.

Vehicle integration. Instead of controlling the vehicles directly through the PLANET middleware, each vehicle was controlled by a Ground Control Station (GCS) that accepted commands through the PLANET middleware and forwarded them to the autopilot of the vehicle. The main reason for having GCSs was safety regulation, since the GCS allowed us to filter out commands that could be dangerous for the vehicle, its surrounding or living human beings close to the vehicle. Furthermore the GCS could initiate a safety routine, e.g., for landing, and drop all further incoming messages. Commands that can be sent to all vehicles are routeplans, that include a list of waypoints and actions to perform, like starting, landing, deploying a sensor or turning an integrated sensor on or off. The UGV also allows more direct control, like sending just a single command to specify the next location to go to. In return, the vehicles provide constant output about their status, e.g., the telemetry.

Sensor node integration. In PLANET, sensors are not directly communicating through the PLANET middleware, instead,

data provided by the sensors is collected centrally at a Data Collection Station (DCS), which acts as a gateway between the low power sensor network and the IP based PLANET network. The DCS was implemented in two versions, one implemented in C++ and another one in Java. Both implementations could either just store collected messages in a log file, or forward single or buffered messages received. The Java implementation could also store collected messages in a Derby database, allowing to request data on demand via the PLANET middleware. If combined with a GCS, a vehicle could be used as a mobile Data Collection Station. Commands sent to the DCS can change the reporting period or the powering of the USB ports to reset all sensor nodes directly attached to the DCS; they can also request historical data and the status of the DCS itself, like the number of buffered messages.

Visualisation and Control Centre. This PLANET component, shown in Figure 3, acts as the bridge between the user application and other PLANET components. In the context of KASSANDRA, it is specifically used to configure the simulation modules and to allow the user to control the simulation progress. It is implemented as a web application using the Silverlight technology. VCC offers a set of user interfaces through the web browser for application input and framework output. The most prominent part of the VCC is a map that can be used in two ways. First, it can display the location and status of COs. This includes the trajectory of vehicles as well as a description of the COs, their battery level, telemetry, or readings of sensors. Second, it allows to define waypoints and actions to be sent as commands to the vehicles. The user can interact with the map for changing its zoom level or its centre and to choose which information should be displayed. The left part of the VCC provides more control over the other components, e.g., for sending a landing command to a UAV or to enable or disable a sensor. In the context of KASSANDRA, the VCC provides a configuration dialogue to configure all involved components. The bottom part of the VCC shows logging information, which includes sent and received messages. More complex information like video streams are not integrated directly into the VCC, but can be opened in a separate window.

Several application scenarios have been defined and carried out in PLANET. We describe these application modules in Section 5.

4.3 Simulation-Specific Components

For simulation, we tried to utilise as much as possible real hardware and software components (upper part of Figure 2). This allowed us on the one hand to enable a more realistic simulation, while on the other hand it also allowed us to test the real hardware and software logic in simulation. All simulation-specific components have to be configured prior a simulation with the state of the component they simulate, like number of nodes to be simulated, initial position of a vehicle, maximum speed allowed. This configuration is done through a dedicated simulation channel. For the following communication, however, the simulated COs use the very same message channels as their real counterparts. Therefore, during operation, it is possible to interact with a simulated CO in the same way as with a real one.

Vehicle Simulation. Based on the PLANET middleware, the implementation of KASSANDRA integrates a variety of simulation engines. To simulate UAVs or UGVs physics, a common solution is to develop Matlab/Simulink models corresponding to the process controlling the vehicle as well as the vehicle itself. The former is responsible for the generation of the required signals for the onboard sensors and actuators. It receives information about the vehicle state as well as about the intended flight or driving plan. By computing the deviation between the current and the desired state, the commands for the actuators are generated following appropriate manoeuvres. These types of controllers are based on classical PID control methodologies and can either be modelled as Simulink blocks or directly executed in the same implementation used by the final system. The other component necessary to allow the simulation of the vehicle is a model of the platform itself as well as its response to actuation commands under different environmental conditions.

For the UAV simulation, the GCS and the vehicles autopilot software are the same as in a real life scenario. However, instead of using the actual vehicle peripherals (sensors and actuators), a Matlab-based model is used for simulating the physical behaviour of the vehicle and its interaction with the environment. This model also considers dynamics and errors of the different on-board sensors, which allows to verify the autopilot commands under various conditions. For simulating a UAV, the according GCS needs to both redirect the received commands to the corresponding model and provide the models output as status information through the communication middleware. Furthermore, using the real GCS also for simulation allows to seamlessly integrate various simulated vehicles into KASSANDRA similarly to real devices.

The UGV simulation has two modes: The first one offers a motion model for the mobility of the UGV. This motion model provides accurate information about the internal state of the vehicle during movement (momentum, spin, etc.). This model could be controlled in the same way, providing the same output, as a real UGV. The other mode allows to provide a trace of a former UGV run for a simulation.

WSN Simulation. Concerning wireless sensor nodes, COOJA is a flexible Java-based simulator that enables cross-level simulation at different levels of the system, at a low level through an instruction-level TI MSP430 emulator or at a higher level by executing Java programs. The result is the possibility to combine different abstraction levels in one single simulation, trading off simulation speed by executing Java nodes for accuracy by emulating deployable Contiki or TinyOS code. In addition, COOJA provides a large range of communication models to properly represent the properties of the wireless signal propagation in different environments. We integrate COOJA in KASSANDRA by extending it with the PLANET messaging schemes to interact with other components. In this way, COOJA basically replaces a single DCS, or several ones from a real deployment. In case of multiple separated networks, each one can be simulated in its own COOJA instance. We augmented COOJA with the status information of the autonomous vehicle to give the possibility to interact with single sensor nodes: modify a

node position, as in the case of a UAV carrying a wireless sensor node, acting as a mobile DCS, and to add or remove nodes from a deployment with a vehicle as deployment tool. This mobile DCS or deployment tool could either be a simulated or a real vehicle. In case of multiple simulated networks, a vehicle could interact with each of them. However, since the different WSN simulators do not share any internal state of the sensors, it is not possible to create direct wireless links between devices belonging to different simulations. In addition, an interface was added to manipulate the virtual environment of the simulation, e.g., if a Cooperating Object affects an event or a phenomenon to be sensed.

Simulation Control. In addition to the above simulators, the *Simulation Control* component is implemented to coordinate the execution of the different simulation engines. It also receives commands from VCC to control the simulation progress by starting, stopping, pausing a process or configuring it. In order to start a simulated sub-system, an appropriate configuration must be provided to define the initial state, e.g., number and position of Cooperating Objects and event timings.

A *Simulation Replay* component, embedded into the Simulation Control, is also able to replay recorded traces by injecting previously collected messages exchanged through the PLANET middleware, either in a real-life scenario or in a simulated one. By subscribing to the different communication channels of the middleware, it is possible to trace the content and timings of the exchanged messages between the (real or simulated) COs. This is essential to provide logging and support debugging of the system, as well as enable the replay of messages and events at a later point in time.

Event Simulation. The *Event Simulation* module defines the type, the position and the time of a phenomena within a simulation. It could either be used to trigger a reaction within the *WSN Simulation*, or to notify about the sensed event in a corresponding status message. This module also subscribes to the telemetry information of any simulated or real COs to create a bridge between the existence of a phenomena and the corresponding position of the vehicles (e.g., to simulate obstacle detection).

4.4 KASSANDRA Modules Interaction

The following subsection provides information on how the KASSANDRA modules interact with each other. The first part describes which kind of information is exchanged during the different stages of a simulation. The second part includes examples about how the PLANET Platform API is used by the KASSANDRA modules.

4.4.1 KASSANDRA *stages*

The interaction between the different modules is dependent on the actual scenario to be simulated. However, each simulation can be separated into the following four stages.

Registration. During the registration stage, all components (simulated or real) register at the PLANET Control through a dedicated channel by providing information about their type, their status and their capabilities. The actual registration message depends on the type of component. For instance, a UAV could register as rotary wing UAV with a payload of 3 additional wireless sensor devices. The information about all registered components is stored centrally.

Component	Configuration Phase (Received Data)	Runtime Phase (Sent Data)
Vehicles	Waypoints Actions Initial position (sim) Battery Level (sim) Deploy Load (sim) Max Speed (sim)	Position Timestamp Velocity Accelerometer Euler Angle Gyrometer
Sensors	Sampl. Period Sensor Count (sim) Sensor program (sim)	Sensor Readings
VCC	Acknowledgements	Start/Stop Sim. Sampl. Period (Sensors) Waypoints & Actions (Veh.)

TABLE 1: Example of interactions during the configuration and runtime stage for different COs and for the VCC.

Furthermore, each of them receives a unique id that can be later used for addressing.

Configuration. All components are configured through the VCC. The middle column in Table 1 provides a summary of the configuration messages exchanged between the different involved modules. In this stage, a real UAV can receive a user-defined route plan with waypoints and actions to perform. The simulation modules will be configured with status information, such as number of sensors, type of applied models, conditions for reporting events. An excerpt for a message used to configure a WSN simulator is shown in Listing 1. Other modules are configured through specific messages defined with Google Protocol Buffer. Furthermore, simulated modules can be configured to register to the status messages of other simulated or real modules. This allows, for example, to simulate a mobile DCS by sharing the status of a vehicle with the WSN simulator, or to make a phenomena dependent on a vehicles position. This stage ends when the VCC starts the simulation through the Simulation Control.

Runtime. During simulation, all simulation modules behave like their real counterpart. Additionally, the Simulation Control starts recording all exchanged messages. These collected messages, an example of which is reported in the right column of Table 1, can be used to either debug or replay a scenario. At any time, the user can reconfigure or control single simulated or real components, e.g., controlling a vehicle, changing the list of nodes to be simulated or starting a phenomena.

Analysis. As soon as the user decides, the simulation can be stopped. Data logged by the Simulation Control can be used to replay the simulation in the VCC. Furthermore, the data of single devices, e.g., the real-life trace of a UAV, can also be extracted and replayed into another simulation.

```
message TopConfigurationSimulator {
  optional string identifier = 1;
  optional int32 randomSeed = 2;
  ...
  optional RadioMedium medium = 8;
  repeated simulatedNode nodeList = 9;
}

message simulatedNode {
  optional int32 id = 1;
  optional string application = 2;
  optional Position position = 3;
  optional bool isMobile = 4;
  ...
```

```
}
```

Listing 1: Excerpt of a configuration example specific to a WSN simulator written in Google Protocol Buffer. Node specific configuration like application and type are given through additional embedded information (not reported in this example).

4.4.2 PLANET Platform API

The PLANET Platform provides an API to support KASSANDRA in each of the aforementioned stages. This API enables the modules to publish messages, subscribe to message-types and channels, and to register to the PLANET Control. The following provides a description of a basic set of functions used by KASSANDRA.

Publication of Messages. Listing 2 shows how the Java implementation of the PLANET Platform can be exploited to publish a message (e.g., the TopConfigurationSimulator data shown in Listing 1). It first obtains an instance of the PLANET Platform, which allows to create a channel by providing the corresponding id ("SIM_CHL", in this case). As soon as the message is created, the module can use the publish method of the channel to distribute it to the subscribed modules.

```
publishSimulatorConfiguration(){
  PlanetPlatform platform =
    PlanetPlatform.getInstance();
  PlanetChannel channel =
    platform.joinChannel(ChannelType.SIM_CHL);

  TopConfigurationSimulator simConf =
    createConfigurationMessage();
  channel.publish(simConf);
}
```

Listing 2: Example of publication of a simulator configuration.

Subscription. Listing 3 depicts the code necessary to subscribe to UAV telemetry information on the "CO_DATA_CHL" channel. The channel object is created similarly to the previous case. However, for a subscription, an additional handler is necessary to be notified at the reception of messages matching the subscription filter. This handler is initialised with the type of message it should receive. The initialised handler is the only parameter for the subscribe method of the channel object. The second part of the example shows how the handler can be created by extending the according PLANET class.

```
subscribeToTelemetryInformation(){
  PlanetPlatform platform =
    PlanetPlatform.getInstance();
  PlanetChannel returnChannel =
    platform.joinChannel(ChannelType.CO_DATA_CHL);

  COHandler handler =
    new COHandler(TopUAVTelemetryStatus.newBuilder());
  returnChannel.subscribe(handler);
}
...

//handler for telemetry
class COHandler extends PlanetMessageHandler {
  ...
  @Override
  public void handleMessage(GeneratedMessage arg0,
      PlatformHeader arg1) {
    TopUAVTelemetryStatus telemetry =
      (TopUAVTelemetryStatus) arg0;
    // do something with telemetry
  }
}
```

Listing 3: Example of a subscription to telemetry information.

Registration. Listing 4 shows how a module can register to the PLANET Control. After obtaining an instance of the PLANET Platform, for a registration the module might have to provide some specific configuration. This has to be put in the according Registration message (TopRegistrationInfo, in this case). Since the communication is asynchronous in a publish/subscribe system, it also has to specify a handler to be notified as soon as the PLANET Control has successfully integrated the according module into the PLANET system. To finalise the registration, the module has to call the registerCo method of the platform, providing the registration message and the handler as parameters. For the sake of brevity, the implementation of the handler is not shown in this example.

```
startCO(){
  PlanetPlatform platform =
    PlanetPlatform.getInstance();
  TopRegistrationInfo info = createRegistrationMsg();

  RegistrationHandler handler =
    new RegistrationHandler(
  TopDCSRegistrationConfig.newBuilder());
  platform.registerCo(info, handler);
}
```

Listing 4: Example of a module registration.

5 KASSANDRA IN PLANET

In this section, we first elaborate different modules that are included in KASSANDRA to support testing and validating CO operations in two PLANET application scenarios. We then discuss several implementation issues for simulating PLANET COs in KASSANDRA with the PLANET middleware. Additionally, we discuss the lessons learned during the realisation and use of KASSANDRA.

5.1 DDS-Configuration

The performance of the middleware highly depends on the settings of the underlying DDS configuration. While the performance of the simulator could be improved by using an optimised DDS-configuration, we relied on the very same configuration that was specified by the real PLANET applications.

For the experiments, we used RTI DDS Version 5.1.0.00 and the QoS configuration was given through an XML file. For the simulation in KASSANDRA, we used the parameter RELIABLE_RELIABILITY_QOS, which enforces a reliable communication, while still using UDP. The data length was increased to 30000 elements through the parameter "sequenceSize". This was necessary since some of the sensor transmitted visual information in one single message. Furthermore, to enable better debugging in simulation, we stored all messages through the KEEP_ALL_HISTORY option.

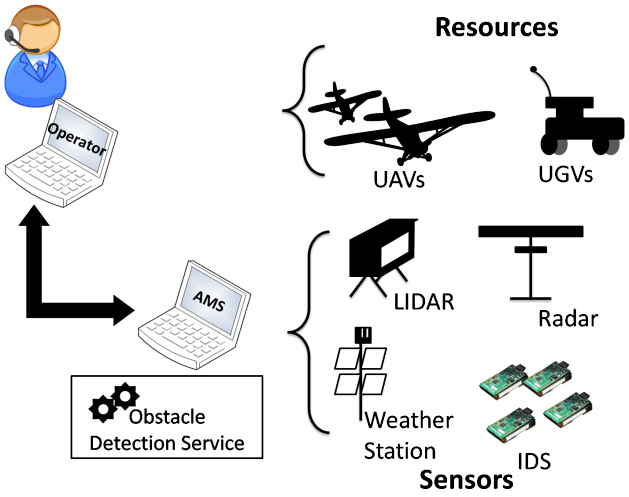


Fig. 4: An operator is remotely requesting for AMS resources (UAVs, UGVs). The AMS additionally manages a heterogeneous set of sensors and services: Obstacle Detection System (ODS) and Intrusion Detection System (IDS) services.

5.2 PLANET Application Scenarios

The PLANET framework supports efficient deployment, maintenance and operation of large scale distributed systems of heterogeneous COs. To prove its flexibility, the PLANET framework was tested in two fundamentally different scenarios: An automated airfield (AIR) and a biological reserve (DBR). In this section, we describe how the different simulation modules are integrated in KASSANDRA to carry out specific simulation scenarios.

5.2.1 AIR-Scenario: Automated Mission Service Provision

In the context of PLANET, an Airfield Management System (AMS) was developed to manage UAVs of an unattended airfield and to provide these UAVs to a remote operator.

However, in case of the absence of any human supervision, it is important that the AMS can automatically detect any incident that could endanger safe airfield operations, such as taxiing, takeoff or landing. Critical incidents can be an obstacles on the runway, or an unauthorized person in the airfield perimeter. For that reason, PLANET included a system for automated obstacle detection and removal as shown in Figure 4.

In PLANET, the final implementation for the automated airfield was tested in ATLAS (Air Traffic Laboratory for Advanced unmanned Systems) [38], a test flight centre, intended for the development of experimental flights with UAVs. However, the development process of the AMS logic required continuous testing and thorough evaluation to ensure safe operations. This development process could not be performed with real devices to avoid costly and potentially dangerous tests.

For this reason, a use case was simulated, in which the core system for the automatic airfield management was tested, together with obstacle detection and removal. In this use case, the AMS should stop any UAV operation as soon as the weather condition is considered to be bad, or if an obstacle is detected on a runway.

To simulate this use case, the real AMS logic was orchestrated together with a set of UAVs, following randomized operations, a single UGV for removing obstacles and two event-specific modules: A *Weather Station Module*, which reports information regarding the weather conditions through a predefined sequence of measurement values and an *Obstacle Detection System*, which is in charge of detecting any unexpected object in the runway that could endanger UAV operations. The second module allows to report the detection of an obstacle at random positions on the runway, at a predefined or random point in simulation time. The module was configured so that it stopped reporting an obstacle as soon as a UGV was nearby (to simulate the removal process of an obstacle). As soon as an obstacle was detected, the control logic for the UGV should command the UGV to remove the obstacle.

With these two additional event-specific modules KASSANDRA allowed us to: 1) stress the logic of the real AMS to check whether it could properly and timely react on different events (bad weather, detected obstacles), during normal airfield operations, guaranteeing safe UAV operations. 2) Test the UGV control logic, to find appropriate routes at different initial positions of the UGV and varying obstacle positions. The scenarios could be tested with accurate timings and provided valuable information for the design of the AMS, which allowed us to evaluate the effectiveness of the Obstacle Detection System, regarding the delay of removing an obstacle (including the routing of the UGV) and its impact on the UAV operations.

5.2.2 DBR-Scenario: Pollution Monitoring in Doñana

Doñana National Park is a world heritage site in the south west of Spain. The park features a great variety of ecosystems and shelters abundant wildlife, including endangered species such as the Iberian lynx (*Lynx pardinus*) or the Spanish imperial eagle (*Aquila adalberti*). This biodiversity hotspot is of special interest for biological research, utilizing more and more sensors and UAVs for environmental monitoring and animal tracking [39].

In contrast to the airfield scenario, it was important to ensure a minimal impact on the environment. For that reason, combustion engines were prohibited to operate in the national reserve. In result, a sensor deployment has to be planned carefully for the requirements of the specific use case, especially a long lifetime is necessary to ensure that no regular management operations are necessary. One of the major problems for planning and preparations for the PLANET-operations is that the access to the Doñana park is highly restricted and experiments inside of the park had to be performed in just a few days per year.

For this scenario, we simulated a use case in which a pollution monitoring system was tested. Pollution is one of the biggest threats to the preservation of a natural environment. A sever incident was the disaster due to the heavy metals coming from the dam breakage of the *Boliden Aznalcóllar* mine (April 25, 1998)) [40], which resulted in thousands of animals killed and severe pollution of this privileged natural landscape.

In case of the event of a pollution spill in the national park, it is important to know where the event started and

Fig. 5: Left: The VCC showing a (water) sensor network (green dots), with two nodes to be replaced (red dots). Right: The automatic deployment of an additional water sensor through a rotary wing UAV.

how it distributed in the environment. This allows to effectively concentrate counter measurements at the locations of the incident. Using wireless sensors for a pollution detection system is ideal, since they can organise autonomously and provide regular readings about the environment to the user. In PLANET, a WSN was envisioned as an early detection component for a pollution detection system. Sensors deployed in the marshes of Doñana provided regular readings about the pH-value of the water, the temperature and oxygen level, which serve as an indicator in case of a pollution incident. In this use case, the proper functionality of the network is of crucial importance. Single failing nodes might lead to undetected events, or to disconnected sub-networks and thus uncovered sub-areas. In PLANET, a WSN monitoring logic was checking the received WSN data for errors or missing nodes. In case of a detected node failure, a rotary wing UAV platform was used to deploy additional nodes, if a failing node endangered the functionality of the system [8].

To simulate this use case, the real monitoring logic was orchestrated together with a simulated rotary wing UAV and a WSN, running a simple data collection application. Node failures could either be triggered on demand, or a trace, collected from a real reference WSN could be used as input for the monitoring logic. As soon as a node failure was detected, this incident was reported to the VCC to allow the user to start an automated deployment procedure. A simulated rotary wing UAV could then deploy an additional note into the WSN simulator. The deployment procedure and final WSN performance was shown in the VCC.

In this use case KASSANDRA allowed, before the real deployment in Doñana, to evaluate the deployment procedure, timing and healing performance (especially energy consumption), when using a rotary wing UAV as deployment tool, as well as the VCC to show the status of any operation as depicted in Figure 5.

5.3 Discussion

During the implementation of Kassandra, we paid attention to simulation efficiency, which can be affected by the scalability of the different simulation modules. Another influential factor is the scalability of the underlying communication framework. We show in this section how we perform these scalability analysis regarding these two aspects.

5.3.1 *Simulation Scalability*

In PLANET, KASSANDRA integrates simulators that provides simulation capabilities for rotary wing UAVs, fixed

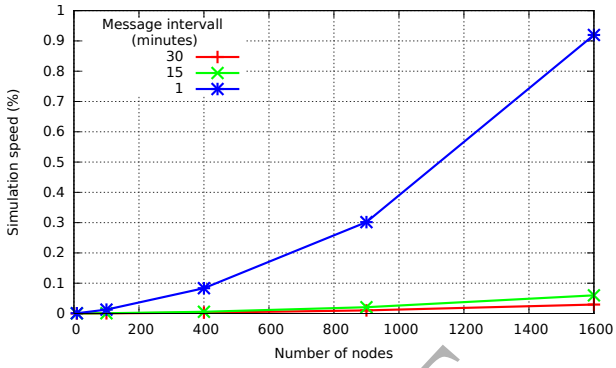


Fig. 6: Simulation speed for different sending intervals depending on the node count. Values at: 9, 100, 400, 900, and 1600 nodes.

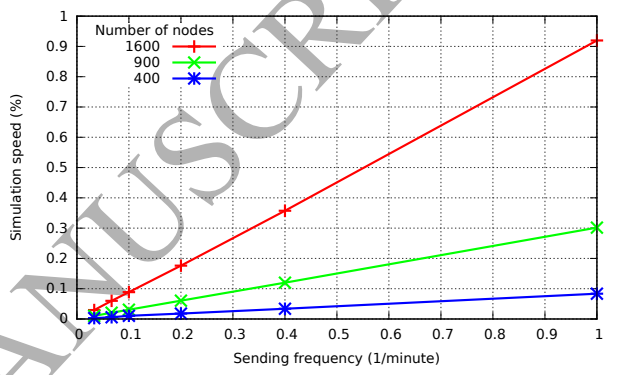


Fig. 7: Simulation speed for different number of nodes depending on the sending frequency. Values at: $\frac{1}{1}$, $\frac{1}{2.5}$, $\frac{1}{5}$, $\frac{1}{10}$, $\frac{1}{15}$, and $\frac{1}{30}$ 1/minutes.

wing UAVs, UGVs and WSNs. Since UAV and UGV simulation engines just simulate a single device, the scalability of a system with more vehicles can be easily supported by adding more processing resources for each simulation engine, e.g., one dedicated simulator for each isolated system. However, if single sensors are tightly connected with each other in a stand-alone network, adding more sensors requires stronger processing capabilities of the WSN simulation engine. For that reason, we performed evaluation analysis on the WSN simulator (COOJA) in terms of number of nodes to be simulated or number of messages to be handled inside the WSN simulator.

The performance evaluation consisted of a various number of Java-based sensor nodes. Each node was installed with the application RimeABC, which is a default application provided by COOJA. This application was slightly modified in order to prevent any additional terminal- or logging output that could have altered the performance results. The application RimeABC transmits a 6 byte packet periodically to all surrounding nodes. Furthermore, it indicates the reception of a packet by raising an LED. We used a grid based topology for each scenario and the communication range was adjusted so that each node had a maximum of four neighbours. The reporting time initiated after a random initial sending time for the first message.

Each simulated sensor node has its own state and mem-

ory, it produces additional messages for transmission and it is able to receive transmissions from surrounding nodes. In result, the simulation time is highly affected by the number of simulated nodes, as reported in Figure 6. In this test, we measured the runtime of the simulator with respect to an increasing number of simulated nodes. Each value was generated by using 5 test runs with identical configuration but different random seeds. The simulation time for each test was 30 minutes.

Figure 6 shows the speed of a simulation with respect to the real time passed. A value below "1" means that the simulation could be performed faster than real time. For the evaluation, different numbers of nodes and different sending periods of the node applications were considered. The figure shows that the runtime of a simulation exhibit quadratic growth in terms of the number of nodes. However, increasing the sending period to 1 message per minute with a deployment of 1600 nodes can still be simulated faster than real time. This number represents the maximum number of sensor nodes, to be simulated by a single simulation engine. If several networks have to be simulated, with each of them just connected through the middleware but not through low power communication, the total number can be much higher (see also Section 5.3.2).

The number of messages managed by a simulation highly affects the simulation speed, since each message sent during the runtime of the simulation must be stored and maintained within the simulator and must be processed by all receiving nodes. The application RimeABC transmits a message in periodic intervals. For testing the scalability of the simulator, we used different message intervals. Figure 7 shows the result of the evaluation with different message intervals and nodes. As shown by the figure the speed is inverse dependant on the message interval.

5.3.2 Communication Middleware Performance

As mentioned before , the performance of the communication middleware can affect the simulation performance. To understand such impact for KASSANDRA, we show the scalability analysis on the PLANET middleware with increasing number of messages to represent increasing number of interacting actors (both real- and simulated COs). We test the middleware in terms of sending frequency and the payload of a message. It is important to notice that the number of actual messages sent by a module strongly depends on the actual application. For example, in the Doñana Pollution Detection scenario, each sensor reported a reading every 15 minutes to a base station connected to the PLANET middleware. We took the biggest status information sent by a Cooperating Object as reference message for estimating the number of devices that can be simulated.

The performance tests were performed on an ad-hoc testbed, in order to facilitate the measurement of the required metrics. The machines used were two identical IBM ThinkPad T60 laptops with Windows 8.1 as operating system. The network connectivity was established using wired Gigabit LAN. To analyse the achievable throughput, messages were sent for 10 minutes, containing a payload between 1 Byte and 10000 Byte and with a sending rate between 10 messages/s and 200 messages/s.

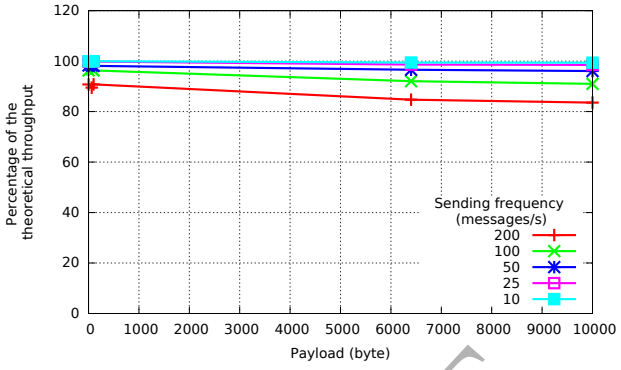


Fig. 8: Evaluation of the percentage of the theoretical throughput achieved for different message frequencies depending on the payload size. Values at: 1, 64, 100, 6400, and 10000 bytes.

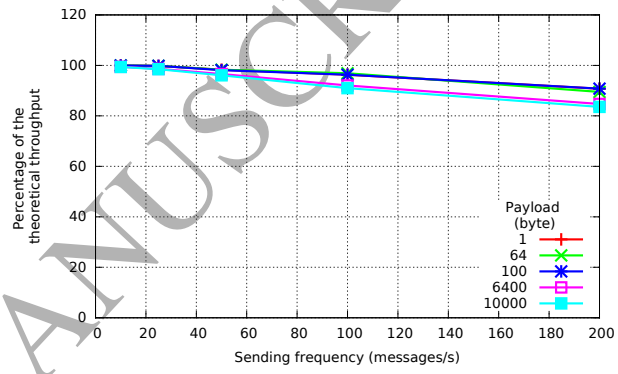


Fig. 9: Evaluation of the percentage of theoretical throughput achieved for different payload sizes depending on the sending frequency. Values at 10, 25, 50, 100, and 200 messages/s.

Figure 8 shows the percentage of the throughput achieved in the experiments compared to the theoretically ideal maximum throughput. The ideal maximum throughput is the theoretical number of messages sent in the 10 minutes of the experiments duration. In most cases, the PLANET middleware performs with values greater than 90% of the theoretical maximum, with a minimum at around 83%. During that experiment, we noticed a slight number of message losses when sending big messages at a high frequency. However, considering our assumption of a message of 155 bytes sent every 5 seconds, the Platform put no strong limitations on the number of COs to be simulated.

Figure 9 shows the performance of the PLANET middleware according to the payload size of a message. It shows that the payload size mostly has a constant overhead, which is due to the PLANET middleware and the DDS encoding and decoding of messages.

Absolute numbers for the throughput are depicted in Figure 10 (for different payloads) and in Figure 11 for different message frequencies. Both figures show a linear increase in throughput for the tested parameter range. Indeed, this is confirmed by official benchmarks provided by RTI, that identified the network capabilities as the limiting factor for the throughput and not the protocol overhead

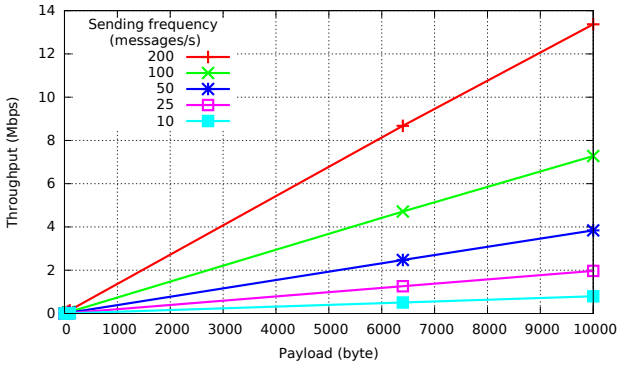


Fig. 10: Evaluation of the absolute throughput achieved for different sending frequencies depending on the payload size. Values at: 1, 64, 100, 6400, and 10000 bytes.

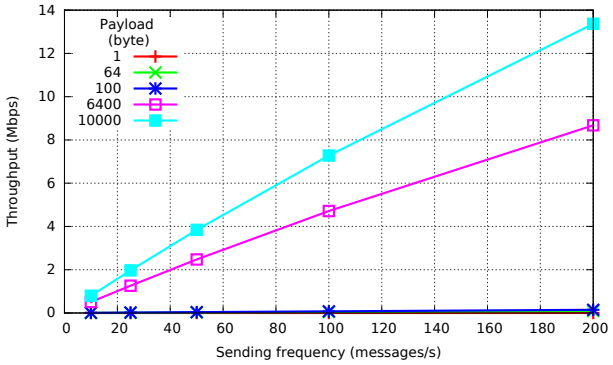


Fig. 11: Evaluation of the absolute throughput achieved for different payload sizes depending on the sending frequency. Values at 10, 25, 50, 100, and 200 messages/s.

[41]. For the limited number of devices required for target scenarios, KASSANDRA never reached the limits of the PLANET communication middleware, or the underlying DDS middleware respectively. Readers interested in more detailed information are referred to further publications on DDS performance evaluation for wired communication [42], or for wireless communication [43].

The performance analysis results indicates that neither the WSN simulation nor the middleware place a strong limitation on the number of COs to be simulated in KASSANDRA. With this, we ensure CO simulations can be performed smoothly in KASSANDRA for our application scenarios.

6 Conclusion

Achieving high simulation accuracy for performance analysis of Cooperating Objects applications require multidomain considerations regarding node heterogeneity, simulation extensibility and dynamic environments. In this paper, we address these issues and propose the KASSANDRA framework that enables COs application simulation using heterogeneous real- and simulated COs. The architecture of KASSANDRA expploits the existing communication middleware based on DDS to seamlessly interconnect real and simulated modules. We believe that such approach do not only allow more application simulation scenarios, but also improves the simulation accuracy with real-world objects in actual environments. As a result, it makes also possible to analyse the system behaviour at each stage of the application lifecycle, validating application requirements and identifying effective system configurations.

We demonstrated the scalability of KASSANDRA, which allows to simulate a deployment of 1600 sensor nodes, each sending one message per minute, faster than real time. Moreover, the communication middleware could deal with 200 messages per second, with each message composed of 10.000 bytes, assuming a stable network connection. Finally, the use of the KASSANDRA prototype in the PLANET application scenarios demonstrated the applicability of our approach.

References

[1] C.-Y. Shih, J. Capitán, P. J. Marrón, A. Viguria, F. Alarcón, M. Schwarzbach, M. Laiacker, K. Kondak, J. R. Martínez-de Dios, and A. Ollero, "On the Cooperation between Mobile Robots and Wireless Sensor Networks," in *Studies in Computational Intelligence*, 2014, vol. 507, pp. 67–86.

[2] R. Soleymani-fard, C.-Y. Shih, M. Baudewig, and P. J. Marrón, "COPlanner: A Wireless Sensor Network Deployment Planning Architecture Using Unmanned Vehicles As Deployment Tools," no. c, pp. 73–76, 2012.

[3] H. Kuntze, C. W. Frey, I. Tchouchenkov, B. Staehle, E. Rome, K. Pfeiffer, a. Wenzel, and J. Wollenstein, "SENEKA - sensor network with mobile robots for disaster management," *Homeland Security (HST), IEEE Conference on Technologies for*, pp. 406–410, 2012.

[4] M. Di Francesco, S. K. Das, and G. Anastasi, "Data Collection in Wireless Sensor Networks with Mobile Elements," *ACM Transactions on Sensor Networks*, vol. 8, no. 1, pp. 1–31, aug 2011.

[5] A. de San Bernabé, J. R. Martínez-de Dios, C. Regoli, and A. Ollero, "Wireless Sensor Network Connectivity and Redundancy Repairing with Mobile Robots," 2014, pp. 185–204.

[6] J. R. Martinez-De Dios, K. Lferd, A. De San Bernabé, G. Núñez, A. Torres-González, and A. Ollero, "Cooperation between UAS and wireless sensor networks for efficient data collection in large environments," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 70, no. 1-4, pp. 491–508, 2013.

[7] R. Figura, O. Schmitz, T. Hagemeier, M. Ceriotti, F. Brockmann, M. Mulero-Pazmany, and P. J. Marron, "Icelus: investigating strategy switching for throughput maximization to a mobile sink," in *12th Annual Conference on Wireless On-demand Network Systems and Services (WONS)*, Jan 2016, pp. 1–8.

[8] M. Schwarzbach, K. Kondak, M. Laiacker, C.-Y. Shih, and P. J. Marrón, "Helicopter UAV systems for in situ measurements and sensor placement," in *IEEE International Geoscience and Remote Sensing Symposium*. IEEE, jul 2012, pp. 4766–4769.

[9] P. Corke, S. Hrabar, R. Peterson, D. Rus, S. Saripalli, and G. Sukhatme, "Autonomous deployment and repair of a sensor network using an unmanned aerial vehicle," in *IEEE International Conference on Robotics and Automation, (ICRA)*. IEEE, 2004, pp. 3602–3608 Vol.4.

[10] G. Werner-Allen, K. Lorincz, M. Ruiz, O. Marcillo, J. Johnson, J. Lees, and M. Welsh, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Computing*, vol. 10, no. 2, pp. 18–25, March 2006.

[11] P. Marron, C. Shih, R. Figura, S. Fu, and R. Soleymani, "Challenges in the Planning, Deployment, Maintenance and Operation of Large-Scale Networked Heterogeneous Cooperating Objects," *the Fifth International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies (UBICOMM)*, no. c, pp. 338–343, 2011.

[12] "NS2." [Online]. Available: http://www.isi.edu/nsnam/ns/

[13] A. Varga and R. Hornig, "An Overview of the OMNeT++ Simulation Environment," *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, pp. 60:1—-60:10, 2008.

[14] "Castalia." [Online]. Available: https://castalia.forge.nicta.com.au/index.php/en/

[15] "NesCT." [Online]. Available: http://nesct.sourceforge.net/

[16] P. Levis, N. Lee, M. Welsh, and D. Culler, "TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications," in *Proceedings of the first international conference on Embedded networked sensor systems (SenSys)*. New York, USA: ACM Press, 2003, p. 126.

[17] B. Titzer, D. Lee, and J. Palsberg, "Avrora: scalable sensor network simulation with precise timing," in *Fourth International Symposium on Information Processing in Sensor Networks (IPSN)*, IEEE. IEEE, 2005, pp. 477–482.

[18] F. Österlind, A. Dunkels, J. Eriksson, N. Finne, and T. Voigt, "Cross-level sensor network simulation with COOJA," in *Proceedings - Conference on Local Computer Networks, LCN*, 2006, pp. 641–648.

[19] "TinyOS." [Online]. Available: http://www.tinyos.net/

[20] Y. Wen, R. Wolski, and G. Moore, "Disens," in *Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP)*. New York, USA: ACM Press, 2007, p. 24.

[21] G. Coulson, M. Anwander, G. Wagenknecht, S. P. Fekete, A. Kröller, T. Baumgartner, B. Porter, I. Chatzigiannakis, C. Koninis, S. Fischer, D. Pfisterer, D. Bimschas, T. Braun, and P. Hurni, "Flexible experimentation in wireless sensor networks," *Communications of the ACM*, vol. 55, no. 1, p. 82, jan 2012.

[22] P. Lu and Q. Geng, "Real-time simulation system for UAV based on Matlab/Simulink," in *IEEE 2nd International Conference on Computing, Control and Industrial Engineering*. IEEE, aug 2011, pp. 399–404.

[23] C. S. Malavenda, "Jaguar 4x4 UGV: from dynamic model to autonomous navigation in an outdoor wireless sensor network field," no. 3, pp. 1213–1218, 2012.

[24] D. Santamaría, F. Alarcón, A. Jiménez, A. Viguria, M. Béjar, and A. Ollero, "Model-Based Design, Development and Validation for UAS Critical Software," *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1-4, pp. 103–114, jan 2012.

[25] "JSBSim." [Online]. Available: http://jsbsim.sourceforge.net/

[26] "FlightGear." [Online]. Available: http://www.flightgear.org/

[27] "Openeagle." [Online]. Available: www.openeaagles.org/

[28] "Gazebo." [Online]. Available: http://gazebosim.org/

[29] "V-Rep." [Online]. Available: http://www.coppeliarobotics.com/

[30] "Robot Operating System." [Online]. Available: http://www.ros.org/

[31] G. Pardo-Castellote, "OMG Data-Distribution Service: Architectural overview," *Proceedings - 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pp. 200–206, 2003.

[32] "Data Distribution Service." [Online]. Available: http://www.omg.org/spec/DDS/

[33] "Advanced Message Queuing Protocol." [Online]. Available: https://www.amqp.org/

[34] "JMS-Specification." [Online]. Available: https://www.jcp.org/en/jsr/detail?id=914

[35] L. Vrdoljak, I. Bojic, V. Podobnik, and M. Kusek, "The amigo-mob: Agent-based middleware for group-oriented mobile service provisioning," in *10th International Conference on telecommunications (ConTEL)*, June 2009, pp. 97–104.

[36] P. Costa, G. Coulson, C. Mascolo, L. Mottola, G. P. Picco, and S. Zachariadis, "Reconfigurable component-based middleware for networked embedded systems," *International Journal of Wireless Information Networks*, vol. 14, no. 2, pp. 149–162, 2007.

[37] M. G. Valls, I. R. Lopez, and L. F. Villar, "iland: An enhanced middleware for real-time reconfiguration of service oriented distributed real-time systems," *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 228–236, Feb 2013.

[38] "ATLAS." [Online]. Available: http://atlascenter.aero/

[39] M. Mulero-Pázmány, J. Á. Barasona, P. Acevedo, J. Vicente, and J. J. Negro, "Unmanned Aircraft Systems complement biologging in spatial ecology studies," *Ecology and Evolution*, vol. 5, no. 21, pp. 4808–4818, nov 2015.

[40] D. Pain, A. Sánchez, and A. Meharg, "The Doñana ecological disaster: Contamination of a world heritage estuarine marsh ecosystem with acidified pyrite mine waste," *Science of The Total Environment*, vol. 222, no. 1-2, pp. 45–54, oct 1998.

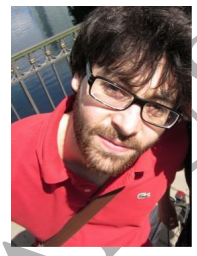
[41] "RTI DDS Benchmarks." [Online]. Available: https://www.rti.com/products/dds/benchmarks.html

[42] M. Xiong, J. Parsons, J. Edmondson, H. Nguyen, and D. C. Schmidt, "Evaluating the performance of publish/subscribe platforms for information management in distributed real-time and embedded systems," 2011.

[43] B. Almadani, M. N. Bajwa, S.-H. Yang, and A.-W. A. Saif, "Performance evaluation of dds-based middleware over wireless channel for reconfigurable manufacturing systems," *International Journal of Distributed Sensor Networks*, vol. 11, no. 7, 2015.

Richard Figura is a doctoral student at the Pervasive Computing, Networked Embedded Systems Group (NES), University of Duisburg-Essen in Germany. He received his diploma degree in Computer Science ("Informatik") from the University of Bonn in 2010. His research interests include the simulation based analysis and reconfiguration of sensor network deployments with mobile elements.

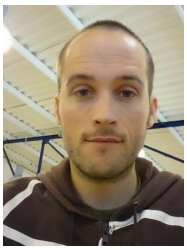
Chia-Yen Shih is a postdoctoral researcher at the Pervasive Computing, Networked Embedded Systems Group (NES), University of Duisburg-Essen in Germany. She received her Ph.D. in Electrical and Computer Engineering at the University of California, Irvine CA, USA, in 2008. Her research interests include Sensor Network Planning and Deployment, Internet of Things and Smart Sensing.



Matteo Ceriotti is a postdoctoral researcher at the Pervasive Computing, Networked Embedded Systems Group (NES), University of Duisburg-Essen in Germany. Prior to this, he worked as a researcher at the RWTH Aachen University, Germany and the Bruno Kessler Foundation—IRST, Italy. His research interests include the design and deployment of wireless embedded systems in real settings.

Songwei Fu is a doctoral student at the Pervasive Computing, Networked Embedded Systems Group (NES), University of Duisburg-Essen in Germany. He received his Bachelor degree in Physics at the Nanjing University, P.R. China in 2004 and his Master degree in electrical engineering at University of Kaiserslautern, Germany in 2010. His research interests include performance analysis of wireless communication and monitoring and healing techniques for wireless sensor networks.

Falk Brockmann is a doctoral student at the Pervasive Computing, Networked Embedded Systems Group (NES), University of Duisburg-Essen in Germany. He received his Master degree in Computer Science ("Informatik") from Leibniz Universität Hannover in 2011. His research interests include Device-free Passive Motion Detection with wireless sensor networks.



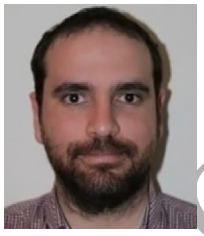
Héctor Nebot Molmeneu received his Telecommunication Engineering in the Polytechnic University of Valencia, Spain in 2008. Afterwards he received his Master Degree in Information Technology Integration Consulting also the Polytechnic University of Valencia in 2009. Since 2009 he has been working in ETRA Research and Development in the transport and hardware department. His main interests are in the field of contactless technologies and also automated testing platforms based on Labview.

Francisco Alarón Romero received both his Master Degree in Telecommunication Engineering and E.T.S. Degree in Computer Engineering from the University of Seville, Spain, in 2008 and 2013, respectively. From 2007 to 2009, he was Research Fellow at the University of Seville and the Andalusian Association for Research and Industrial Cooperation (AICIA). Since 2009 he has been working for the Center for Advanced Aerospace Technologies (CATEC), Seville, Spain. His interests are in the field of avionics and unmanned aerial systems.

Andrea Kropp , born 1970 in Catanzaro, Italy, received degree in Electronics Engineering in the 1995. He is currently Project Leader in the European R&D Projects. He joined SELEX Sistemi Integrati, formerly Datamat, and now Leonardo - Finmeccanica, in 1997, in R&D Unit. His expertise spans from system Integration, technological scouting, analysis, to product design and prototype, in the HF/VHF/UHF high data rate communications, Wireless technologies, Machine-2-Machine Systems, Geographic Information Systems (GIS) fields.

Konstantin Kondak is the head of the Flying Robots research group at the German Aerospace Center (DLR) Institute of Robotics and Mechatronics in Wessling, Germany. He received the Computer Science degree from the Technical University of Berlin (TUB) in 1999. He also received the Ph.D. and postdoctoral degree in Robotics from the same university in 2001 and 2006, respectively. He was also the head of the Real-Time Systems and Robotics research group at TUB (representative full professorship) working on different subjects in computer science and robotics, including exoskeletons and UAVs and participates in several national and international projects, amongst them the FP7 projects ARCAS and EC-SAFEMOBIL for the European Commission.

Marc Schwarzbach worked as a postdoctoral researcher at the flying robotics research group at German Aerospace Center (DLR) in Wessling, Germany. He received his diploma degree at the University of Stuttgart in 2005 and his Ph.D. in Eingineering in 2012. His research interests include unmanned aircraft systems and applications in different fields. He is now heading the UAV development group of Autel Europe in Gilching, Germany.



Antidio Viguria received the “Engineering” degree in telecommunication from the University of Seville, Spain, in 2004. From 2004 to 2006 he was working as a researcher at the University of Seville. From 2006 to 2008, he was a Fulbright scholar and a graduate student at the Georgia Institute of Technology. Since 2008 he is with the Center for Advanced Aerospace Technologies (CATEC), Seville, Spain, where he is Head of Avionics and Unmanned Systems Department. His main research interests are in multi-robot coordination and cooperation and robot architectures.

Margarita Mulero-Pázmáni , Ph.D. in Biology and MBA, started her research career in CSIC, Spain, where she focused on multidisciplinary projects using Unmanned Aircraft Systems combined with Internet of Things as tools for solving practical environmental problems and to answer theoretical ecological questions. While doing research activity in ecology (in Vogelwarte, Switzerland), she also provides assessment to biology research groups at different universities (e.g. Mexico and Ecuador) to develop lines of work involving UAS. Currently, she is also a lecturer at University of Cádiz, on the course “Expert on UAS civil applications” and works as a consultant for private companies developing UAS for environmental purposes.

Gianluca Dini received his Electronics Engineering Degree from the University of Pisa (IT), in 1990 and his Ph.D. in Computer Engineering from Scuola Superiore S.Anna, Pisa (IT), in 1995. In 1993 he was research fellow at the Department of Computer Science of the University of Twente (NL). In 1999 he was adjunct professor at the University of Siena. From 1993 to 2000 he was assistant professor at the University of Pisa where is now Associate Professor. In 2014, he got the National Scientific Qualification to function as Full Professor in Computer Science. He has been (co)author of 100+ papers in international journal and conferences. He has been principal investigator in quite a few projects funded by the European Union, the Italian Government, and private companies. His research interests are in the field of distributed systems, operating systems, and cyber-security.

Jesús Capitán is assistant professor at the University of Seville in Spain. He received his Master degree in Telecommunication Engineering and his PhD in Robotics at the same university (2006, 2011). His has also worked as postdoctoral researcher at the Instituto Superior Tecnico (Lisbon, Portugal), and the University of Duisburg-Essen (Essen, Germany). His main research topics are cooperative multi-robot systems, planning under uncertainty and decentralized data fusion.

Pedro José Marrón received his bachelor and master’s degree in computer engineering from the University of Michigan in Ann Arbor, USA in 1996 and 1998 respectively. He received his Ph.D. with honours from the University of Freiburg in Germany in 2001. From 2005 to 2009, he was head of the Sensor Network and Pervasive Computing group in Bonn. Since 2009, he is the head of the Networked Embedded Systems group at the University of Duisburg-Essen and since 2012 Director of the newly founded research Center “UBICITEC” and a lead scientist at Fraunhofer FKIE in Wachtberg. His current research interests are distributed systems, location-aware computing, sensor networks and pervasive systems.