Model Continuity in Cyber-Physical Systems: a control-centered methodology based on agents

Franco Cicirelli[a], Libero Nigro[b], Paolo F. Sciammarella[b]

[a] *CNR - National Research Council of Italy Institute for High Performance Computing and Networking (ICAR) - 87036 Rende (CS) Italy*
[b] *Engineering Department of Informatics Modelling Electronics and Systems Science University of Calabria, 87036 Rende (CS) - Italy*

Abstract

A Cyber Physical System (CPS) is given by the integration of cyber and physical components, usually with feedback loops, where physical processes affect computations and vice versa. Design and implementation of complex CPSs is a multidisciplinary and demanding task. Challenges arise especially for the exploitation of heterogeneous and different models during the various phases of system life cycle. This paper proposes an agent-based and control-centric methodology which is well suited for the development of complex CPSs. The approach is novel and supports *model continuity* which enables the use of a unique model along all the development stages of a system ranging from analysis, by simulation, down to real-time implementation and execution. In the paper, basic concepts of the methodology are provided together with implementation details. Effectivenesses of the approach is demonstrated through a case study concerning a prototyped CPS devoted to the optimization of power consumption in a smart micro-grid automation system.

Keywords: Multi-agent systems, control-based methodology, actors, parallel actions, model continuity, Cyber-Physical Systems, simulation, real-time execution, power management, smart micro-grid

1. Introduction

Cyber-physical systems (CPSs) [1, 2, 3, 4] integrate a physical system with a computational part through a network infrastructure. Their exploitation is advocated in various domains including avionics, automotive, traffic management, health care system, mobile communications, medical technology,

manufacturing, smart grid, procurement and logistics, industry and building automation, plant construction and engineering [5]. A correct design for CPSs is of great importance as they are often applied in safety or business-critical contexts [6].

CPS development challenges arise from the necessity of adopting powerful software engineering methods for the cyber part, capable of ensuring modularity and evolution of a software architecture, while at the same time guaranteeing an effective control of the runtime platform and communication network for the fulfillment of the physical plant real-time constraints. Design difficulties [7, 6, 8] are related, for instance, to the needs of conjoining continuous dynamics of the physical components with the discrete time model of the cyber components. In addition, the use of open and public networks requires the handling of security concerns [9] arising from the real-time operation of a CPS.

Architectural means for CPS modelling are described, for instance, in [10], where the use of agents [11] and their interactions (events) to one another and with the external controlled environment are the basic concepts. The adoption of crosscutting agent coordination policies at both the local and the global/system level emerged as a fundamental way to control the achievement of system goals. Multi-agent systems have demonstrated their advantages as an open and flexible software technology capable of unifying control aspects in smart grid applications [12]. As an example, agents were used to handle the power management problem in a smart home automation system [13]. Holonic agents, instead, are used in [14] as basic architectural building blocks for the development of manufacturing automation systems.

In this work an original agent-based control framework [15, 16] is advocated for CPSs, which rests on mechanisms for managing control and coordination aspects of agents as in [10]. Managing control aspects means that the approach makes it possible to use, in a transparent way, different message scheduling and dispatching policies according to a chosen time notion (real or virtual) so as to fulfill specific application requirements. The control framework acts as an *operating software* solution that integrates both flexibility of an agent-based design [13] with time-sensitive control structures which coordinate agents' evolution. A unique feature of the adopted framework, not supported by other existing agent-based approaches for CPSs, is *model continuity* [17, 18], which consists in the possibility of transitioning unaltered an agent model throughout the entire development life cycle, from analysis, down to design, implementation and real-time execution. The approach pro-

vides also a concurrency model which favours predictability and determinacy by avoiding common pitfalls of multi-threaded programming [19] (see section 3.2).

With respect to other approaches supporting model continuity [17, 18], the proposed framework distinguishes by its abstraction mechanisms which enhances separation-of-concerns during the development of CPSs. In particular, the following abstraction entities can be exploited: (i) agents to structure the *business logic* of the application to realize, (ii) *boundary elements* to interface the application with the external physical environment, (iii) the environment Gateway (*envGateway*) taking into account aspects related to modelling, analysis and implementation of the physical part of a CPS and more in general of the external environment in which an application runs, and (iv) customisable time-sensitive *control structures* suited to scheduling and dispatching system events and message exchanges. Model continuity depends on different concretizations of the boundary elements, the envGateway and the control-specific components. The envGateway requires to be re-interpreted when moving from the analysis to the implementation phase. It offers a transparent yet uniform way for dealing with communication protocols and hardware equipments needed for sensing and acting upon a controlled environment. During system analysis, besides the modelling of single sensors and actuators, the envGateway takes also into account the causal-effect relations tied to operations carried out on the environment. As an example, turning-on a lamp through a relay implicitly affects the value read by a luminosity sensor, placed near the lamp itself.

During the simulation phase the envGateway can also interface software components like ordinary differential equations (ODEs), modelling continuous time behavior of a system plant. From this point of view, the proposed approach can integrate continuous models within an overall discrete-event based framework. As an example, such techniques as quantization [20, 21], experimented, e.g., in the DEVS community [22], can be used.

In this paper the above mentioned agents and control framework is tailored to CPSs and the focus will be on proposing a methodology which addresses all the development stages of a system.

The paper is an extended version of authors' previous work published in conferences [23] and [24]. With respect to previous authors' works, the contributions of this paper are the following: (i) the methodological aspects of the proposed approach are better defined and weaved so as to cover all the development phases, ranging from modelling, analysis, to final implementa-

tion of a CPS; (ii) more insights are provided about the envGateway design and about the hardware equipments necessary to instrument a real CPS; (iii) the methodology is practically demonstrated through a case study concerned with optimizing power management in a smart micro-grid home or industrial electric power system.

The paper structure is as follows. Section 2 describes related works about CPS methodologies and challenges. Section 3 details the proposed methodology and outlines the adopted agent and control based framework. Section 4 presents the case study and applies to it all the phases of the methodology. Some experimental results are reported and discussed. Section 5, finally, draws some conclusions and furnishes indications about some on-going and future work.

2. Related Work

CPS engineering challenges include the use of integrated models, facing issues related to interoperability, reconciliation of Newtonian time of the physical part with the discrete time of the cyber part [8], privacy protection, security, non-functional requirements, timing constraints, humans-system cooperation and so forth [5].

Service-oriented architectures (SOA) and multi-agent systems (MAS) are two important software technologies which have proved their effectiveness in general ICT systems and whose exploitation for CPS is deemed promising to sustain a revolution in industry automation and smart factories [25, 26, 27, 28, 29, 30].

A service-based approach for developing CPS is proposed in [25] which exploits service-oriented architecture concepts and/or cloud concepts to realize service-based CPS. The approach deals with some design challenges of CPSs such as dynamic composition, dynamic adaptation, and high confidence CPS management, hardware heterogeneity. Three tiers were defined: an *Environmental Tier* for dealing with the target physical environment, a *Control Tier* for making decisions for networked physical devices, and a *Service Tier* for managing reusable services. The final goal is that of allowing the handling of complex and resource-consuming processes even on downsized mobile Internet devices which are usually involved in a CPS.

Another service-based approach is discussed in [26] where the WebMed middleware is proposed. The goal is promoting the use of the service metaphor for the development of CPS applications. By exploiting the service-oriented

computing, WebMed fosters the realization of loosely coupled CPS infrastructures that expose the functionality of physical devices as Web services. Exposed functionalities can be easily integrated with other existing software components. The middleware consists of: (i) a *WebMed device adapter*, aiming at hiding heterogeneity related to the use of specific hardware, data structures and communication protocols; (ii) a *Web service enabler*, which provides a mechanism for the data and functionalities of a physical device to become accessible as a Web service; (iii) a *service repository*; (iv) an *engine*, which is the core element providing a runtime environment for all Web services and operations in the middleware; and (v) an *application development* tool providing high-level management of interaction and composition of Web service components in the middleware. The latter serves as user interfaces for developers, and as front-end in order to invoke a developed Web service.

In [29] MAS and SOA are identified as strategic technologies for CPS development and industry automation [14]. Agents contribution mainly derives from being decentralized, autonomous and modular entities, encapsulating data and "intelligence", and interacting to one another (for sociality and holonic aspects [14]) for the fulfillment of goals which could not be reached by each agent operating in isolation (property emergence at the society/population level). Other relevant agent features include robustness, flexibility, learning and adaptation, and self re-configurability. The usage of MAS, though, can be critical from the timeliness point of view. Therefore in [29, 14] the notion of an "industrial agent" is envisioned where an agent is paired with a Programmable Logic Controller (PLC) for low-level control and responsiveness, while ensuring, at the higher level, intelligence and adaptation. Services are purposely combined with agents in [29], by abstracting and exposing agent functionalities and low-level control through services, to favor in-the-large interoperability, modularity and composability. In [14] holonic agents interact and coordinate each other by FIPA [31] inspired mechanisms. Various kinds of simulators, already existing or especially developed for specific needs, are used to validate a control solution.

The work described in this paper argues that an agent framework together with a control-based approach, can be a basis for light-weight concurrency and effective timing control.

A methodological approach based on MAS is proposed in [28] for the analysis and prototyping of CPS. The analysis phase is directed to MAS simulation and CPS validation. However, the paper mainly focus on the system requirements elicitation, e.g., sensor measurements and effector actions,

using a specialization of SysML profile, and the assignment of requirements to behaviors of organizations which finally map on agents. The identification of organizations is helped by a problem ontology which describes all the concepts involved in a CPS and their relationships. The proposed methodology appears at a preliminary stage and has to demonstrate its effectiveness in the design of real systems.

Another agent-based approach aiming at developing CPS is proposed in [27]. A goal of the approach is that of trying to assess system behavior. Both qualitative and quantitative system property evaluation is considered. The quantitative evaluation, which is based on the exploitation of the INGENIAS methodology [32], is carried out by using a multi-agent model that supports event-driven behaviors. In the paper, the multi-agent approach is considered as the proper one to model a CPS with dependability features. This is due to the flexibility provided by agents as autonomous and intelligent components in decisions support actions. Raw data-streams, collected by various devices like sensors, video cameras, mobile phones, and measuring devices, are transmitted to the cyber components which, by using hardware and software facilities as well as communication connections, can provide several main functions as learning and adapting for intelligent control, self-maintenance, and self-organization.

An agent-based framework for smart factory is proposed in [30]. The framework consists of four layers, namely *physical resource layer*, *industrial network layer*, *cloud layer*, and *supervisory control terminal layer*. The physical resources are implemented as smart things which communicate each other through the industrial network. The integrated information system exists in the cloud which collects massive data from the physical resource layer and interacts with people through supervisory control terminals. All of this, actually forms a CPS where physical objects and informational entities are deeply integrated. Furthermore, a negotiation mechanism for agents to cooperate each other is proposed, and four complementary strategies are designed to prevent deadlocks by improving the agents decision making and the coordinators behaviour. Properties of the proposed framework are assessed through simulation. A simulation program has been developed by using the Microsoft VS integrated development environment (IDE).

In [6, 7] it is argued that the design of CPSs strongly requires both an integrated view and co-designing of the physical and the computational part of a whole system. Authors of [6] suggest a model-based view to cope with designing aspects of the hardware and software components, and their inter-

actions. The goal is to derive relationships among the design, analysis and implementation models so as to ensure, for example, that analysis results are reflected in the executable system. What is envisioned is an incremental, simulation-based development approach, where initially the whole system is simulated and, subsequently, simulated parts are replaced by real ones.

To summarize, CPS development is currently trying to exploit powerful methodologies capable of systematically addressing all the CPS design issues. Anyway, the research about models, approaches, middleware architectures or platforms for realising CPS applications is still in its infancy [25, 26, 6].

This paper claims that *model continuity* [16, 17, 18] and MAS technology are fundamental tools for CPS development. Model continuity naturally addresses the CPS requirement of *model integration* [6], that is the need of ensuring coherence between the properties assessed during model analysis with the properties exhibited by a system during its execution. However, no experimented methodology based on the concepts of model continuity is actually exploited for CPSs. The contribution of this paper is to propose a novel approach based on MAS and model continuity and to demonstrate its practical usefulness through the realization of a real CPS case study.

The proposed approach mainly focuses on methodological issues embedded within a discrete-event framework. The approach, though, can integrate continuous time (CT) components and co-simulation activities as permitted, e.g., by well-known frameworks and toolboxes supporting CPS modelling and analysis like DEVS [22] and PTolemy II [8]. Both DEVS and Ptolemy enable the construction of hierarchical complex models. In addition, Ptolemy too rests on actors as the basic building blocks. The model of computation of a non atomic actor model can be defined so as to work with either synchronous or asynchronous interactions. Code generators are finally in charge of transforming an analyzed model into a final implementation. DEVS builds on a discrete-event world vision and has an efficient and modular simulation structure, which is open to interact with CT components. The DEVS community has experimented with such techniques as quantization [20, 21] for integrating CT components with discrete-event operation. With respect to CT components, which can be required during system analysis, the approach developed in this paper is able to exploit the same concepts and techniques of DEVS.

3. From Modelling to Implementation of a CPS

The life cycle of a CPS can be viewed as composed of different *transition phases*. First a model of the system is built. The model is then used to analyse its functional and temporal behaviour both in a simulation context and in a real execution environment. Thereafter, an analysed system can be put into operation.

In this section, a methodology is proposed which is based on the agent and control framework introduced in [15, 16] and here specialized for its use with CPSs. The approach relies on *pure-software components*, which *remain unchanged* during the transition from model analysis to system implementation, and on *hybrid components* which require to be concretized for actual system implementation. The methodology furnishes also entities suitable to model *external resources* needed by the system, and to capture and abstract the interactions between the system and the *external environment* the system operates in. In the following, the methodology is discussed together with its related entities. Current version of the control framework is prototyped on top of the JADE (Java Agent DEvelopment framework) open source FIPA compliant project [33], which provides a distributed architecture supporting agent naming, creation, execution, message passing, behaviour and mobility.

3.1. The proposed methodology

The development of a CPS follows four main phases, namely *modelling*, *analysis* (e.g., by standalone or distributed simulation), *preliminary execution* and *real execution*, which are described below.

3.1.1. The modelling phase

A model is built in terms of the following basic abstractions: *actors* (or agents), *messages*, *actions*, *processing units* and the *environmental gateway* (*envGateway*).

Actors and *messages* are pure-software components which capture the business logic of a model. Actors are thread-less agents whose behavior is patterned by a finite state machine, and whose communication model depends on asynchronous message passing (see Figure 1). Message processing is atomic and consumes a negligible time. A *time-stamp* can be attached to a message to specify when it has to be consigned to its recipient. If the time-stamp is not specified the message has to be delivered at the current time.

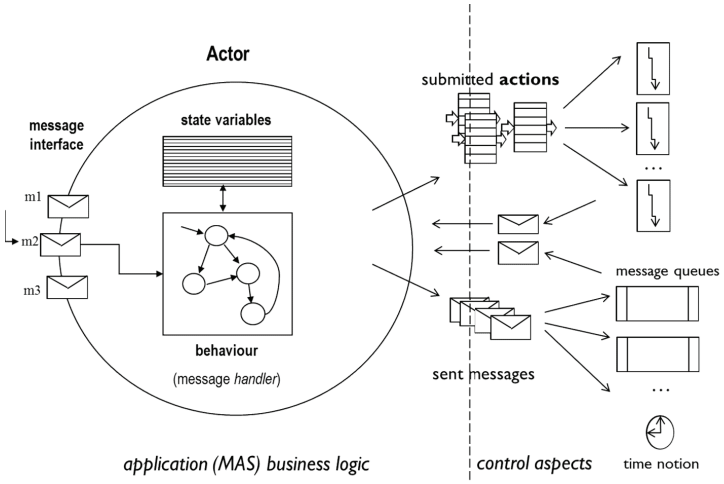

Figure 1: Actor structure and cross-cutting control aspects

Actions are self-contained computational entities which are submitted for execution by actors. Following its submission, an action can run to completion or it can be suspended/resumed or aborted [15]. Actions are hybrid-components modelling time-consuming tasks which require external entities not owned by actors (e.g., a document to be printed requires a printer to print it). Actions are executed on top of *processing units* which also are hybrid components. An action is ready to be executed as soon as it has been submitted. However, the available processing units actually determine *if* and *when* a submitted action is actually executed.

The *envGateway* is a hybrid component devoted to (i) modelling the external environment the actor-based application runs in, (ii) abstracting the interactions of an actor with its external environment, e.g., for sensing or actuation purposes. The *envGateway* plays the role of abstracting the external devices as well as hiding the used communication protocols. For example, if a temperature sensor is handled by an Arduino device [34], only the *envGateway* is aware of the presence of Arduino and of the specific protocol adopted for interacting with the temperature sensor. From the application viewpoint, the only relevant thing is requiring a read operation from the sensor. In addition, the external environment needs to be modelled, e.g., with the help of continuous time components implementing ODEs, each time a carried out operation has a side effect on the environment itself. For instance, let's consider an application which monitors the temperature in a

room and, when the temperature goes below a certain threshold, activates a heating system. Within the real system, the activation of the heating system increases the temperature of the room and, as a consequence, the increased temperature is automatically observed by the sensor. During simulation, instead, the cause/effect relation existing between the heating system and the temperature read by the sensor requires to be explicitly considered. Such aspects are dealt with through the *envGateway* modelling.

3.1.2. The analysis phase

Properties and behaviour of a CPS actor model can be checked by simulation. The model can be simulated in a sequential context or it can be partitioned so as to be handled by distributed simulation, which is actually supported by the proposed framework. Model partitioning is achieved by allocating actors onto different computational nodes (containers or Logical Processes, LPs). Distributed simulation [35] can be required in the case a large/complex model has to be analysed, or in the case the model refers to a system which is intrinsically distributed.

The same model can be simulated in different conditions by simply configuring a different *simulation context*. Setting a particular simulation context corresponds to defining the *number* and the *behaviour* of the processing units as well as the *policy* adopted for scheduling and executing submitted actions. As an example, actions can be processed in A first-in-first-out order or in a priority-driven way, in which low-priority actions are suspended and subsequently resumed when no more high priority actions exist. The *action schedulers* are the entities which are responsible for managing action scheduling issues. Properties and behaviour of a same model vary as a different simulation context is considered. For example, if a call-centre is modelled where the calls-to-serve are expressed through actions and the receptionists by processing units, the study of how the number of served customers (i.e., the throughput of the model) changes as further receptionists get available, can be carried out by simply changing the number of the exploitable processing units, without any modification to the model. In this phase, all the hybrid-components are configured by their simulated counterpart.

During analysis, a *simulated* (i.e., virtual) *time notion* is used and, in addition, in the case of a distributed simulation, the evolution of the entire actor model has to be time-coherent among all the distributed simulators. Ensuring the right time notion and coordination among simulators is the responsibility of a *control machine*. A control machine is also responsible of

coordinating and actualizing message delivery to recipient actors. Message scheduling, dispatching and processing do not increase the simulation time. The simulation time augments only when a timed-message is processed or an action gets executed.

3.1.3. The preliminary execution phase

Preliminary execution is an intermediate stage between the simulation phase and the real execution of a system. A notable difference between this phase and the previous one concerns the used time notion which is no longer a simulated time but the *real time* (the wall-clock time) of the system. Therefore, all the time needed for processing messages, and for sending information through a network, are implicitly taken into account. In fact, the execution of the business logic of the system, e.g., the execution of a control or an optimization algorithm, can be expensive in terms of computational and time resources, and the real time required by the computation and communication ultimately depends on the chosen hardware infrastructure. Such real execution time is taken into account during the preliminary execution phase. As a consequence, this phase can be exploited to assess if the time constraints and system performance, previously checked in simulation, continue to be satisfied during real-time execution on top of the final exploitable hardware infrastructure. More in particular, the *behavioural drift* existing between simulation and real-time execution, i.e., the deviation between the actual processing of a timed message and its due time, can be quantitatively assessed during this phase.

Analogously to the previous phase, an *execution context* requires to be configured: the processing units and the action schedulers used in simulation must be replaced by the corresponding entities, able to deal with real-time and physical computational resources (e.g., processing units can be mapped on Java threads). Actions are implemented as pure resource-consuming tasks having a time duration and being capable of keeping busy a processing unit. The *envGateway* and all the pure software components remain exactly those used in the analysis phase.

A *real-time aware control machine* is now required, both in a sequential or distributed execution scenario.

3.1.4. The real execution phase

In this phase the system is put into real execution onto the target physical architecture. All the hybrid-components of the model are replaced by their

real counterpart. The control machine and the execution context coincide with those used in the previous phase. With respect to the preliminary execution, only the actions and the *envGateway* must be modified. Actions are reified so as to interact with physical devices and carry out real computational tasks, whereas the *envGateway* abstracts the used physical devices together with the communication protocol and physical infrastructure. Obviously, the pure-software components remain unchanged also in this phase.

3.2. Control machines and time management

A subsystem of actors (Logical Process or LP) is allocated for the execution on a computing node. All the actors of a same subsystem are governed by a *local control machine*, which transparently buffers exchanged messages into one or more message queues and ultimately consigns messages, one at a time, to recipient actors, according to a proper *control structure*, e.g., based on a specific time notion (simulated or real-time). Message processing is the unit of message dispatching (macro-step semantics). All of this determines a *cooperative* (i.e., not pre-emptive) *concurrency schema* for the local actors of an LP, ensured by message interleaving, which favors time predictability [15, 16].

Multiple actor subsystems (LPs) are federated to constitute a distributed system, using the services of a transport layer and communication protocol. Fig. 2 shows a distributed actor system as prototyped by using JADE. Both actors and messages can be dynamically transferred from an LP (JADE container) to another one. Migrating actors can be a need to ensure that an actor is located close, e.g., to a controlled device, or it can respond to dynamic load-balancing issues. A *Time Server* is responsible of maintaining a global time notion across the entire system. In a distributed simulation setting [16], all the control machines interact with the time server in order to negotiate time advancements so as to evolve all together in a coherent way. The exploitable APIs and the library of the available control machines are detailed in [15, 16].

3.3. Actions and processing units

By design, an action is a black box with a list of *input parameters* and a list of *output parameters*. Actions have no visibility to the internal data variables of the submitter actor and they do not share any data. Therefore, no mutual exclusion mechanism is needed and no interference problem can derive from the action parallel execution schema and message processing.

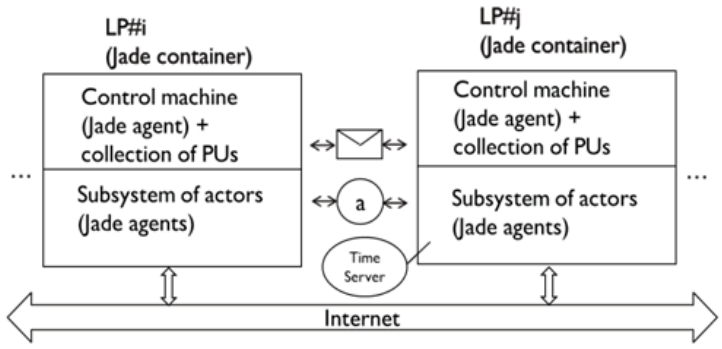

Figure 2: A JADE based distributed actor system

An *action completion message* can be generated by an action to inform its submitter actor about action termination.

Actions are hybrid-components which have different concretizations during the life cycle of a CPS. *Simulated actions* usually do not carry out any computation except that used to produce output parameters. They have a time duration, specified by an input parameter, which is an estimation of the time duration of the associated modelled task. *Real* or *effective* actions hide a concrete algorithm implementing a computational task. The execution of a real action increases the real time. Pseudo-real actions, used during a preliminary execution, advances the real time but have no concrete computation to perform.

In the case an action interacts with physical devices, the interactions are simulated in both the simulated and the *pseudo-real* actions. On the contrary, they are concretely implemented when a real action is used. A useful feature of actions, which is exploitable during the development of CPS, refers to the capability of returning partial computed results at some selected time points. The *return* primitive is made available for these purposes (see also the sequence diagram in Fig. 3). The *return* statement naturally can serve for implementing a periodic behavior within an action. In this case the time points are equally spaced within a time window assigned to the action. At action completion, an operation result message is issued.

An action scheduler administers the local processing units and stores actions which find no available processing unit in pending action queues. A processing unit is a hybrid-component: it can be a physical core or it can be realized by a Java thread, or it is a fake object in the case of simulated

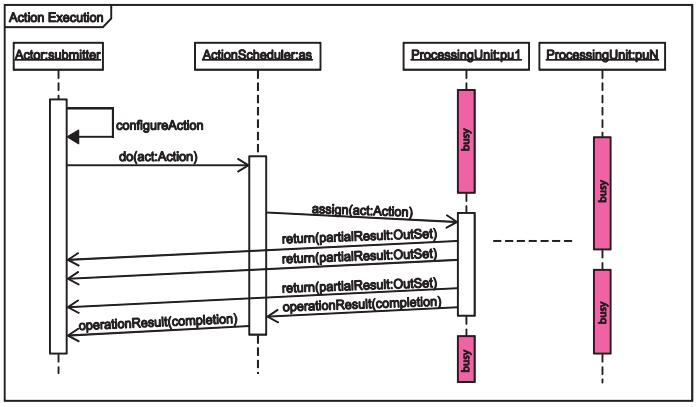

Figure 3: Message interplay during an action execution with multiple returns

actions. A detailed description of the supported kinds of actions, related schedulers and processing units can be found in [15, 16].

3.4. envGateway and environment control

During the analysis and the preliminary execution phases, the *envGateway* is exploited to abstract the environment within which the system operates, in the sense of mirroring the effects of the actuations upon the environment itself. In the following, a description of the implemented *envGateway* for the real-execution phase, is provided (see Fig. 4). During real-execution, the read/write operations are typically requested by submitted actions which, for generality, can be executed on dedicated Java threads. In a case, one action can be interested to get multiple sensor data, each one being related to a given time point within an assigned time window. The *envGateway* maintains a collection of data variables, which correspond to sensor/actuator devices. An In/Out layer in the *envGateway* is in charge of controlling the communication links with the physical devices and to update the data variables. In particular, the In/Out layer is composed of input/output Java threads, which interface the communication channels with a number of I/O hardware components, e.g., Arduino [34] or similar equipments. Sensors/actuators are physically linked to the I/O hardware. To simplify configuration and operation, each I/O hardware can be specialized to handling a disjoint subset of sensors or actuators.

The communication channels between the *envGateway* and the I/O hardware components, can either be based on the serial connection or on a wireless

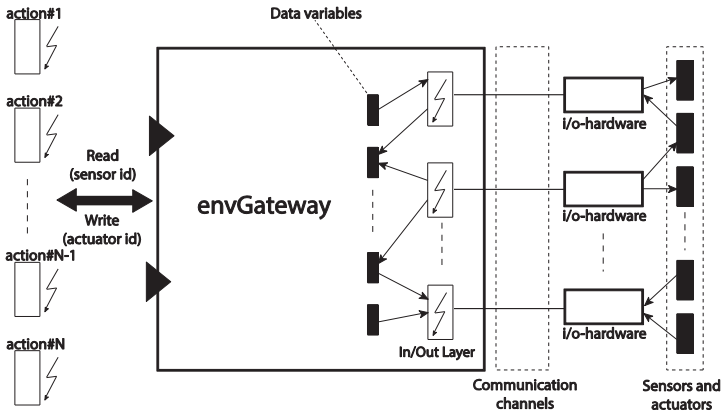

Figure 4: Organization of an *envGateway* component

connection. A suitable protocol requires to be established for the exchange of information between the *envGateway* and the I/O hardware devices. The protocol specifies the input/output operation, the involved physical device and (possibly) accompanying data (in an output command).

The *envGateway* was implemented as a monitor, which manages the action threads and the input/output threads, thus guaranteeing interference-free access to the I/O device data variables. More precisely, separate concurrent hash maps are used for handling the input (sensor) data variables and the output (actuators) commands and data. A design issue of the *envGateway* is concerned with the adoption of an anticipation schema as described in the following. The I/O hardware components are supposed to be programmed so as to repeatedly reading the sensors and providing the data to the *envGateway*. At any instant in time, the values of the data variables represent the most recent data values. Such values are then acquired by actions according to their own timing. For generality concerns, an action which needs some sensor data can provide a filter object at the request time. The filter exposes a *guard method* (i.e., a boolean function) which must be satisfied by the values of involved data variables for them to be actually returned.

Finally, it is worth noting, that correct behavior of a real execution of a CPS system, can require that the reaction to sensed data generated by a controller actor in the cyber part be provided within the sampling period of sensors of the physical part.

3.5. Specializing the envGateway to work with Arduino

The experiments described in this paper were accomplished by using the serial connection managed by the RXTX.jar Java library. The defined protocol between the *envGateway* and the I/O hardware components clarifies the input/output operation, the involved physical device and (possibly) accompanying data (for an output command). The *envGateway* was concretely interfaced with some Arduino [34] devices. Arduino configuration is carried out in the `setup()` function, where the details of pin connections with physical devices are defined. `setup()` is executed only once following a reset of the device. After that, Arduino enters its main `loop()`. The `loop()` instructions can be directed to reading from sensors and to put the data, after some A/D conversions, onto the communication channels towards the *envGateway*. At the end of the loop, a delay statement is executed before starting the next loop iteration. During its loop operation, Arduino can also receive and process interrupt signals. For example, a *serialEvent* interrupt which is raised whenever new data arrive through the serial communication link (RX), can be heard and managed only at the end of each loop iteration. The mechanism can be exploited to modify dynamically the amount of the loop delay. An Arduino dedicated to controlling only actuators, has an empty loop and all its output operations are delegated to the serial interrupt handling mechanism. Each interrupt signal is expected to be accompanied by all the command information needed for completing the output operation.

During analysis by simulation, the In/Out and I/O hardware layers can be transparently replaced by software agents which provide, in simulated time, pre-generated input data to the *envGateway* or simply consume output commands. Moreover, an *EnvAgent* can be introduced which through a mathematical model, fuzzy logic etc., is able to reproduce the necessary changes in the environmental variables monitored by the *envGateway*, implied by an actuation.

4. A case study using power management

A problem of electric power management [36, 37, 38], whose context can be a domestic home or an industrial plant, is considered. Motivation behind the problem stems from the need to exploit to the greatest extent the power generated, e.g., by a local photovoltaic panel, thus ensuring that the power loads in the context are dynamically activated/deactivated (i.e., scheduled) so as to optimally fit, at any instant in time, to the available generated power

(reference or threshold power signal). Indeed, it is not economically viable to sell the surplus of the local produced energy to the external electric provider as it is not properly paid.

The input for the case study is constituted by a threshold signal representing the generated power, and by a certain number of user power loads. Each load is characterized by a dynamic temporal behavior such as the start time and the duration (or computation cost) in the case of one-shot load, or the start time, the duration and a period in the case of a periodic load. Every load is tagged with a *utility* measure. The scheduler gives priority to loads having a greater utility. To avoid starvation, the utility is aged (i.e., increased) in the case a load gets not selected by the scheduler. To capture the quality of scheduler decisions, a *fitting* measure, inversely proportional to the offset between the overall consumed power and the reference threshold signal, is also considered. The scheduler decision takes place as soon as a variation is sensed either in the threshold reference signal and/or in the power loads (e.g., a load notifies it would execute, or it informs it just finished its execution).

The problem of CPS systems like the chosen case study, is trying to keep aligned the Newtonian time of the physical part with the discrete time of the cyber part. Depending on the physical dynamics of the controlled system, communication and computational delays can make a control reaction inconsistent (and possibly useless) with the actual state of the physical plant. The development of the case study was aimed at both assessing the timing problem and demonstrating the application of the proposed methodology with model continuity. The impact of the computational/communication overhead was checked by designing a scheduler agent which can search for an optimal solution (i.e., optimal load configuration), if there are any, through a full exploration of the solution space which can be computational demanding, or it can exploit a greedy heuristic which looks for a suboptimal, approximate solution, generated in a small amount of time.

For the purpose of the case study, an optimal solution is looked for by an iterative backtracking technique. An approximate solution, instead, is greedy searched by examining the candidate set of active loads, preliminarily ranked by decreasing utility and for the same utility by increasing power.

From a practical point of view, since the backtracking technique can be applied by specifying the number of required solutions, the heuristic solution can be generated as the first-found solution by backtracking, which operates on the candidate set of loads ranked by decreasing utility and then by in-

creasing power. The backtracking process prunes, as early as possible, those partial solutions which can be predicted they cannot become a full solution. Among the acceptable solutions, the optimal one is selected as the one which maximizes the overall utility of loads, and, among the solutions having the same maximal utility, the one which optimizes the fitting measure is preferred.

In the following, the development of the case study is detailed according to the various transition phases enabled by the approach. The provided description highlights the achieved benefits which stem from the exploitation of the same model during the development, to the capability of validating the correctness of the scheduling algorithms and predicting their overhead when executed on a real platform. Finally, it is shown how the implemented system is achievable by concretizing only the hybrid components as described in Section 3. The methodology is demonstrated without considering distribution aspects.

4.1. Modelling the system

The developed multi-agent model for the power control system consists of load agents (instances of the *LoadAgent* class), one *ThresholdAgent*, one *SchedulerAgent* and the *envGateway* for interacting with the external environment.

The *ThresholdAgent* handles the samples of the generated power signal. The *ThresholdAgent* helps separating the functionalities of the *SchedulerAgent* from those of the *envGateway*. It is the *ThresholdAgent* which reads, through a periodic action, the samples of the reference power signal and transmits them to the *SchedulerAgent*.

Each load agent manages a single physical power load. A *publish/subscribe* design pattern is adopted among the scheduler and the load agents. At its arrival, a *LoadAgent* first registers itself at the *SchedulerAgent* by an *Announcement* message (see also Fig. 5) which carries the identification data about the load. Subsequently, the *LoadAgent* communicates with the *SchedulerAgent* each time a variation in the temporal behaviour of the load occurs. Similarly, the *SchedulerAgent* sends commands to load agents for activating or deactivating the corresponding power load. When a load decides to abandon the candidate set of loads, it detaches from the *SchedulerAgent* through a Detach message. A *LoadAgent* is configured with the *id* of the controlled actuation device, the temporal parameters which regulate the load behaviour,

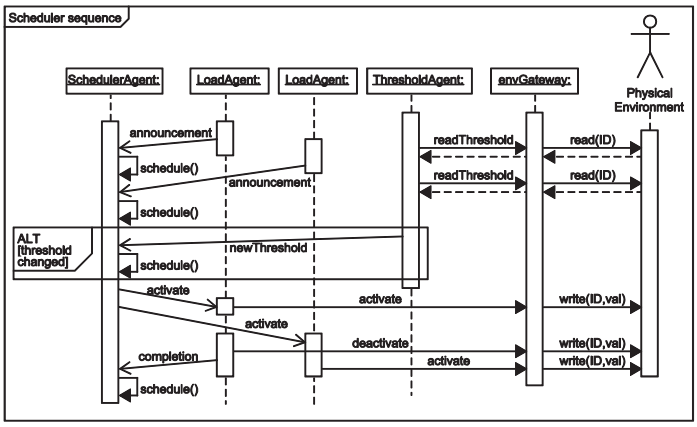

Figure 5: Main interactions among model agents

and the utility value of the correlated load. The agent also holds the active/inactive status of the corresponding physical load. Each time a load agent receives a command from the scheduler agent, it submits a one-shot action which implements the scheduler command through an interaction with the *envGateway*. An activation command sent to an already active load agent is simply ignored.

A load agent sends to itself a time-stamped message to step along the load power consumption curve, according to the basic time unit (e.g., 1s). Processing such a message causes an interaction with the *SchedulerAgent*, for communicating current load power level, activation/deactivation status and its actual utility value.

By design, the scheduler algorithm of the *SchedulerAgent* was directly coded in the `handler()` method of the agent. To avoid taking so long for the `handler()` to complete its calculations about an optimal solution, the `handler()` of the scheduler is kept busy only for the (minimal) time required for finding the next solution. A *Next* message is sent by the scheduler to itself for starting the computation of the next solution and so forth. In this way the scheduler remains capable of quickly sensing and processing variation events, which can require starting from scratch the scheduling process, just after one further solution was found.

Table 1: Reference threshold power signal

Available Power (W)	**Time Duration (t.u.)**
750	10
900	15
1300	20
2400	60
3300	205
1800	5
3100	100
2100	25
1800	10
1100	10
600	20

Table 2: Power loads parameters

Load ID	Utility	Power Request (W)	Time duration	Available at	Periodic
1	3	500	300	8	No
2	2	250	380	18	No
3	4	500	400	12	Yes
4	5	750	350	0	No
5	4	250	330	0	No
6	3	250	350	0	No
7	2	250	320	0	No
8	1	250	410	0	No

4.2. Data configuration

The multi-agent system model was experimented using the configuration data reported in Table 1 and Table 2. Table 1 specifies the adopted threshold (or available) power signal. The time-span of the available signal is 480 time units (*t.u.*) where a time unit corresponds to 1s of Newtonian time of the physical plant, that is the sampling period of the generated power signal. The Table 2 details the assumed power loads. Only the load #3 is periodic, and, following its termination, it becomes ready again to be scheduled after 12 *t.u.*.

4.3. Analysis phase

A first concern was studying in simulation the multi-agent system model. The goal was checking both functional and non functional (temporal) properties, particularly the behaviour of the scheduling algorithm. It is worth noting that, in simulation, message processing consumes 0 time. As a consequence,

the scheduler algorithm, being implemented in the `handler()` method of the *SchedulerAgent*, either searching an optimal or suboptimal solution, is always virtually completed in 0 time. Time advancement is mainly related to the duration of actions, and then to stepping through the power signal samples.

The model was executed on a standalone machine with the *Simulation* control structure [16]. *Simulated actions* are used and their execution immediately schedules the *completion message* or the message of the next *return*, which is time-stamped with their due time. The samples of the available power signal are pre-loaded in the *envGateway*. Output commands are simply logged.

When an active load is interrupted because the *SchedulerAgent* chooses a different load, its completion message is invalidated and the remaining time to completion is stored by the *LoadAgent* so as to be exploited at its next activation.

Fig. 6 portrays the scheduling effects on the total consumed power by loads with respect to the available threshold power signal, when an optimal or a suboptimal solution is adopted.

For the assumed loads (Table 2), the power consumption curves in Fig. 6 are very similar. However, since the scheduler tends to select different loads, differences will ultimately emerge between the two curves. At time 437 the periodic load is reactivated and the consumed power suddenly raises under optimal scheduling. Under suboptimal scheduling, instead, the same load reactivates at time 457. This is due to the fact that the optimal scheduler is capable of ensuring a greater power consumption in the time interval from 332 to 410 t.u..

Fig. 7 depicts the observed fitting, i.e., the deviation between the available power and the total consumed power, vs. time, in the two cases optimal/suboptimal scheduling.

As one can see from Fig. 7, the two algorithms tend to behave differently in the long time when, definitely, it *seems* that the optimal algorithm is outperformed by the suboptimal algorithm. In reality, since the optimal algorithm is capable of activating simultaneously more loads (although with a same total consumption power level) which in the considered case are almost one-shot, in the long time the optimal scheduler has fewer loads to manage and then it exhibits a greater deviation of the consumed power from the available power.

The above observations are confirmed by Fig. 8 which shows the total observed utility of the scheduled loads vs. time, in the two scenarios. Initial

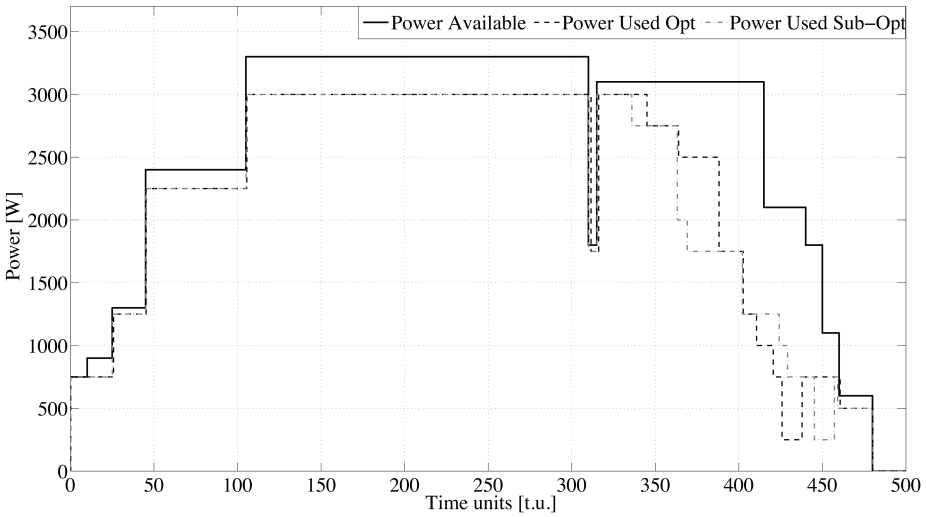

Figure 6: Power available and total consumed power vs. time optimal/suboptimal scheduling

and final differences are respectively due to the ageing process of the load utility and to the fact that definitely the optimal algorithm handles fewer loads.

4.4. Preliminary execution

Under preliminary execution, the model was run using the *RealTime* control machine (with the time unit set to 1s) along with simulated actions and the *FirstComeFirstServedAS* action scheduler [16]. No change was introduced in the agent model. The goal was to check the effects of message processing, which now is no longer negligible, on the scheduling process. Message processing overhead obviously depends also on the performance of the hosting computer machine. Fig. 9 depicts the total consumed power vs. time when the optimal scheduling is used, both in the simulation and the preliminary execution scenarios. The experiments refer to the use of a Macbook Pro Intel Core i5, 2.9GHz, 16GB, OS X El Capitan. A delay clearly emerges in the scheduler reaction due to the algorithm overhead. Such a delay disappears in the case the suboptimal algorithm is adopted (picture not reported for brevity).

In order to highlight the usefulness of the preliminary execution phase, the experiment was repeated also on a less performing (older) Macbook,

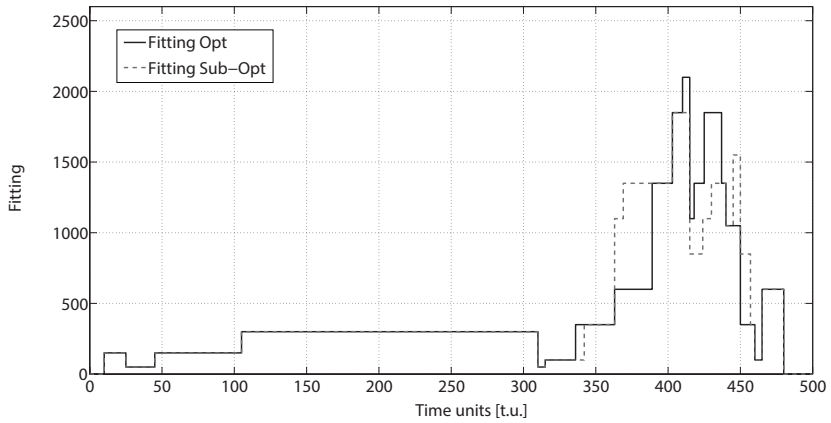

Figure 7: Observed fitting vs. time

Intel Core 2 Duo, 2GHz, 2GB, OS X Mavericks. In this case the delay of the optimal algorithm gets increased (see Fig. 10) as expected. A closer examination of the generated logs reveals that at time 315 a reaction is required but, whereas the simulation is capable of producing an optimal schedule by instantaneously (although ideally) responding to the variation event, the new Macbook employs 10 time units for finding an optimal solution in preliminary execution, and the old Macbook requires more time and it happens that a new variation event is sensed during the scheduler operation which forces it to restart its computation thus missing one reaction.

The worst case of observed *drift*, i.e., the time deviation with which a message is processed with respect to its due time, was found to be about 101ms on the less performing Macbook and about 48ms on the high performing Macbook. The worst case occurs during the initialization/bootstrap of the model. Definitely, the drift tends to be a few ms.

The documented experimental results show that the optimal algorithm, despite computational and communication delays, is capable of managing loads by generating control actions consistent with system dynamics. However, the suboptimal algorithm is preferable because it favors correct temporal dynamics. Its use, in fact, guarantees that the reaction to a variation event is computed and actuated during the same sampling period or, in the worst case, until the next one, if a particular disalignment occurs between the physical clock (of Arduino) and the cyber clock. All these properties were confirmed by considering both the overhead of the scheduler algorithm

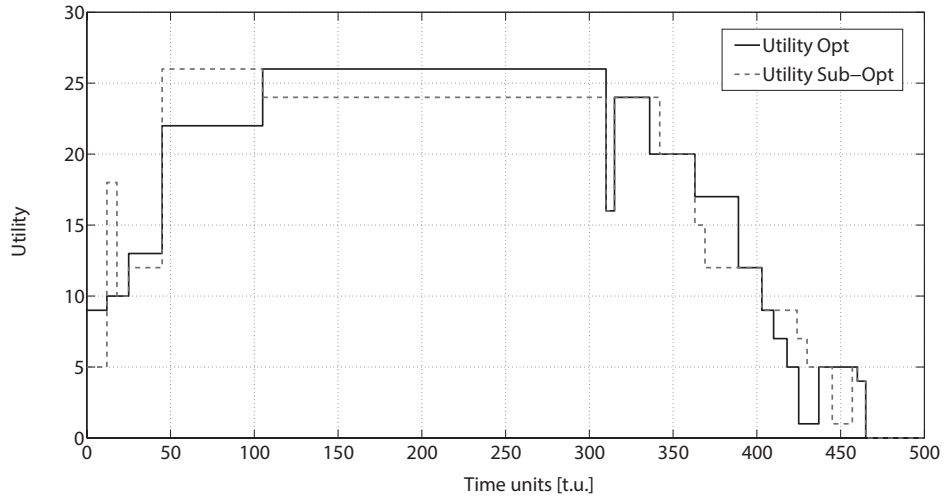

Figure 8: Total utility vs. time

and the bookkeeping of message scheduling and dispatching, also in the case a low performing computer is adopted.

4.5. Prototype implementation and real execution

The CPS power management case study was assembled in the context of an academic electronic measurement laboratory, where loads are realized by lamps of a basic power of 250W. Single or groups of lamps whose power is a multiple of 250W were realized so as to be controlled by few relays. This explains the data assumed in Table 2. Fig. 11 provides an overview of the overall CPS, in which a group of physical loads (lamps) are controlled by corresponding load agents, which in turn receive control commands from the *SchedulerAgent* which is in charge of monitoring the reference input power signal and to adapt the power loads accordingly. The threshold power signal was generated by LabView software and loaded in the memory of an Arbitrary Waveform Generator AWG2021, configured to generate the signal with a frequency of 10Hz. Two Arduino Uno [34] were used as I/O hardware components, with serial communication channels. The first Arduino is devoted to reading the threshold power signal samples. The second one serves to effecting the commands to relays which activate/deactivate the power loads. The prototyped model implementation was executed using the *RealTime* control machine, effective actions and the *envGateway* which provides interactions with real input and output physical devices, controlled by the two Arduino.

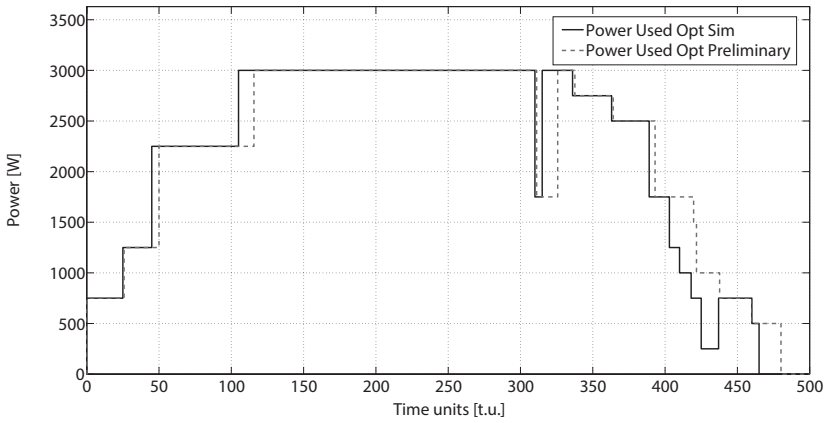

Figure 9: Used power vs. time optimal scheduling, simulation/preliminary execution

Except for an initialization time (due to setting up the Arduino, opening the serial communication channels etc.), which establishes an initial offset of about 10s, the behaviour of the available power and used power is that observed during the preliminary execution. As a final remark, although the case study was driven by pre-configured signals for the generated power and the consuming loads, the agent-based solution is open and flexible to work with, e.g., more dynamic load configurations. This is due to the reactive character of the scheduler which is capable of intervening at each occurrence of a variation event. In the following, a description of the used measurement bench, the load subsystem, and the adopted communication protocol is provided.

4.5.1. Measurement bench

The assembled measurement bench is shown in Fig. 12. It is composed of an Arbitrary Waveform Generator Sony/Tektronix AWG2021, an Arduino One board, a Digital Oscilloscope Tektronix TDS220, and a personal computer. The personal computer interfaces the AWG2021 by a General Purpose Interface Bus (GPIB) and runs a proper program developed in the NI LabView environment. The output channel of the AWG2021 feeds the analog channel 0 of the Arduino board to provide the emulated power signal. Moreover, the marker output signal of the AWG2021 feeds the digital I/O pin 7 of the Arduino One board in order to synchronize the two devices. The TDS220 is connected in parallel to the output channel of the AWG2021 in

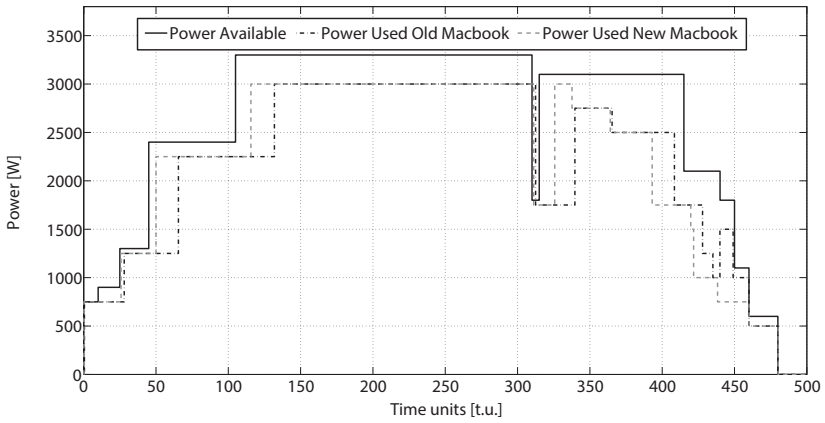

Figure 10: Power available and total consumed power vs. time optimal scheduling, preliminary execution, the two Macbooks

order to visualize the trend of the output signal and then the correct shape of this signal.

4.5.2. Load subsystem

The load subsystem consists of one Arduino One board, 2 relay boards with 8 high voltage channels characterized by rating of 10A at 250 and 125 V AC and 10A at 30 and 28 V DC, managed by means of 8 digital pins adapted to work with the Arduino output operating voltage, and 12 Standard High Pressure Mercury lamps Philips HPL-N 250 W and 12 ballasts. Each ballast is connected to a lamp in order to regulate the current to the lamps and provides sufficient voltage to start the lamp. The loads are obtained by connection of a pre-established number of lamps. The digital pins of the Arduino board feed the digital pins of the relay modules, so permitting to turn on and off the loads.

4.5.3. Communication protocol

The cyber and the physical subsystems interact with each other through the exchange of character strings. Strings can express commands to be executed by the loads, or can capture information that Arduino achieve from sensors, ultimately destined to agents. The following clarifies the adopted format: *sensorId/actuatorId # message*, where the content of the token *message* can vary. Information about sampling the available power signal is

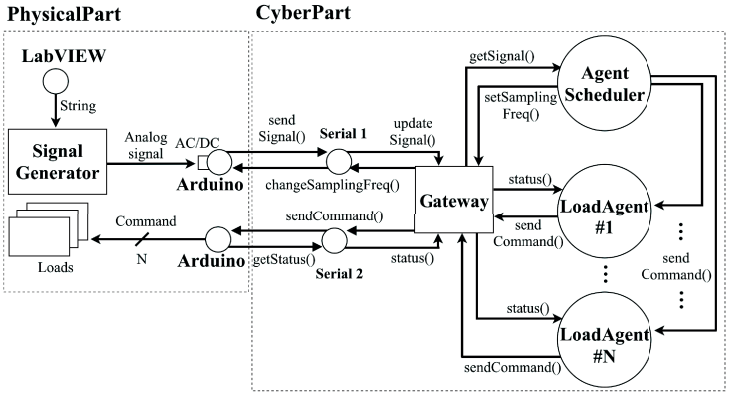

Figure 11: An overview of the realized CPS

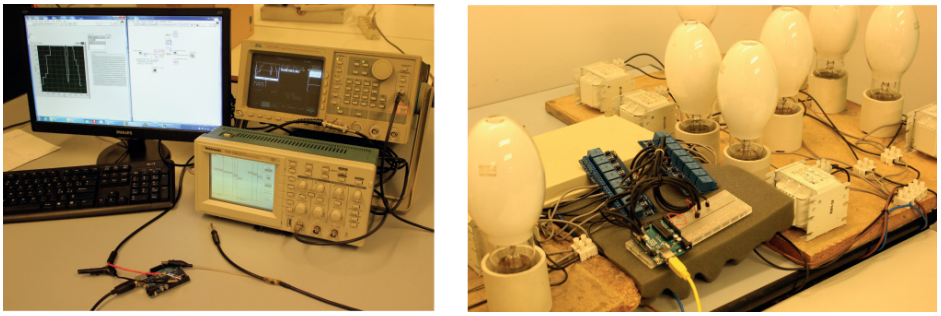

Figure 12: Measurement bench and controlled loads (lamps)

transmitted from Arduino as: *watt # powerLevel* where *powerLevel* is a double number. To change the sampling period the following command can be sent by agents: *arduinoId #samplingTimeInMillis*. To act on a relay it is necessary to send to Arduino the load *id* and the *type* of the command to be executed. The format is: *loadId #command powerLevel* in which the type of the command can be:
⇒ **A**, to activate. It allows to (re)connect a load with a given power level;
⇒ **C**, to change load power. It permits to change the power level of an already connected load;
⇒ **D**, to deactivate. It asks to disconnect a load (in this case no power level is furnished).

5. Conclusions

This paper proposes an agent and control based methodology for the development of Cyber-Physical Systems (CPSs). An original contribution of the approach consists in the exploitation of model continuity [17, 18, 16] in the field of CPSs. For demonstration purposes, the methodology was applied to the design and implementation of a realistic power management case study, thus illustrating how the development of a CPS can benefit from model continuity. Prosecution of the research aims at:

- extending the micro-grid power management case study by covering more complex scenarios, e.g., by experimenting with a scheduler agent based on the imprecise computing paradigm [39]. One such a scheduler could evaluate multiple solutions and in the case no more time is available, the best one among the computed solutions could be adopted. Such a design would ensure a quality of the scheduler decisions which should be better than that of the suboptimal scheduler which rests on the first searched solution, and a little less inferior to that of the optimal scheduler;

- improving/extending the interconnection between the cyber and the physical parts currently based on the mediation of Arduino, with more powerful hardware also in the presence of wireless communications and protocols;

- enhancing the capabilities of the envGateway by offering basic design constructs and mechanisms with the goal of simplifying the modelling and the simulation of the behaviour of more complex physical environment;

- experimenting with the use of the methodology in more general IoT-based applications and for the development of augmented environments like smart homes and smart offices;

- optimizing the agent and control framework by avoiding the recourse to JADE [33] and directly using the Theatre architecture [40]. All of this would improve the efficiency of the execution platform and then the fulfillment of CPS real-time constraints.

- exploiting the available concurrency, which is cooperative for actors and true parallelism for actions, in general GPU based high performance applications.

Acknowledgment

Authors are grateful to colleagues Domenico Grimaldi and Domenico Luca Carní for their help during the physical prototyping of the CPS power management case study in the Laboratory for Processing Measurement Information at the DIMES/UNICAL.

References

[1] E. A. Lee, Cyber physical systems: Design challenges, in: Proceedings of the 2008 11th IEEE Symposium on Object Oriented Real-Time Distributed Computing, ISORC '08, IEEE Computer Society, Washington, DC, USA, 2008, pp. 363–369. doi:10.1109/ISORC.2008.25.

[2] K. D. Kim, P. R. Kumar, Cyber physical systems: A perspective at the centennial, Proceedings of the IEEE 100 (Special Centennial Issue) (2012) 1287–1308. doi:10.1109/JPROC.2012.2189792.

[3] E. A. Lee, The past, present and future of cyber-physical systems: A focus on models, Sensors 15 (3) (2015) 4837–4869.

[4] T. Sanislav, L. Miclea, Cyber-physical systems-concept, challenges and research areas, Journal of Control Engineering and Applied Informatics 14 (2) (2012) 28–33.

[5] M. Broy, M. V. Cengarle, E. Geisberger, Cyber-Physical Systems: Imminent Challenges, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 1–28.

[6] G. Karsai, J. Sztipanovits, Model-integrated development of cyber-physical systems, in: Proceedings of the 6th IFIP WG 10.2 International Workshop on Software Technologies for Embedded and Ubiquitous Systems, SEUS '08, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 46–54.

[7] J. C. Jensen, D. H. Chang, E. A. Lee, A model-based design methodology for cyber-physical systems, in: 2011 7th International Wireless Communications and Mobile Computing Conference, 2011, pp. 1666–1671. doi:10.1109/IWCMC.2011.5982785.

[8] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, Y. Xiong, S. Neuendorffer, Taming heterogeneity - the Ptolemy approach, Proceedings of the IEEE 91 (1) (2003) 127–144.
URL http://chess.eecs.berkeley.edu/pubs/488.html

[9] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry, Challenges for securing cyber physical systems, in: Workshop on Future Directions in Cyber-physical Systems Security, DHS, 2009.

[10] C. Talcott, Cyber-Physical Systems and Events, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 101–115.

[11] M. Wooldridge, An introduction to multi-agent systems, 2nd Edition, John Wiley & Sons, 2009.

[12] G. Rohbogner, U. Hahnel, P. Benoit, S. Fey, Multi-agent systems' asset for smart grid applications., Comput. Sci. Inf. Syst. 10 (4) (2013) 1799–1822. doi:http://dx.doi.org/10.2298/CSIS130224072R.

[13] G. Rohbogner, U. Hahnel, P. Benoit, S. Fey, Multi-agent systems' asset for smart grid applications., Comput. Sci. Inf. Syst. 10 (4) (2013) 1799–1822.

[14] P. Vrba, P. Tichy, V. Mar, K. H. Hall, R. J. Staron, F. P. Maturana, P. Kadera, Rockwell automation's holonic and multiagent control systems compendium, Trans. Sys. Man Cyber Part C 41 (1) (2011) 14–30. doi:10.1109/TSMCC.2010.2055852.
URL http://dx.doi.org/10.1109/TSMCC.2010.2055852.

[15] F. Cicirelli, L. Nigro, Control aspects in multi-agent systems, in: K. J., C. L., M. M. J. (Eds.), Intelligent Agents in Data Intensive Computing, Springer, 2016, pp. 27–50.

[16] F. Cicirelli, L. Nigro, Control centric framework for model continuity in time-dependent multi-agent systems, Concurrency and Computation: Practice and Experience 28 (12) (2016) 3333–3356, cpe.3802. doi:10.1002/cpe.3802.

[17] X. Hu, B. P. Zeigler, Model continuity to support software development for distributed robotic systems: A team formation example, J. Intell. Robotics Syst. 39 (1) (2004) 71–87.

[18] X.Hu, B. Zeigler, A simulation-based virtual environment to study cooperative robotic system 12 (4) (2005) 353–367.

[19] E. A. Lee, The problem with threads, Computer 39 (2006) 33–42. doi:doi.ieeecomputersociety.org/10.1109/MC.2006.180.

[20] H. P. Bernard P. Zeigler, Hessam Sarjoughian, Theory of quantized systems: Devs simulation of perceiving agents, Cybernetics and Systems 31 (6) (2000) 611–647. arXiv:https://doi.org/10.1080/01969720050143175, doi:10.1080/01969720050143175.
URL `https://doi.org/10.1080/01969720050143175`

[21] E. Kofman, S. Junco, Quantized State Systems. A DEVS Approach for Continuous System Simulation, Transactions of SCS 18 (3) (2001) 123–132.
URL `files/qss.pdf`

[22] B. P. Zeigler, H. Praehofer, T. G. Kim, Theory of Modeling and Simulation (second ed.), Academic Press, New York, 2000.

[23] F. Cicirelli, L. Nigro, P. F. Sciammarella, Agents+control: A methodology for CPSs, in: 2016 IEEE/ACM 20th International Symposium on Distributed Simulation and Real Time Applications (DS-RT), 2016, pp. 45–52. doi:10.1109/DS-RT.2016.21.

[24] D. L. Carní, F. Cicirelli, D. Grimaldi, L. Nigro, P. F. Sciammarella, Exploiting model continuity in agent-based cyber-physical systems", in: Advances in Intelligent Systems and Computing, Springer, 2017.

[25] H. J. La, S. D. Kim, A service-based approach to designing cyber physical systems, in: 2010 IEEE/ACIS 9th International Conference on Computer and Information Science, 2010, pp. 895–900. doi:10.1109/ICIS.2010.73.

[26] D. D. Hoang, H.-Y. Paik, C.-K. Kim, Service-oriented middleware architectures for cyber-physical systems, International Journal of Computer Science and Network Security 12 (1) (2012) 79–87.

[27] T. Sanislav, L. Miclea, An agent-oriented approach for cyber-physical system with dependability features, in: Proceedings of 2012 IEEE International Conference on Automation, Quality and Testing, Robotics, 2012, pp. 356–361. doi:10.1109/AQTR.2012.6237732.

[28] S. Deniaud, P. Descamps, V. Hilaire, O. Lamotte, S. Rodriguez, An analysis and prototyping approach for cyber-physical systems, Procedia Computer Science 56 (2015) 520 – 525. doi:http://dx.doi.org/10.1016/j.procs.2015.07.245.
URL http://www.sciencedirect.com/science/article/pii/S1877050915017263

[29] P. Leito, A. W. Colombo, S. Karnouskos, Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges, Computers in Industry 81 (2016) 11 – 25, emerging ICT concepts for smart, safe and sustainable industrial systems. doi:http://doi.org/10.1016/j.compind.2015.08.004.
URL http://www.sciencedirect.com/science/article/pii/S0166361515300348

[30] S. Wang, J. Wan, D. Zhang, D. Li, C. Zhang, Towards smart factory for industry 4.0: a self-organized multi-agent system with big data based feedback and coordination, Computer Networks 101 (2016) 158 – 168, industrial Technologies and Applications for the Internet of Things. doi:http://doi.org/10.1016/j.comnet.2015.12.017.
URL http://www.sciencedirect.com/science/article/pii/S1389128615005046

[31] FIPA, Foundation for intelligent physical agents, http://www.fipa.org.

[32] J. J. Gomez-Sanz, Ten Years of the INGENIAS Methodology, Springer Berlin Heidelberg, Berlin, Heidelberg, 2014, pp. 193–209.

[33] F. L. Bellifemine, G. Caire, D. Greenwood, Developing Multi-Agent Systems with JADE (Wiley Series in Agent Technology), John Wiley & Sons, 2007.

[34] Arduino, web-site.
URL https://www.arduino.cc

[35] R. M. Fujimoto, Parallel and Distribution Simulation Systems, 1st Edition, John Wiley & Sons, Inc., New York, NY, USA, 1999.

[36] S. Abras, S. Ploix, S. Pesty, M. Jacomino, A Multi-agent Home Automation System for Power Management, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008, pp. 59–68.

[37] H. Joumaa, S. Ploix, S. Abras, G. D. Oliveira, A MAS integrated into home automation system, for the resolution of power management problem in smart homes, Energy Procedia 6 (2011) 786 – 794.

[38] F. Cicirelli, D. Grimaldi, A. Furfaro, L. Nigro, F. Pupo, MADAMS: a software architecture for the management of networked measurement services, Computer Standards & Interfaces 28 (4) (2006) 396–411.

[39] W. A. Halang, Load adaptive dynamic scheduling of tasks with hard deadlines useful for industrial applications, Computing 47 (3) (1992) 199–213.

[40] F. Cicirelli, A. Giordano, L. Nigro, Efficient environment management for distributed simulation of large-scale situated multi-agent systems, Concurrency and Computation: Practice and Experience 27 (3) (2015) 610–632.