RA4Self-CPS: A Reference Architecture for Self-Adaptive Cyber-Physical Systems

Marcos Paulo de Oliveira Camargo, Gabriel dos Santos Pereira, Daniel de Almeida, Leandro Apolinário Bento, William Fernandes Dorante, and Frank José Affonso

***Abstract*—Cyber-Physical Systems (CPS) represent an evolution of embedded systems by the computational elements interacting with physical entities through a network. Self-adaptive Cyber-Physical Systems (Self-CPS) present specific features compared to traditional CPS because this type of system can deal with changes at runtime. In parallel, Reference Architectures (RA) enable reusable artifacts that aggregate the knowledge of software architectures in specific domains. RAs have facilitated the development, standardization, and system evolution in different domains. Despite their relevance, reference architectures that could support the more systematic development of Self-CPS, covering issues like self-protecting and observability, are not found yet. Based on this scenario, the main contribution of this paper is to present an RA for Self-CPS named RA4Self-CPS. The goal of this RA is to support the Self-CPS development that requires self-protecting, observability, and adaptation at runtime. To show the viability of our RA, we conducted a case study that revealed a good perspective to contribute to the Self-CPS area.**

Link to graphical and video abstracts, and to code: https://latamt.ieeer9.org/index.php/transactions/article/view/8354

***Index Terms*—Reference architecture, Cyber-Physical Systems, Self-adaptive, Self-CPS.**

I. Introduction

A new paradigm has emerged in the literature whose purpose is to make human-machine interaction possible imperceptibly. In this perspective, several elements of the environment become a valuable source of information so that more accurate reactions of the systems can be interpreted. Among the systems that enable such interaction, Cyber-Physical Systems (CPS) is a system class that can be highlighted because their computational elements also interact with physical entities through a communication network [1], [2], [3].

The current scenario of the software suggests that most parts of software systems must be prepared to operate in normal mode and be available 24/7 (i.e., 24 hours per day and seven days per week). Therefore, features like robustness, reliability, scalability, and customization are required by these systems. These features match with a specific class of software system, which represents the self-adaptive systems. As this term is used in different areas/domains and we will focus only on the software domain, it will be referenced as Self-adaptive Software (SaS). According to Salehie & Tahvildari [4] and Affonso et al. [5], SaS enables the incorporation of new features and/or behavior at runtime (i.e., adaptations without interrupting the execution).

According to Muccini et al. [6] and Musil et al. [7], CPS that deals with some type of uncertainty or unpredictability at runtime can incorporate SaS features in its development, generating a new class of systems known as Self-adaptive Cyber-Physical Systems (Self-CPS). These authors also commented that the MAPE-K loop and specific use of self-* properties have been the starting points for the development of Self-CPS. As CPS are critical systems, the adoption of good engineering practices combined with the segmentation of the adaptation logic into well-defined layers has also been a point to be highlighted in the development of such systems.

From another perspective, Reference Architectures (RA) refer to a special type of software architecture that has become an important element for systematically reusing architectural knowledge [8]. Thus, different domains of software systems, including Self-CPS, have understood the need for encapsulating knowledge (i.e., experiences and good practices) to disseminate and reuse this knowledge in the development of systems. Although the use of RA has relevantly figured in the development of software systems, Self-CPSs have been developed with no concern on the reuse of previously acquired knowledge and experiences, such as best practices, guidelines, and better architectural styles. Based on this scenario, the main goal of this article is to present the RA4Self-CPS, an RA for Self-CPS based on ProSA-RA (Process based on Software Architecture – Reference Architecture) [9]. This architecture was designed based on previous experiences of our research group, namely: (i) an RA called RA4SaS to support the development of SaS [10]; (ii) a framework to support the development of decision support systems for SaS domain [11]; and (iii) an RA named RA4Self-MobApps to support the development of Self-MobApps (Self-adaptive Service-oriented Mobile Applications) [5], [12]. In short, our architecture was organized into three layers, namely: physical environment, cyber, and adaptation. The first represents the physical environment of a CPS. The second represents the software of a CPS, where data collected from the physical environment is captured and processed for decision-making and/or information processing. This layer also contains two important mechanisms for any type of CPS, which are self-protecting and observability. The self-protecting mechanism acts as a protecting strategy for Self-CPS [13]. The observability mechanism aims to present

M. P. O. Camargo, G. S. Pereira, D. Almeida, L. A. Bento, W. F. Dorante, and F. J. Affonso are with São Paulo State University – UNESP, Rio Claro, Brazil (e-mail: mp.camargo@unesp.br, gabriel-santos.pereira@unesp.br, daniel.almeida19@unesp.br, leandro.bento@unesp.br, w.dorante@unesp.br, f.affonso@unesp.br).

to its stakeholders the current state of the system (Self-CPS) through a set of metrics, tracking information, and logs. Finally, the third layer represents the adaptation logic of a Self-CPS in an external approach [4].

Based on the presented context, our architecture aims to consolidate itself as a feasible alternative to facilitate the development of Self-CPS that requires features like service adaptation at runtime, self-protecting, and observability. Thus, we can summarize the main contributions of our RA in three items. From an engineering viewpoint, it enables boosting the software reuse by means of well-defined modules (i.e., module standardization); it makes possible the use of third-party components (e.g., observability solutions); and it enables the developers in their native development environments (e.g., programming language). Regarding the adaptation features, it enables addressing service adaptation at runtime through a dynamic approach of deployment based on previous experience of our research group [12]. Concerning the features of security, it enables us to address the self-protecting of Self-CPS by means of a non-intrusive approach developed in a previous work of our research group [13]. Based on these items, we can create a favorable scenario of development for Self-CPS, since our architecture can agile the development of such systems while enabling the server side to monitor them in the background according to the interest of each system.

The article is organized as follows: Section II presents the background and related work; Section III provides a description of our architecture; Section IV presents a case study to show the applicability of our RA; and Section VI summarizes our conclusions and perspectives for further research.

II. Background and Related Work

In this section we present the background and related work that contributed to the development of our architecture. Concepts of self-adaptive software, cyber-physical systems, and RA are presented in Section II-A. Next, related work on Reference Models (RM) and RA for Self-CPS is addressed in Section II-B.

A. Background

Self-adaptive software. SaS has specific features compared to a traditional one since this type of software must be designed to deal with structural, behavioral, or contextual changes at runtime. Some changes are made to handle complexity and unexpected conditions (e.g., quality degradation), changing priorities and policies governing the goals, and changing conditions (e.g., execution environment) [4]. According to Salehie & Tahvildari [4], "*SaS is expected to fulfill its requirements at runtime in response to changes. To achieve this goal, software should have certain characteristics, known as self-* properties. These properties provide some degree of variability, and consequently, help to overcome deviations from expected goals (e.g., reliability)*". These authors mentioned the 5W1H model as a feasible alternative to elicit the essential requirements of SaS. In short, this model is composed of six questions (i.e., What, Where, Who, When, Why, and How) that need to be answered in both the design and runtime phases. The MAPE-K loop proposed by IBM [14] has been a good alternative to manage the changes at runtime because it enables all decisions to be taken based on a plan established in the data collected from the execution environment. Based on these concepts, a framework to support decision-making in the SaS domain was developed by our research group in previous works [11].

Cyber-physical system. According to the National Science Foundation (NSF), cyber-physical systems (CPS) can be characterized by the interaction between computational elements with physical entities through a communication network. According to Rajkumar et al. [1], CPS may be described as "physical systems, whose operations are monitored, coordinated, controlled and integrated by a computing and communication core". In the view of Li et al. [2], these systems are based on supporting decision-making in real-time, which directly depends on factors such as (i) security, which refers to the guarantee that the impacts of the operations performed are within the desired limit, and (ii) predictability, which is characterized by the system's ability to act in the face of unforeseen conditions through self-adaptation mechanisms. Muccini et al. [6] and Musil et al. [7] presented a study on the presence of CPS in different application domains. For instance, the authors reported the use of CPS in the manufacturing domain (Cyber-Physical Production Systems – CPPS). In summary, these systems (CPPS) can perform autonomous operations in the industry, such as exchanging information, triggering events, and controlling processes independently. Another application domain that has benefited from CPS is the health domain (e.g., Medical Cyber-Physical Systems – MCPS). In short, these systems (MCPS) interact with the physical environment (i.e., patient monitoring mechanisms) so that decisions can be taken. Due to its critical nature, it is worth mentioning that MCPS requires special attention in relation to quality attributes (for example, response time, security, and reliability).

The concept of Digital Twins (DT) is related to the CPS integration. A DT can simulate the behaviors of physical objects through high-fidelity virtual models in virtual space, offering insights into their real-world performance [15]. According to Moyne et al. [16], a virtual model uses data to remain synchronized with its physical counterpart. These capabilities and scope are limited to a predefined purpose and application environment. DTs empower organizations to anticipate and identify physical issues with greater speed and precision, optimizing manufacturing processes, enhancing product quality (e.g., agriculture), boosting medical monitoring, and optimizing autonomous automobile systems [17].

Another element to be highlighted in the CPS context is the collaborative aspect, which can associate a value to the system (i.e., (Self-)CPS) coming from the collaboration of the system components among themselves, and even from the collaboration between these components and humans, enabling the ideal level of abstraction for the systems [18]. This combination resulted in the concept of Collaborative CPS (CCPS), which was defined as "*a system containing interconnected, tightly coupled physical (hardware) and cyber (software) components with defined borders, allowing identifying the inner and outer components as well as inputs and outputs, jointly acting and* *sharing information, resources and responsibilities in order to achieve a common goal, and operating over a given period of time*" [19].

Reference architecture. RA is a special type of architecture that provides a set of guidelines for the specification of concrete architectures. In this sense, guidelines and processes have been proposed in order to systematize the design of such architectures [20], [21]. To Kruchten [22], "*RA is, in essence, a predefined architectural pattern, or set of patterns, possibly partially or completely instantiated, designed and proven for use in particular business and technical contexts, together with supporting artifacts to enable their use. Often, these artifacts are harvested from previous projects*". In Bass et al. [8], "*RA is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them*". Therefore, it can note that the effective knowledge reuse of RAs depends not only on raising the domain knowledge but also documenting and communicating efficiently this knowledge through an adequate architectural description. Considering the relevance of this research topic as the basis of software development, a diversity of architectures has been proposed and used, including self-* software (e.g., FORMS [23] RA4SaS [10], RA4Self-MobApps [5], among others).

B. Related Work

As related work, we conducted a Systematic Mapping Study (SMS) to identify studies on reference models and reference architectures for Self-CPS [24]. The main purpose of this SMS was to identify primary studies that report reference models and/or reference architectures for the Self-CPS domain. This mapping also aimed to get a comprehensive overview of the features of these models and/or architectures. In this sense, we identified the application domains that have proposed models/architectures for Self-CPS, the quality attributes adopted in the design of such models and/or architectures, besides different aspects related to design and (self-)adaptation. Next, an overview of this mapping is addressed based on 12 studies presented in Table I.

The first column shows the ID for each study, and the second one the reference for each study. Each reference is composed of the publication title with an external link where the primary study was published, besides its respective reference after the title. The third column shows the type of each study (i.e., RM – Reference Model and RA – Reference Architecture). In the fourth column are presented the application domains identified in our mapping. An overview of the techniques used to evaluate each study is presented in the fifth column. The sixth column shows the quality attributes (QA) identified in each study from our mapping. The target of monitoring adopted in each study is presented in the seventh column. Finally, the eighth column shows the evidence identified in each study concerning the adaptation features.

Of the 12 studies, 10 were classified as RA and two as RM, revealing that most studies proposed more comprehensive solutions from the design viewpoint. "Industry 4.0" with four studies, "Automotive" with three studies, "Urban" with one study, and "Unmanned Aerial Vehicle" with one study were the application domains identified in our mapping. Three studies were classified as "General" because they did not provide concrete evidence about the application domain. Although such studies have been classified into different domains, we found some similarities in the elements of the models and architectures proposed as CPS with adaptation features. "Case study" with three studies and "Prototype" development were the most used techniques for the evaluation of the models and/or architectures. Quality was another aspect investigated in our mapping and the evidence showed us that there is no prominent attribute. Reliability, performance, interoperability, and security were the most used attributes. The evidence reveals that the seven attributes identified were used according to the purpose of each model and/or architecture.

Regarding the design of the models and/or architectures, the evidence of our mapping revealed that we organized them according to the approach selected to deal with software adaptation. In addition, the monitoring mechanisms in such models/architecture must be highlighted, since they enable perception of the physical environment. Monitoring focused on quality aspects was found in three studies because the application domain of these studies requires special attention to communication and/or performance degradation for task execution.

With regard to adaptation interests, two issues were investigated. The **first** focused on understanding how the Self-CPS has dealt with uncertainties, which can be internal or external. In this sense, it is worth highlighting the use of the MAPE-K loop as a solid alternative from the self-* community to deal with adaptation concerns. Our mapping data also reveal that the self-organizing property (S5, S6, S7, S10, and S11) was identified in five of the seven studies in which it was possible to evidence a self-* property. The **second** focused on gathering evidence on the Artificial Intelligence Techniques (AIT) used in the design of such models/architecture so that these systems (Self-CPS) can handle adaptation at runtime. "Supervisor system", Learning techniques, and "Intelligent agents" were AIT techniques identified in our mapping capable of dealing with knowledge acquisition and decision-making to promote some type of modification in the Self-CPS.

As reported in this section, Self-CPS design is a complex issue that requires knowledge from different areas. Therefore, designing reference architectures that can serve as a guide for the development of this type of system is still an open issue and of interest to different scientific communities and practitioners. To the best of our knowledge, there is no reference architecture for Self-CPS that can handle adaptation, protecting, and monitoring concerns simultaneously. To do so, this architecture must be flexible and scalar in relation to the design so that new threats/vulnerabilities can be incorporated into a self-protecting module without interfering with the other interests of the application. This feature must be valid to other modules of this architecture like observability concerns (i.e.,

TABLE I
LIST OF PRIMARY STUDIES ANALYZED IN THE MAPPING

ID	Reference	Type	Domain	Evaluation	QA	Monitoring	Adaptation
S1	A Conceptual Reference Model for Human as a Service Provider in Cyber Physical Systems [25]	RM	General	Case study, Metric Analysis	Reliability	Physical environment	MAPE-K
S2	Analysis of autonomous deconfliction in Unmanned Aircraft Systems for Testing and Evaluation [26]	RM	Unmanned Aerial Vehicle	Not evaluated	Security	-	self-* property
S3	An architecture for context-aware reactive systems based on run-time semantic models [27]	RA	General	Case study, Prototype	Interoperability, Reliability	-	MAPE-K
S4	A reference architecture for cooperative driving [28]	RA	Automotive	System example	Performance	Quality aspect	Supervisor systems
S5	BIOSOARM: a bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors [29]	RA	Industry 4.0	Test scenario	Performance	-	Learning techniques, self-* property
S6	Engineering self-organizing urban superorganisms [30]	RA	Urban	Not evaluated	Modularity	-	self-* property
S7	Industrial Dataspace: A Broker to Run Cyber-Physical-Social Production System in Level of Machining Workshops [31]	RA	Industry 4.0	Prototype	-	-	self-* property
S8	Integration of digital twin and deep learning in cyber-physical systems: Towards smart manufacturing [32]	RA	Industry 4.0	Case study	Resiliency, Performance, Reliability	Quality aspect	Learning techniques, MAPE-K
S9	Risk Assessment for Cooperative Automated Driving [33]	RA	Automotive	Not evaluated	-	-	self-* property
S10	Tools and Methodologies for Autonomous Driving Systems [34]	RA	Automotive	Not evaluated	Compatibility, Reliability, Security	-	self-* property
S11	Towards a cloud-assisted and agent-oriented architecture for the Internet of Things [35]	RA	General	Not evaluated	-	-	Intelligent agents, self-* property
S12	Towards Cyber-Physical Infrastructure as-a-Service (CPIaaS) in the Era of Industry 4.0 [36]	RA	Industry 4.0	Not evaluated	Resiliency, Interoperability, Reliability	Physical environment Quality aspect	-

metrics, trace information, and logs)[1] that can support the monitoring activity.

III. Reference Architecture for Self-CPS

RA4Self-CPS is a reference architecture that aims to support the development of Self-CPS, whose main features are observability, self-protecting, and adaptation at runtime without the perception of the stakeholders. We used a process to build reference architectures named ProSA-RA [9] and an approach proposed by Tummers et al. [38] to establish our architecture. In short, we selected and investigated the information sources in Step RA-1. Next, we identified the architectural requirements in Step RA-2 so that an architectural description of the RA4Self-CPS is established in Step RA-3. Finally, we evaluated our architecture in Step RA-4. We addressed the details of each step in Sections III-A to III-D.

[1] According to Tozzi [37], metrics can be defined as a logical meter (i.e., a counter or histogram over a period). Trace information deals with request-scoped data (i.e., any data or metadata associated with the life cycle of a single transactional object in a system). Logs deal with discrete events that occur while running a system, such as error messages, audit events, or request-specific metadata.

A. Step RA-1: Information Source Investigation

In this step, the main sources of information were listed to identify the requirements related to the RA4Self-CPS design. In summary, we grouped these sources of information in two sets, namely: (i) guidelines for the development of Self-CPS, and (ii) reference models and reference architectures for Self-CPS. The first set aimed to gather the key concepts and provide evidence on good practices used in the development of Self-CPS. The second set aimed to synthesize the main evidence on the design of architectures and/or models for Self-CPS, where it was possible to identify the mandatory functionalities and the similarities between the models and/or architectures. Next, details of each set are presented.

Set 1 – Guidelines for the development of Self-CPS. In this set, we grouped concepts and information that act as a support for the development of SaS and CPS, generating the so-called Self-CPS. As reported in Section II-A, SaS is a special type of software system because enables adaptation at runtime. Thus, in order to elicit the requirements for the RA4Self-CPS design regarding adaptation, we used the 5W1H model based on the previous experience of our research group [5], [10]. To do so, we have used the following questions: (i) ***What** will be adapted?* (e.g., CPS); (ii) ***Where** will the* *adaptation occur?* (e.g., software architectures); (iii) ***Who** will perform the adaptation?* (e.g. supervisor systems); (iv) ***When** will the adaptation be applied?* (e.g., how often); (v) ***Why** will it be adapted?* (e.g., low QoS); and (vi) ***How** will it be adapted?* (e.g., actions plan).

Regarding the state of the art on Self-CPS, we conducted a literature mapping (see Section II-B). Based on the studies selected in this mapping, important information was extracted, namely: (i) the main domains involving (Self-)CPS, (ii) the main approaches adopted to enable self-adaptation, (iii) the main design patterns used in (Self-)CPS, (iv) the main concerns addressed by the adaptation, and (v) the main difficulties encountered when implementing adaptation properties [39]. This information forms an important knowledge base for the design of our architecture. As reported in Section II-B (see Table I), 12 studies reported having designed models and/or reference architecture for Self-CPS. Among the evidence identified in these models and/or architectures, we can highlight the following features: modular organization, observability features, monitoring (e.g., QoS degradation), data parsing between software layers of this type of system, external approach to adaptation, and security. Although such features are relevant for the design of a Self-CPS, we always partly identify them. Therefore, it can be said that of the initiatives found in the literature, none of these models/architectures address the self-protecting, observability, and adaptation at runtime of the Self-CPS as our RA.

Set 2 – Reference Architectures for Self-CPS. In this set, we grouped and studied the main reference models and reference architectures for Self-CPS. To do so, we conducted a systematic mapping (see Section II-B), which resulted in 12 primary studies distributed between January 1, 2009 and September 17, 2021. The results of this mapping enable us to establish an overview and a more solid understanding of how the reference models and/or architectures were designed in recent years. Thus, we highlighted important indicators, such as the current stage of the Self-CPS, research gaps and perspectives for future work. Therefore, it can be said that this overview/understanding was important to establish the architectural requirements of our architecture.

B. Step RA-2: Architectural Requirement Establishment

Based on the information identified in Step RA-1, we identified the requirements for the proposed reference architecture. In this step, we organized the requirements into two categories, namely: (i) Cyber-Physical Requirements (CPR), which are related to the overall purposes of the (Self-)CPS, (ii) ADaptation Requirements (ADR). Next, a description of each requirement is addressed:

- **[CPR-1]** The RA should provide a monitoring mechanism capable of identifying and notifying system stakeholders of anomalous requests or unwanted behavior (e.g., execution environment);
- **[CPR-2]** The RA must enable the user the option of storing a state history of the application environment and/or software components;
- **[CPR-3]** The RA must provide a dashboard to visualize the status of the execution environment (metrics, traceability, and logs);
- **[CPR-4]** The RA must enable a Self-CPS to be aware of the context in which it is inserted;
- **[CPR-5]** The RA must provide a mechanism for exchanging data between the execution environment and the decision-making modules;
- **[ADR-1]** The RA must identify and overcome an adverse situation (e.g., quality degradation) at runtime;
- **[ADR-2]** The RA must enable the replacement of a software entity or service at runtime;
- **[ADR-3]** The RA must provide a mechanism for identifying and reporting a problem when an adaptation operation has been performed;
- **[ADR-4]** The RA must allow elaborating an adaptation plan so that a Self-CPS can deal with unforeseen situations at runtime (structure, behavior, and context).
- **[AR-5]** The RA must alert interested parties about an unavoidable situation. From this scenario, it is understood that the system can transition to an irreversible scenario that requires human intervention to return to the normal operating state.
- **[AR-6]** The RA must enable monitoring of the Self-CPS to assess its operating conditions (e.g., quality of service).

Besides standards and reference architectures, other sources of information were also analyzed so that the list of requirements was generated: (i) SaS development and its adaptation engine [4]; (ii) SaS developers with experience designing and implementing SaS [40], and (iii) example systems [25], [41], [42]. Regarding the last item, it is worth mentioning that the aforementioned authors developed such systems for different domains and provided excellent contributions (e.g., adaptation scenarios, architecture requirements, end-user needs, among other resources) for the reference architecture design proposed in this article.

C. Step RA-3: Architectural Design

As mentioned in Section III, our architecture was designed based on a combination of the PROSA-RA process guidelines and the approach proposed by Tummers et al. [38], which provides a process for building and/or reusing modules in reference architectures for the information systems domain. Based on this methodological strategy, RA4Self-CPS was organized in layers according to a top-down view. To do so, we used the information sources (i.e., Sets 1 and 2) identified in Section III-A and the architectural requirements established in Section III-B. Fig. 1 illustrates an overview of RA4Self-CPS, where it is possible to observe the following layers: adaptation, cyber, and physical. The first represents the logic to deal with the self-adaptation of the system at runtime. The second, represented by the dotted line, represents the software communication layer with the physical layer (third layer). Regarding the internal elements of these layers, we recommended that some modules be designed. For instance, so that a CPS software can modify its structure and/or behavior at runtime or even adjust to context scenarios (i.e., execution environment), the **adaptation layer** has three modules: metamodel, control loop, and adaptation mechanism. The **cyber layer** contains all the communication logic between the CPS and the **physical layer**, providing modules for service design, communication between CPS components, and routing. Because it is a layer that deals with distributed systems via services, this layer also provides for the elaboration of modules for monitoring services and protecting them via HTTP (Hypertext Transfer Protocol) requests. Finally, the **physical layer** is responsible for enabling communication between the **physical environment** and the CPS through sensors and effectors. The **physical environment** represents the physical side of a CPS. Next, a description of the general purpose of each layer is presented.

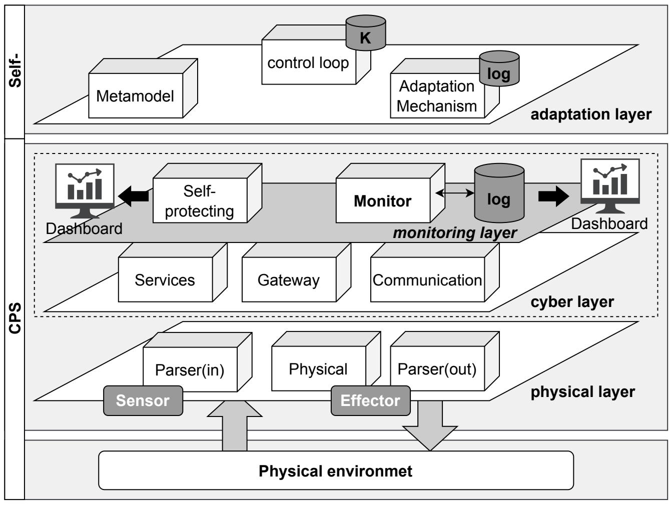

Fig. 1. Overview of RA4Self-CPS

Physical environment. Based on the concepts and definitions reported in Section II-A, it can be said that the operating environment of a (Self-)CPS can meet different application domains. As each application has specific requirements regarding the number of equipment and sensors (among other elements), the definition of the operational environment must be in accordance (and may vary) according to each application. Therefore, this layer represents the physical components of a Self-CPS that are used to monitor and control the physical processes seamlessly.

Physical layer. This layer aims to enable the interaction between the layer that represents the physical environment (**physical environment** layer) and the **cyber layer** of a Self-CPS, as illustrated in Fig. 2. The physical and logical sensors represent the hardware and software components that are part of a Self-CPS, respectively. Effectors represent all hardware components that can interact and make modifications in the physical environment. Next, a description of the components of the **physical layer** is presented.

Sensors components capture data from the operational environment, and ***Effectors*** forward the data to this environment. The ***Controller*** component receives the data from the sensors and, after activities of execution of a Self-CPS, returns the processed data to the execution environment. The ***Parser(in)*** and ***Parser(out)*** modules translate data between the **physical**

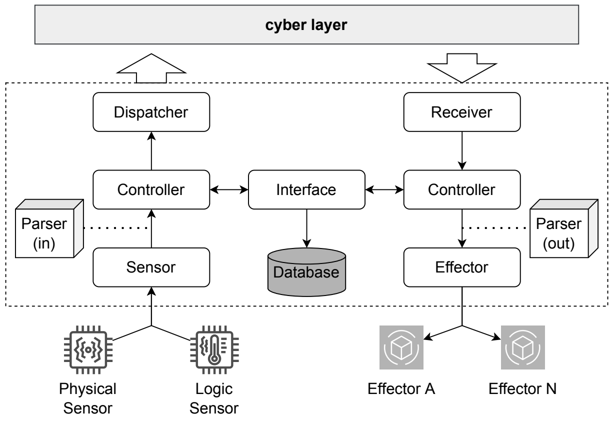

Fig. 2. Physical layer components

environment layer and the other layers of the architecture. This operation is performed to make the data collected from the environment compatible with the format required by artificial intelligence algorithms. The ***Interface*** component works in parallel with the ***Controller*** component, storing data collected from the physical environment via sensors and actuators in a ***Database*** component. This data is used by upper-layer systems to manage a Self-CPS at runtime. The **Dispatcher** component aims to provide the data collected from the runtime environment to the **cyber layer** via Web services. The ***Receiver*** component receives the processed data from the upper layers and forwards it to the components of the **physical layer** to be sent to the operational environment (**physical environment layer**).

Cyber layer. This layer is represented by the intermediate layers (i.e., **cyber** and **monitoring**) and enables the communication between the **physical layer** and the software of a Self-CPS, as illustrated in Fig. 3. As reported in this section, the purpose of the *physical layer* is to deliver the data collected from the execution environment via the ***Dispatcher*** component (see Fig. 2) to be processed by the **cyber layer** in the right format. Once processed, this data is returned to the **physical layer** through the ***Receiver*** component. During the processing step, changes such as anomaly detection and/or quality degradation can be detected in the application (Self-CPS) by the **adaptation layer**. Such changes must be analyzed and corrected so that damage or interruptions in this application are not generated. Next, a description of the components of the **cyber layer** is presented.

The ***Communication*** component provides the interfaces for the services (i.e., concrete implementation) that will perform the processing of an operation. In this sense, three types of communication are illustrated, namely: (i) Endpoint for communication via REST services; (ii) WDSL (Web Service Description Language) for communication via SOAP services; and (iii) API (Application Programming Interface), which represents a generic communication type. It is worth mentioning that these interfaces must be organized in a service catalog to be consulted by the other RA4Self-CPS components.

The concrete implementation of the services used by Self-CPS is represented by the ***Service*** component. To enable self-

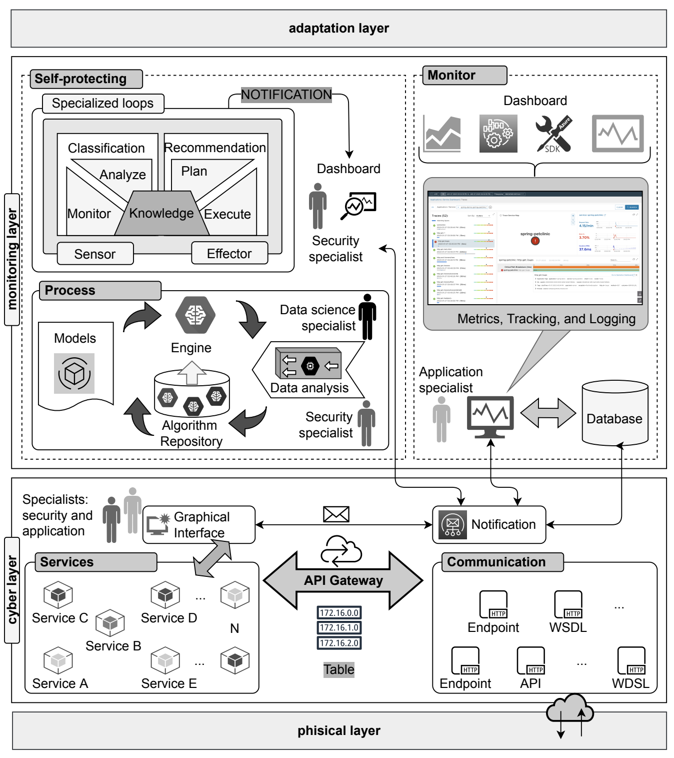

Fig. 3. Cyber layer components

adaptation in a Self-CPS regarding services, previous experiences of our research group can be used. In this sense, it is worth highlighting the framework [12], [43] and the reference architecture [5] that enables dealing with adaptation of REST and SOAP service at runtime. In short, the software engineer must define the primary service and the set of alternative services for this primary service. Thus, in case any type of anomaly (e.g., failure or quality degradation) is identified during the execution of a primary service, an alternative one takes over running without the user's perception.

The ***API Gateway*** component aims to facilitate communication/interaction between the services that are part of a Self-CPS. To do so, this component must be available at a different address for the target service, making it possible to replace/relocate the target service without changing the details of the associated gateway service. In addition, this component must provide a common interface for the services, in which the interested parties do not need to know where each underlying service is located (e.g., internal or external service to the Self-CPS operational environment).

As a Self-CPS can be complex and critical, monitoring the state of execution and security of this type of system has been an aspect identified in our mapping (see Section II – related work). Thus, we created a supervisory layer for the **cyber layer** called **monitoring**, which is composed of two modules: Self-protecting and Monitor. Next, a description of each module is addressed.

The ***Self-protecting module*** (see Fig. 3 – dotted line) aims to analyze all HTTP requests from a Self-CPS and identify whether they represent any type of threat/vulnerability. To do so, we used an approach proposed by Martins [44], which enables self-protecting at the application layer in Web and service-oriented applications. In synthesis, this approach enables flexibility and scalability in relation to the development of models for anomaly classification. In addition, it is worth noting that other security alternatives can be used in the architecture proposed in this article, besides other security levels that can also be inserted in this architecture.

The ***Monitor module*** aims to present interested parties (i.e., security and application specialists) in the current state of the system (Self-CPS) through a set of metrics, trace information, and logs. In short, this information can be useful for identifying any type of anomaly that could compromise the system's operation (Self-CPS). To do so, we can adopt a set of existing tools in the literature to deal with monitoring an application that goes beyond the collection of observability information. These tools can interpret and make sense of the data tracked and monitored and the created relationships between various data sources. Finally, this module should enable the interaction between the interested parties and the Self-CPS, enabling modification in behaviors and/or awareness of the monitored environment.

Adaptation layer. This layer must meet the adaptation interests of a Self-CPS at runtime. Regarding adaptation, two considerations should be highlighted. First, software that requires runtime modification can be designed to satisfy one (or more) self-* properties. As the number of these properties increases according to the application domain, the complexity of development also increases by an equivalent or even greater proportion. Second, the control loop proposed by IBM [14] has been adopted by the SaS community to address adaptation concerns (see Section II-A). In previous experiments by our research group [11], this loop was organized into two modules to deal with SaS adaptation interests, namely: classification and recommendation. Fig. 4 illustrates the organization of the **adaptation layer** for our architecture, based on the MAPE-K loop and on the aforementioned organization. In short, in the ***Process phase*** occur the elaboration of the classification and recommendation models that will be used by the loops (***Loop phase***). Next, a description of each phase is addressed.

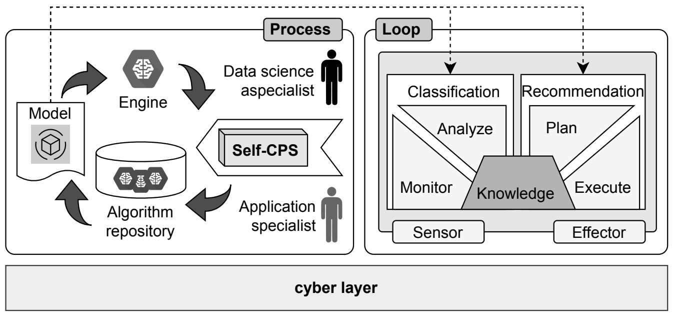

Fig. 4. Adaptation layer components

In the ***Process phase***, specialists must create the classification and recommendation models, which will deal with the adaptation interests of a Self-CPS at runtime. In short, the classification module must be responsible for identifying some type of anomaly/adversity in a Self-CPS. On the other hand, the recommendation module must be able to establish a modification plan so that such anomalies/adversities do not harm or interrupt the execution of the system (Self-CPS). To design these models, the Self-CPS specialist must define which modifications this system must support and which self-* properties will be used so that the modifications at runtime be performed. To do so, this specialist can use the 5W1H model as a mechanism to guide adaptation interests in relation to the target software and its modifications. Finally, the ***Loop phase*** represents the instance of a control loop in the execution environment, based on the classification and recommendation models elaborated in the ***Process phase***.

As reported in this section, the description presented for each module/component can guide the stakeholders when our architecture is instantiated in a concrete architecture. In this sense, we can highlight three considerations in relation to such modules/components, namely: (i) ***Service*** component – Fig. 3: we cite in the description of this component that we used a previous work of our research group [12]. However, it is worth highlighting that we can use any solution that can perform service adaptation at runtime; (ii) ***API Gateway* component**– Fig. 3: Similar to the previous component, we can use other types of solutions that enable communication between services [45]; and (iii) ***Self-protecting module*** (see Fig. 3: we also cite in the description of this module that we used a previous work of our research group [13] based on self-protecting. In this sense, we can use similar or complementary solutions in this module to provide protection at runtime.

D. Step RA-4: Reference Architecture Evaluation

We conducted an inspection based on the checklist in order to improve the quality of RA4Self-CPS. The main purpose of this checklist is to verify if there are defects related to omission, ambiguity, inconsistency, and incorrect information that can be present in our architecture. Moreover, aiming at observing the viability of RA4Self-CPS, as well as its capability to develop SaS, case studies were conducted, and the results are presented in the Section IV.

IV. Case Study

This section presents a case study we have conducted in order to evaluate the applicability, strengths, and weaknesses of our architecture (RA4Self-CPS). As subject application for our empirical analysis, we have selected an application addressed to the Smart Home HealthCare (SHHC), as illustrated in Fig. 5. Next, we presented a brief description of our subject application and the empirical strategies adopted for conducting this case study.

The SHHC project represents a domestic environment composed of basic infrastructure to monitor the health of a person with some type of illness in a simulated computing environment. As illustrated in Fig. 5, this project has the following elements: `SHHCSensorAPI`, `ControlPanel`, `ContainerApp`, `MonitorApp`, `NotifierApp`, and `Observability`. Besides the software elements, this project has the following actors: patient, nurse, doctor, ambulance, and hospital. Next, a brief description of the elements/actors is presented.

The patient actor is represented by the `SHHCSensorAPI` application, where each application was instantiated in a Docker[2] container. In short, each patient simulates monitoring the following sensors: airflow, blood pressure, glucose, heart rate, pulse oxygen, and temperature. These sensors represent the physical environment of a Self-CPS. To enable communication with the other elements of the SHHC project, these sensors were implemented via API via the Spring Boot[3] framework.

The SHHC specialist can adjust (i.e., increase or decrease) the values of the sensors supposedly connected to the patient through `ControlPanel`, an application developed in JavaFX[4] integrated to `SHHCSensorAPI`. It is worth mentioning here that this application does not include a simulation based on disease scenarios. For instance, a patient who has a symptom of fever may have an altered heart rate, besides other health alterations. In addition, other factors must be considered modeling patient scenarios, such as age, weight, height, BMI (Body Mass Index), among others.

`MonitorApp` is an application developed in JavaFX integrated with `SHHCSensorAPI`, which displays a graph for each sensor. To do so, this application consumes the current values of the sensors through REST services in a time interval of 5 seconds. This application acts as a dashboard that can be used by the nurses to observe the status of each patient's sensors. In addition, `MonitorApp` was integrated with `NotifierApp`, whose purpose is to send notifications to smartphones (nurses and doctors) of situations that require urgent treatment. In summary, the nurse receives a notification via mobile application so that a procedure to overcome the health problem identified by the system is started. Based on this scenario, the systems (`MonitorApp` and `NotifierApp`) must await the completion of this procedure to notify it again. If the number of attempts (nurses and doctors) exceeds the limit allowed by the system, the emergency service can be triggered by the `NotifierApp application`. With regard to implementation, the `NotifierApp` application receives data from the sensors and classifies the patient's status through a set of rules. We opted for the framework DROOLS[5] based on previous experience of our research group. Finally, smartphones (nurses) and systems (doctors) are hardware elements that contain applications that receive information from the `NotifierApp` application via Websocket[6].

`ContainerApp` is a Java application designed to interrupt and initialize the `SHHCSensorAPI` application containers to simulate failure scenarios in the sensors. These operations are captured by the monitoring module and displayed on administrators' dashboards. For instance, observability represents a set of information (metrics, tracing, and logs) that can be collected from services at runtime. The significant increase in requests for a service and/or the log of command injection (e.g., SQLi attack) captured by data analyzers are examples of

[2]https://www.docker.com/
[3]https://spring.io/projects/spring-boot
[4]https://openjfx.io/
[5]https://www.drools.org/
[6]https://spring.io/guides/gs/messaging-stomp-websocket

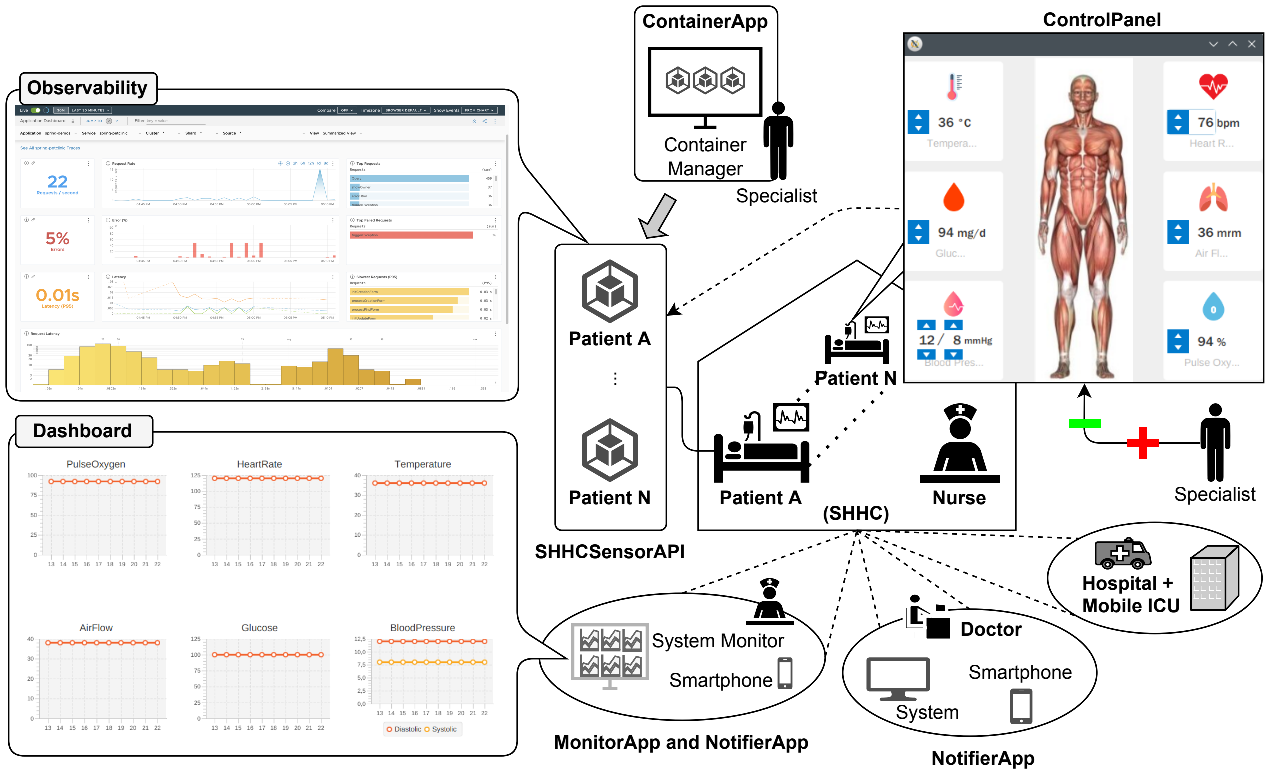

Fig. 5. General architecture for the SHHC

actions that can be malicious for a Self-CPS. To do so, we used Wavefront[7] application for the ease of integration with REST APIs. Besides observability, a self-protecting approach [13], [44] was instantiated to help identify anomalous requests to a Self-CPS. To couple this approach to the `SHHCSensorAPI` module, it is necessary to include the dependency in the `pom.xml` file, as shown in Listing 1.

Listing 1: Source code of the `pom.xml` file

```
<dependency>
   <groupId>br.unesp.rc</groupId>
   <artifactId>LibSelfProtection</artifactId>
   <version>1.0-SNAPSHOT</version>
</dependency>
```

Next, we must configure the `application.properties` file of the `SHHCSensorAPI` application so that the approach works, as shown in Listing 2. We enable the security process in Line 4, and we define the port and URL settings in Lines 5 and 6. Finally, we created an identifier for the application to be managed in Line 7(i.e., `SHHCSensorAPI`).

Listing 2: Source code of the `application.properties` file

```
1  server.port=8084
2  spring.data.mongodb.
3                          ignore-unknown-exceptions=true
4  mape.enable=true
5  mape.port=8081
6  mape.url=http://localhost
7  appmanaged.id=63152073fe5ad305865bc6c9
```

[7]https://docs.wavefront.com/index.html

After the configuration step, we must create a filter in the `SHHCSensorAPI` module so that all *headers* and request data can be captured. Listing 3 shows the source code of this filter. In short, between Lines 6 and 13 the attributes for this filter are defined, which are the information defined in the `application.properties` file. In Line 19, it is verified whether security is enabled in the application so that the monitoring point can be instantiated in Line 20 and put into execution in Line 23. As seen, all requests (`wrappedRequest`) from an application (`appID`) are monitored by the approach. Regarding monitoring, the security specialist visualizes all requests in a dashboard (i.e., a request is normal or anomalous). When an anomalous request is identified, the approach shows a classification for the attack that the application is being subjected to.

Listing 3: Source code of the `CustomFilter.java` class

```
1  package br.unesp.rc.shhc.SHHCSensorAPI.filter;
2  // ...
3  @Configuration
4  @Order(1)
5  public class CustomFilter implements Filter {
6     @Value("${appmanaged.id}")
7     private String appID;
8     @Value("${mape.enable}")
9     private boolean mapeEnable;
10    @Value("${mape.port}")
11    private String mapePort;
```

```
    @Value("${mape.url}")
    private String mapeURL;

    @Override
    public void doFilter(...) throws IOException,
        ↪ ServletException {
      try {
        //...
        if (mapeEnable) {
          MonitoringPointRequest mpr =
          MonitoringPointRequest.
          getInstance(mapeURL, mapePort);
          mpr.run(wrappedRequest, appID);
        }
      } //...
    }
    //..
}
```

To illustrate adaptation scenarios in this case study, we must first show the new organization of the SHHCSensorAPI application, as illustrated in Fig. 6. Instead of this application representing a patient with sensors on different routes, this application now has a central element (`Patient`) that controls/coordinates calls to the other sensors (e.g., `Temperature`). It is worth highlighting that we implemented each sensor as an isolated application in a container. To illustrate the service adaptation, we considered two scenarios of adaptation, namely: recovery and evolution.

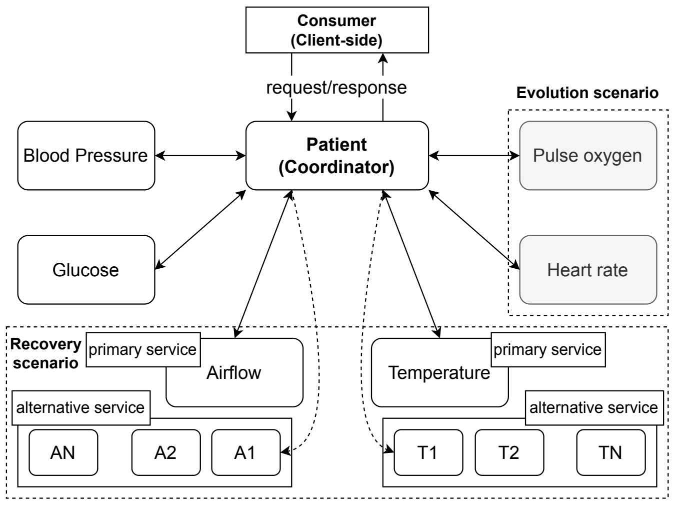

Fig. 6. Adaptation scenarios

The first aims to deal with the system's recovery in relation to the occurrence of failure or degradation of service quality. To do so, a failure in the `Temperature` sensor (`TemperatureAPI`) application) will be simulated so that an alternative service (`T1`) will replace a primary service named `Temperature`. The dotted line area named **Recovery Scenario** illustrates this scenario for the `Temperature` and `Airflow` sensors.

The second explores the evolution of the system in relation to a new need of its users. To do so, we will simulate the SHHCSensorAPI application with four sensors (`Bood Pressure`, `Glucose`, `Airflow`, and `Temperature`). Depending on the patient's health status, we can add new sensors to improve the system's ability to monitor the patient. The dotted line area named **Evolution Scenario** illustrates this scenario through the addition of two sensors (`Pulse Oxygen` and `Heart rate`).

We have not reported details of the service adaptation (i.e., source code) at runtime in this section because of the complexity and space reasons required by the framework instantiation. Such details can be found at Passini & Affonso [12] and Affonso et al. [5]. Finally, it is worth mentioning that other technical elements (e.g., Gateway) of our architecture were not described in this section for reasons of scope.

V. Discussion of Results and Limitations

This section summarizes the main findings and discusses the relevance of this work to the RA, Software Engineering, and CPS communities. The main findings and results are listed as follows.

This article proposed the RA4Self-CPS, to the best of our knowledge, the first reference architecture that encompasses three activities at the same time: self-adaptation, self-protecting, and observability through a well-defined modular organization. We can execute the first two activities without the perception of the stakeholders, as well as be supported by specialists through dashboards. The third is a mandatory activity, which must be monitored by the specialists in the application through information extraction tools. These tools can guide the specialist's decision-making depending on any need to change the Sel-CPS.

As reported in Section III, we designed our architecture (RA4Self-CPS) based on a well-defined modular organization, which enables architecture modules, when instantiated in a concrete architecture, to be reused in systems in the same domain or neighboring domains. In this sense, it is worth highlighting the methodology used for the design of our architecture. Initially, we understood the sources of information that permeate the development of Self-CPS, elaborating a set of architectural requirements. Next, analyzing the studies selected in our mapping, we reused and/or adapted the modules of the architectures/models for our architecture (RA4Self-CPS). Therefore, it can be said that this feature can significantly influence (time and cost) the development of new systems (Self-CPS).

We explored the idea that it is possible to monitor the execution status of an application (i.e., Self-CPS) and the services that compose this application. To so do, our architecture provides a "Monitor" component in the monitoring layer (as illustrated in Fig. 1) that enables us to identify anomalies via Dashboard that a service can present during its execution, such as failure, unavailability, unsatisfactory response time, among others requirements. As reported in Section II-B, QoS is an important feature for service to ensure service quality for Self-CPS. In parallel, we can also identify observability features like metrics, logs, and trace information via the Dashboard in order to guide specialist's decisions.

Regarding security aspects, we adopted a self-protecting approach developed in previous work by our research group [13].

In short, this approach enables the use of a security layer by means of a non-intrusive way, as presented in Section IV. To do so, the developer must implement a filter to capture HTTP requests so that analysis of supported attacks can be identified. Another important aspect to be highlighted in this approach is the elasticity in relation to attacks that can be handled without modifications to the Self-CPS source code. Although our architecture has included security aspects (see **Self-protecting** component – Fig. 1), we recognize that other levels of security can be implemented at different layers of our architecture and/or devices in the physical environment.

Based on the discussion presented in this section, we next report a comparative analysis between the models and architectures selected in our mapping (see Section II-B) and RA4Self-CPS. To do so, we consider four features: security, monitoring, adaptation, and engineering. For security, we considered whether the model or architecture implements any non-intrusive security aspect. In relation to monitoring, we analyzed where monitoring occurs in a Self-CPS. In this sense, we adopted the acronym PE for Physical Environment and AoQ for Aspects of Quality (e.g., quality of service). Regarding adaptation, we classified studies according to implementing the self-* property. Although different techniques have been used to carry out the adaptation (see Table I), we selected only studies that presented concrete evidence of the use of some property for comparison with our architecture. Finally, regarding engineering, we analyzed the models and architectures in relation to the intrusion into source code to implement three previous features. In this sense, only RA4Self-CPS enables the use of a non-intrusive security approach, meeting an important principle in the security area, which is to protect the protector [13], [44]. As reported in Section IV, adaptation at runtime and monitoring activity also do not require modification to the Self-CPS source code. Here, the framework developed by Passini et al.[12] stands out, which enables changing services at runtime with adaptation logic separated from the application (Self-CPS). As shown in Table II, the architecture proposed in this article meets the three first features selected for our analysis. It is worth mentioning that we identified these features in the literature mapping (see Section II-B) and they proved to be essential to design Self-CPS. Still in this direction, our architecture also highlights the well-defined modular organization and engineering flexibility (fourth feature) regarding the coupling of third-party solutions based on a non-intrusive strategy, which may represent the system's transversal interests (e.g., security and monitoring) as described in this section.

Regarding limitations, as reported here and in Section III, our architecture enables the development of security solutions in the application layer. However, considering this system type (Self-CPS), we can implement the security aspects in other layers and devices of this system. Concerning the adaptation, we use a framework developed in our research group [12]. Although this framework has the benefits and contributions presented in this section, it acts to monitor services in relation to the interests of quality attributes, and when these interests are not met (see Section IV), it performs a service adaptation. Finally, another limitation of this work refers to the use only of the Java programming language. Although we have conducted the development of previous projects of our research group and this RA based on good engineering practices, we did not evaluate development in a heterogeneous environment from the viewpoint of programming languages and development stack.

TABLE II
COMPARATIVE ANALYSIS BETWEEN THE ARCHITECTURES AND MODELS IDENTIFIED IN THE LITERATURE AND RA4SELF-CPS

ID	Security	Monitoring	Adaptation	Engeneering
S1	–	PE	–	–
S2	✓	–	✓	–
S3	–	–	–	–
S4	–	AoQ	–	–
S5	–	–	✓	–
S6	–	–	✓	–
S7	–	–	✓	–
S8	–	AoQ	–	–
S9	–	–	✓	–
S10	✓	–	✓	–
S11	–	–	✓	–
S12	–	PE, AoQ	–	–
RA4Self-CPS	✓	AoQ	✓	✓

VI. CONCLUSION AND FUTURE WORK

In this article, we have proposed a novel reference architecture to support the development of Self-CPS, addressing issues like self-adaptive service, self-protecting in the application layer, and application observability. According to our investigation (see Section II-B), these issues provide important information for monitoring an application that enables identifying the security level, execution parameter, traceability, and logging. As an inherent feature of this type of system (Self-CPS), our architecture enables dealing with the adaptation of services at runtime through a framework developed in the previous work of our research group [12].

Regarding the RA4Self-CPS design, it is worth highlighting that its organization was based on layers and modules, which can be considered a positive factor in understanding the architecture by reducing implementation complexity. Furthermore, the modular organization can boost the reuse of modules when our architecture is instantiated to act on systems in the same domain, neighbor domains, or different domains.

The case study presented in this article enables us to objectively evaluate the proposed architecture and confirm its applicability in the health system (i.e., SHHC) but, above all, its adherence to the Self-CPS domain. Drawing a parallel between the architecture organization and the case study conducted, our case study revealed good perspectives for the instantiation of our RA in concrete architecture. In this sense, it is worth highlighting the facility of implementing the self-protecting approach, where only one filter must be implemented as a means of coupling the aforementioned approach to the application that will be monitored, characterizing a non-intrusive strategy for the Self-CPS application (i.e., SHHC). Similarly, the implementation of the observability and service adaptation modules follow the same strategy. However, the service adaptation module requires knowledge about the application to define the variability of application adaptation for framework configuration [12], [46].

As future work, we intend to conduct at least two activities, namely: (i) conduction of more case studies intending to completely evaluate RA4Self-CPS; (ii) instantiation of our RA for other programming languages to evaluate its structures and respective elements, behavior, and relationships between them when a concrete architecture is instantiated; (iii) use of this RA in the industry to evaluate its behavior when it is applied in a larger real environment of development and execution; (iv) evaluate the behavior of RA4Self-CPS when it is instantiated to act in a neighboring system domain. Therefore, it is expected a positive scenario of research that aims to make our architecture (RA4Self-CPS) an effective contribution to the software development community.

Acknowledgements

This research is supported by UNESP's Pro-Rectory of Research (PROPe/UNESP) and the São Paulo Research Foundation (FAPESP) – Brazil (Grants: 2020/10288-5).

References

[1] R. R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, "Cyber-physical systems: The next computing revolution," in *Proceedings of the 47th Design Automation Conference*, (New York, NY, USA), pp. 731–736, Association for Computing Machinery, 2010.

[2] T. Li, F. Tan, Q. Wang, L. Bu, J. Cao, and X. Liu, "From offline toward real-time: A hybrid systems model checking and cps co-design approach for medical device plug-and-play (mdpnp)," in *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pp. 13–22, 2012.

[3] B. Bordel SÁnchez, R. Alcarria, D. SÁnchez de Rivera, and T. Robles, "Process execution in cyber-physical systems using cloud and cyber-physical internet services," *The Journal of Supercomputing*, 05 2018.

[4] M. Salehie and L. Tahvildari, "Self-adaptive software: Landscape and research challenges," *ACM Trans. Auton. Adapt. Syst.*, vol. 4, pp. 14:1–14:42, may 2009.

[5] F. J. Affonso, W. F. Passini, and E. Y. Nakagawa, "A reference architecture to support the development of mobile applications based on self-adaptive services," *Pervasive and Mobile Computing*, vol. 53, pp. 33 – 48, 2019.

[6] H. Muccini, M. Sharaf, and D. Weyns, "Self-adaptation for cyber-physical systems: A systematic literature review," in *2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 75–81, May 2016.

[7] A. Musil, J. Musil, D. Weyns, T. Bures, H. Muccini, and M. Sharaf, *Patterns for self-adaptation in Cyber-Physical Systems*. Springer International Publishing, 2017. cited By 17.

[8] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*. Boston, USA: Addison-Wesley Professional, 3rd ed., 2012.

[9] E. Y. Nakagawa, M. Guessi, J. C. Maldonado, D. Feitosa, and F. Oquendo, "Consolidating a process for the design, representation, and evaluation of reference architectures," in *IEEE/IFIP Conference on Software Architecture*, pp. 143–152, 2014.

[10] F. J. Affonso and E. Y. Nakagawa, "A reference architecture based on reflection for self-adaptive software," in *The 7th Brazilian Symposium on Software Components, Architectures and Reuse*, pp. 129–138, 2013.

[11] F. J. Affonso, G. Leite, R. A. P. Oliveira, and E. Y. Nakagawa, "A framework based on learning techniques for decision-making in self-adaptive software," in *The 27th International Conference on Software Engineering and Knowledge Engineering*, (Pittsburgh, USA), pp. 24–29, Knowledge Systems Institute, 2015.

[12] W. F. Passini and F. J. Affonso, "Developing self-adaptive service-oriented mobile applications: A framework based on dynamic deployment," *International Journal of Software Engineering and Knowledge Engineering*, vol. 28, no. 11n12, pp. 1537–1558, 2018.

[13] R. R. Martins, M. P. de Oliveira Camargo, W. F. Passini, G. N. Campos, and F. J. Affonso, "A self-protecting approach for service-oriented mobile applications," in *Proceedings of the 23rd International Conference on Enterprise Information Systems*, pp. 313–320, 2021.

[14] IBM, "An architectural blueprint for autonomic computing." [Online], 2005. Available: https://www-03.ibm.com/autonomic/pdfs/ACBlueprintWhitePaperV7.pdf, Accessed on January 15, 2024.

[15] R. Lutze, "Digital twins in ehealth – : Prospects and challenges focussing on information management," in *2019 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*, pp. 1–9, 2019.

[16] J. Moyne, Y. Qamsane, E. C. Balta, I. Kovalenko, J. Faris, K. Barton, and D. M. Tilbury, "A requirements driven digital twin framework: Specification and opportunities," *IEEE Access*, vol. 8, pp. 107781–107801, 2020.

[17] A. A. Nazarenko and L. M. Camarinha-Matos, "The role of digital twins in collaborative cyber-physical systems," in *Technological Innovation for Life Improvement* (L. M. Camarinha-Matos, N. Farhadi, F. Lopes, and H. Pereira, eds.), (Cham), pp. 191–205, Springer International Publishing, 2020.

[18] L. M. Camarinha-Matos, J. Rosas, A. I. Oliveira, and F. Ferrada, "A collaborative services ecosystem for ambient assisted living," in *Collaborative Networks in the Internet of Services* (L. M. Camarinha-Matos, L. Xu, and H. Afsarmanesh, eds.), (Berlin, Heidelberg), pp. 117–127, Springer Berlin Heidelberg, 2012.

[19] A. A. Nazarenko and L. M. Camarinha-Matos, "Towards collaborative cyber-physical systems," in *2017 International Young Engineers Forum (YEF-ECE)*, pp. 12–17, 2017.

[20] M. Galster and P. Avgeriou, "Empirically-grounded reference architectures: a proposal," in *Proceedings of the joint ACM SIGSOFT conference – QoSA and ACM SIGSOFT symposium – ISARCS on Quality of software architectures – QoSA and architecting critical systems – ISARCS*, QoSA-ISARCS '11, (New York, NY, USA), pp. 153–158, ACM, 2011.

[21] S. Angelov, P. Grefen, and D. Greefhorst, "A framework for analysis and design of software reference architectures," *Information and Software Technology*, vol. 54, no. 4, pp. 417–431, 2012.

[22] P. Kruchten, *The Rational Unified Process: An Introduction*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 3 ed., 2003.

[23] D. Weyns, S. Malek, and J. Andersson, "Forms: A formal reference model for self-adaptation," in *Proceedings of the 7th International Conference on Autonomic Computing*, ICAC '10, (New York, NY, USA), p. 205–214, Association for Computing Machinery, 2010.

[24] M. P. de Oliveira Camargo and F. J. Affonso, "Systematic mapping protocol – mapping study on reference models and/or reference architectures for the self-adaptive cyber-physical systems." on-line, 2023. https://drive.google.com/file/d/1MexCsklL-2vlch3tbM8us3w9cVgqt7LW/view?usp=sharing, accessed on January 15, 2024.

[25] H. T. Ignatius and R. Bahsoon, "A conceptual reference model for human as a service provider in cyber physical systems," in *2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pp. 1–10, May 2021.

[26] M. Castillo-Effen and N. A. Visnevski, "Analysis of autonomous deconfliction in unmanned aircraft systems for testing and evaluation," in *2009 IEEE Aerospace conference*, pp. 1–12, March 2009.

[27] E. Giallonardo, F. Poggi, D. Rossi, and E. Zimeo, "An architecture for context-aware reactive systems based on run-time semantic models," *PeerJ Preprints*, vol. 7, 2019.

[28] S. Behere, M. Torngren, and D.-J. Chen, "A reference architecture for cooperative driving," *Journal of Systems Architecture*, vol. 59, pp. 1095–1112, NOV 2013.

[29] J. Dias-Ferreira, L. Ribeiro, H. Akillioglu, P. Neves, and M. Onori, "Biosoarm: a bio-inspired self-organising architecture for manufacturing cyber-physical shopfloors," *Journal of Intelligent Manufacturing*, vol. 29, pp. 1659–1682, OCT 2018.

[30] F. Zambonelli, "Engineering self-organizing urban superorganisms," *Engineering Applications of Artificial Intelligence*, vol. 41, pp. 325–332, 2015.

[31] P. Jiang, C. Liu, P. Li, and H. Shi, "Industrial dataspace: A broker to run cyber-physical-social production system in level of machining workshops," in *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pp. 1402–1407, Aug 2019.

[32] J. Lee, M. Azamfar, J. Singh, and S. Siahpour, "Integration of digital twin and deep learning in cyber-physical systems: Towards smart man-

ufacturing," *IET Collaborative Intelligent Manufacturing*, vol. 2, no. 1, pp. 34–36, 2020.

[33] D. Dominic, S. Chhawri, R. M. Eustice, D. Ma, and A. Weimerskirch, "Risk assessment for cooperative automated driving," in *Proceedings of the 2nd ACM Workshop on Cyber-Physical Systems Security and Privacy*, CPS-SPC '16, (New York, NY, USA), p. 47–58, Association for Computing Machinery, 2016.

[34] A. Bhat, S. Aoki, and R. Rajkumar, "Tools and methodologies for autonomous driving systems," *Proceedings of the IEEE*, vol. 106, pp. 1700–1716, Sep. 2018.

[35] G. Fortino and W. Russo, "Towards a cloud-assisted and agent-oriented architecture for the internet of things," in *CEUR Workshop Proceedings*, vol. 1099, pp. 97–103, 2013.

[36] J. Dobaj, M. Krisper, and G. Macher, "Towards cyber-physical infrastructure as-a-service (cpiaas) in the era of industry 4.0," in *Systems, Software and Services Process Improvement* (A. Walker, R. OConnor, and R. Messnarz, eds.), vol. 1060 of *Communications in Computer and Information Science*, pp. 310–321, 2019. 26th Systems, Software and Services Process Improvement (EuroSPI) Conference, Edinburgh, SCOTLAND, SEP 18-20, 2019.

[37] C. Tozzi, "The 3 pillars of observability: Logs, metrics and traces," 2023. Available: https://www.techtarget.com/searchitoperations/tip/The-3-pillars-of-observability-Logs-metrics-and-traces, Accessed on January 15, 2024.

[38] J. Tummers, H. Tobi, C. Catal, and B. Tekinerdogan, "Designing a reference architecture for health information systems," *BMC Medical Informatics and Decision Making*, vol. 21, p. 210, Jul 2021.

[39] M. P. de Oliveira Camargo, "Estabelecimento de uma arquitetura de referência para sistemas ciber-físicos autoadaptativos," master thesis, São Paulo State University (Unesp), Institute of Geosciences and Exact Sciences (IGCE), Rio Claro, 2023. Unesp's Graduate Program in Computer Science (PPGCC), Available at https://repositorio.unesp.br/bitstreams/824e6814-cb65-4b6b-9975-0b9a8c416805/download.

[40] D. Weyns, "Software engineering of self-adaptive systems: An organised tour and future challenges," in *Chapter in Handbook of Software Engineering*, Springer, 2017.

[41] P. Leitão, A. W. Colombo, and S. Karnouskos, "Industrial automation based on cyber-physical systems technologies: Prototype implementations and challenges," *Computers in Industry*, vol. 81, pp. 11–25, 2016.

[42] C. Greer, M. Burns, D. Wollman, and E. Griffor, "Cyber-physical systems and internet of things," *NIST Pubs*, 07 2019.

[43] W. Filisbino Passini, C. Aparecida Lana, V. Pfeifer, and F. J. Affonso, "Design of frameworks for self-adaptive service-oriented applications: A systematic analysis," *Software: Practice and Experience*, vol. n/a, no. n/a, pp. 1–34, 2021.

[44] R. R. Martins, "Autoproteção para a camada de aplicação: uma abordagem baseada em técnicas de aprendizado e no laço de controle mape-k," dissertação de mestrado, Universidade Estadual Paulista (UNESP), Instituto de Geociências e Ciências Exatas (IGCE), Rio Claro, 2022. Programa de Pós-Graduação em Ciência da Computação (PPGCC).

[45] C. Richardson, *Microservices patterns*. Manning Publications Company, 2018.

[46] W. F. Passini, "Desenvolvimento de serviços compostos autoadaptativos: um framework baseado em implantação dinâmica, métricas de qos e informação semântica," Master's thesis, Universidade Estadual Paulista (UNESP), Instituto de Biociências Letras e Ciências Exatas, São José do Rio Preto, Programa de Pós-Graduação em Ciência da Computação (PPGCC), 2020.

Marcos Paulo de Oliveira Camargo received the bachelor's degree from São Paulo State University (Unesp) in 2019. He is currently a software developer at Alpha7 Software, Brazil (Limeira/SP). He has experience in Computer Science with emphasis on the following areas: cyber-physical sytems, software engineering, software architecture, and web development.

Gabriel dos Santos Pereira is a graduate student in Computer Science at São Paulo State University (Unesp). His topics of interest are software engineering, service-based systems, mobile computing, java programming language, JavaFX, and software development.

Daniel de Almeida is a graduate student in Computer Science at São Paulo State University (Unesp). His topics of interest are software engineering, service-based systems, mobile computing, java programming language, JavaFX, and software development.

Leandro Apolinário Bento is a graduate student in Computer Science at São Paulo State University (Unesp). His topics of interest are software engineering, service-based systems, mobile computing, java programming language, JavaFX, and software development.

William Fernandes Dorante received the bachelor's degree from São Paulo State University (Unesp) in 2023. He is currently a software developer at AutBank, Brazil (Rio Claro/SP). Has experience in Computer Science with emphasis on the following areas: service-based systems, software engineering, and web development.

Frank José Affonso is assistant professor at the São Paulo State University (Unesp), Rio Claro/SP, Brazil. He received his Ph.D. in 2009 from the University of São Paulo (USP). He was a postdoctoral researcher in Computer Science from 2013 to 2014 at ICMC/USP. He has experience in Computer Science with an emphasis on Software Engineering in the following areas: Reference Architecture, Service-Based Systems, Mobile Computing, Self-protecting systems, and Self-adaptive Software. He advises students in Unesp's graduate program. He is a member of program committees at various conferences and has acted as a reviewer for several journals. He is the institutional representative of the SBC (Brazilian Computing Society) in Rio Claro.