2016 19th International ACM SIGSOFT Symposium on Component-Based Software Engineering

On the Use of Component-Based Principles and Practices for Architecting Cyber-Physical Systems

Ivica Crnkovic*, Ivano Malavolta†, Henry Muccini‡ and Mohammad Sharaf‡
* Chalmers and Mälardalen University, Gothenburg and Västerås, Sweden
Email: crnkovic@chalmers.se
†Gran Sasso Science Institute, L'Aquila, Italy
Email: ivano.malavolta@gssi.infn.it
‡University of L'Aquila, DISIM Departments, Italy
Emails: henry.muccini@univaq.it, mohammad.sharaf@graduate.univaq.it

***Abstract*—By focussing on Cyber Physical Systems (CPS), this paper investigates how component-based principles and practices are used and support the activity of architecting CPS. For doing so, by running a systematic process, we selected 49 primary studies from the most important publishers search engines. Those papers have been analyzed and their contents classified according to the *Classification Framework for Component Models* proposed in our previous work. The results show that the main concerns handled by CPS component models are those of *integration*, *performance*, and *maintainability*. The instruments to satisfy those concerns, while architecting CPS, are ad-hoc *software/system architecture*, *model-based approaches*, *architectural and component languages*, and *design*. The IEC 61499 standard with its functions block is remarkably used to drive the work on six papers. Java is the most frequently used programming language used for implementing the components. Components are deployed mostly at *compile time*. Interfaces are almost equally distributed into *port-based* and *operation-based*. Overall, the results show a transition of technologies and approaches used in Embedded Systems to CPS, but still lacking methods for integrated architecting, in particular in incremental development.**

***Keywords*-Component-Based Software Engineering; Cyber-Physical Systems; Systematic Literature Review.**

I. Introduction

Cyber Physical Systems (CPS) comprise physical processes and computations that control them into complex distributed software-intensive systems. The complexity and distribution, in combination with real-time requirements, resource constraints, and dependability requirements make the development of such systems particular challenging, and requires new methods in the design (including architecting and detailed design) and implementation [1]. On the one hand CPS are extensions of Embedded Systems that usually interact with, and control physical processes, and have similar requirements related to real-time, dependability and resource constraints. On the other hand the computational complexity, distribution and system adaptability include other types of requirements related both to lifecycle (like continuous evolution) and specifically to run-time (e.g., adaptability).

In this paper the focus is on the use and adoption of component-based approaches in CPS. In some cases CPS use specific component models and their features and in this way address the challenges in a similar way as in embedded systems. As these component models do not solve all problems related to the CPS design, new methods, beyond these used for component models must be applied. The question is whether these methods can be encapsulated in new technologies and integrated with the existing component models? In other cases a design of CPS is not based on a particular component model, but some principles and practices from component-based development are used. In these cases it is of interest whether these principles can be encapsulated in a component model that then can be used in a more standardised way. Further, it is also of interest whether particular practices from component-based approach are used more frequently in the design and some less.

In this paper we investigate, using the systematic literature review (SLR) methodology, the current state of art of using component models in architecting of CPS. Concretely, the following research question is stated: *how do component-based principles and practices are used and support the activity of architecting CPSs?*

Studies related to "components" or "component models" for architecting CPS are systematically selected by following a rigorous selection protocol reported in Section II. The selected studies are then thoroughly analysed, and data related to components and component-based approach are extracted according to "The Classification Framework for Component Models" [2] that provides the main categories of component models (i.e, life-cycle, constructs, extra-functional properties, domains), and their refinements in subcategories. This categorisation is used as a reference framework for the analysis of the studies - if and which extensions this characteristics are used or adopted in the studies. In addition, we looked for other elements related to component technologies referred to in the studies. E.g. the following terms, beyond the classification framework, are also of interest: deployment and communication services, runtime services, internal behaviour, etc.

The rest of the paper is organised as follows. Section II describes the SLR setup, and Section III discusses the main results, whereas observations are provided in Section IV. Section V presents related work and Section VI concludes the paper.

978-1-5090-2569-5/16 $31.00 © 2016 IEEE
DOI 10.1109/CBSE.2016.9

II. Study Design

The overall process we followed for this study can be divided into three main phases [3], [4], [5]: planning, conducting, and documenting. In the following we will go through each phase of the process, highlighting its main activities and produced artifacts.

Planning aims at (i) identifying the scope of our study in terms of its goal and research question (see Section II-A), and (ii) detailing the strategies to be followed in each step of the whole review process. The output of our planning phase is a well-defined *review protocol*. In order to mitigate potential threats to validity the produced review protocol has been circulated to external experts for independent review, and it is made publicly available.

In the **conducting** phase we set the previously defined protocol into practice by performing the following activities:

(i) *Studies search* - searching a series of data sources to produce a comprehensive list of the *candidate studies*, as described in Section II-B. (ii) *Studies selection* - the candidate studies have been screened to obtain the list of primary studies to be considered in later activities of our research. The main driver of this activity is the set of selection criteria described in Section II-B1. (iii) *Data extraction*: we collected information from it from the primary studies (described in Section II-C). (iv) *Data synthesis*: this activity focusses on a comprehensive summary and analysis of the extracted data (see Section II-A and Section II-C).

The **documenting** phase is fundamental for reasoning on the obtained findings and for evaluating the quality of our study. The main activities performed in this phase involve (i) a thorough elaboration on the data extracted in the previous phase with the main aim at setting the obtained results in their academic context, (ii) the analysis of possible threats to validity, specially to the ones identified during the definition of the review protocol, and (iii) the writing of a final report describing in details the performed research.

To allow easy replication and verification of our study we provide to interested researchers a complete *replication package*. The replication package is publicly available[1] and contains the full list of selected studies, extracted data, and the R scripts for summarizing and synthesizing our findings.

A. Goal of the Study

We formulate the **goal** of this study by using the Goal-Question-Metric perspectives [6] (Purpose, Issues, Object and Viewpoint):

Analyse the use and characteristics of component-based principles and practices for architecting CPS from a researcher's point of view.

As outcome, we expect to get a map of the various aspects of component-based principles and practices for architecting CPSs (e.g., used modeling languages, interface specifications, extra-functional properties management, deployment, etc.).

[1]Replication package of this study: http://cs.gssi.infn.it/cbse2016

B. Studies Search and Selection

Our search and selection has been designed as a multi-stage process in order to have full control on the number and characteristics of the studies being considered during the various stages.

We give a brief description of each stage of our search and selection process, together with the number of studies passing each stage.

Stage 1 - relevant journals and conferences. In this stage, we manually searched on Google Scholar with search string *architecting cyber physical systems*. The first 100 entries have been scrutinized, to help defining inclusion and exclusion criteria, to refine the scope of the study, and to select pilot studies. Additionally, we manually browsed relevant journals and conference proceedings. The following list shows the venues we selected for this stage as they contain a large number of studies related to architecting cyber-physical systems.

- IEEE Transactions on Software Engineering
- ACM Transactions on Software Engineering and Methodology
- Elsevier Information and Software Technology
- IEEE International Conference on Cyber-Physical Systems
- ACM/IEEE International Conference on Software Engineering
- European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering
- ACM/IEEE International Conference on Automated Software Engineering
- European Conference on Software Architecture
- Working IEEE/IFIP Int. Conference on Software Architecture
- International ACM Sigsoft Symposium on Component-Based Software Engineering

Stage 2 - automatic search. In this stage we performed an automatic search on electronic databases; in order to cover as much as possible relevant literature about architecting CPS, four of the largest and most complete scientific databases were chosen as data sources, namely: IEEE Xplore Digital Library, ACM Digital Library, SpringerLink, and ScienceDirect.

Based on our research question in Introduction we used the following search string (Listing 1).

The search string was developed by executing pilot searches on IEEE Xplore Digital Library, and checked against a set of pilot studies. For the sake of consistency, the search strings has been applied to an identical set of metadata values (i.e., title, abstract and keywords) from all electronic databases. In total we found 1103 studies.

```
(software OR system)
AND (architect* OR "high-level design" OR "
    conceptual design" OR "abstract design")
AND ("cyber-physical" OR "distributed control system
    " OR "networked control system" OR "sensor
    actuator network" OR "distributed scada" OR "
    federated embedded system")
```

Listing 1. Search string

Stage 3 - combination and duplicates removal. In this stage all the results from previous stages have been combined together and duplicates has been removed. In cases of multiple papers regarding the same study, we kept a record of all of them and point them to a single study. This is necessary for ensuring completeness and traceability of results. After this activity we had 783 studies left.

Stage 4 - studies selection. We considered all the selected studies and filtered them according to a set of well-defined inclusion and exclusion criteria (see Section II-B1). In this stage, each paper has been analysed in two phases: firstly, it has been analysed by considering its title, keywords, and abstract. Those papers whose title, keywords, and abstract are clearly out of scope for our study have been discarded, whereas all the others have been retained and their introduction and conclusions have been analysed to come to a final decision. In total 64 studies were left.
Stage 5 - exclusion during data extraction. This stage has been done in parallel with data extraction. Basically, when reading a study in details for extracting its information, researchers may have agreed that the currently analysed study was semantically too far away from the scope of this research, and so it has been excluded. Finally we got 49 primary studies.

1) Selection criteria: As suggested in [4], we defined the selection criteria of our study during the protocol definition, so to reduce the likelihood of bias. In order to clearly shape the CPS domain (and avoid confusion with related technologies, such as embedded systems, sensor networks, and so on) we stick to the following definition, provided in [1]: *"A CPS is a system of collaborating computational elements controlling physical entities. Unlike more traditional embedded systems, a full-fledged CPS is typically designed as a network of interacting elements with physical input and output instead of as standalone devices"*.

We identified four main aspects that must be always present, that are: (i) interaction with the physical process or environment; (ii) distributed or networked devices, no single device; (iii) open: new entities can come in or out; (iv) a CPS is a system as a whole (possibly at a global level), not a single component of a bigger system.

In the following we provide the **inclusion criteria** of our study: (i) studies proposing, leveraging, or analyzing an architectural solution, architectural method or technique (e.g., tactics, patterns, styles, views, models, reference architectures, or languages) specific for CPSs; (ii) studies in which component-based principles and practices are used during the architecting phase; (iii) studies subject to peer review [5] (e.g., journal papers, papers published as part of conference proceedings); (iv) studies published after or in 2006, since the cyber-physical systems discipline emerged in 2006.

The **exclusion criteria** of our study are the following: (i) studies that, while focusing on CPS, do not explicitly deal with their architecture; (ii) studies that are written in a language other than English, or that are not available in full-text; (iii) secondary studies (e.g., systematic literature reviews, surveys, etc.).

C. Data extraction and Data Synthesis

In order to equally compare the primary studies and to avoid bias, we extracted data from selected studies using a common classification framework presented in [2], whose characteristics are described Section II-D. Moreover, we suitably extended the classification framework for component models in order to better answer our research question. These are the added characteristics:

- Publication data: standard research papers metadata like title, authors, abstract, venue, year of publication;
- Contribution type: paper fragments presenting the overall research contribution of the analyzed study;
- Components use: paper fragments describing how components are used in the presented approach;
- Other CBSE features: paper fragments dealing with component-based principles and practices that do not exactly fit into the classification framework for component models (e.g., used component frameworks, communication and run-time services, etc.).

In order to have a rigorous data extraction process a well-structured data extraction form has been designed upfront. The data extraction form is composed of a spreadsheet with the characteristics of the extended classification framework.

The data synthesis activity involves collating and summarising the data extracted from the primary studies in order to answer our research question [4, § 6.5].

In this study we applied both quantitative and qualitative synthesis methods, depending on the nature of each specific characteristic being considered. Our *quantitative* synthesis method relies on standard descriptive statistics based on the number of primary studies falling into relevant categories for each characteristic of the classification framework. We can divide our *quantitative* synthesis method into two main steps: (i) *counting*: we counted the number of primary studies falling in relevant categories in the context of the characteristic being considered (e.g., we counted the number of primary studies supporting the publish/subscribe interaction style); and (ii) *mapping*: the extracted information has been suitably visualized so that similarities and differences between primary studies was clear. For what concerns the synthesis of *qualitative* data, we used the *coding* technique for qualitative data analysis [7]. Here the main goal was to encode qualitative data (e.g., paper fragments) into emerging relevant categories so that quantitative and standard statistical data analyses can be applied on them. The starting set of categories of our study has been directly derived from the component models characteristics presented in [2].

Finally, we analyzed and summarized the trends and collected information regarding each characteristic of our extended classification framework. In order to mitigate possible biases, the coding procedure and its subsequent analysis have been performed by two researchers; then, the obtained categories have been checked by a third researcher, who was also in charge of resolving possible disagreements.

D. The Classification Framework

The Classification Framework for Software Component Models [2] describes the principles of component-based software engineering and characteristics of component models, and gives overview of a number of component models used in academia and practice in different domains. Here we give

a short summary of the part that we are using for characterising CPS in respect to component-based principles. Figure 1 summarizes the characteristics of component models that are grouped in three dimensions: *lifecycle, construction, and extra-functional properties*. In addition we studied in which *domains* the component-based principles were used.

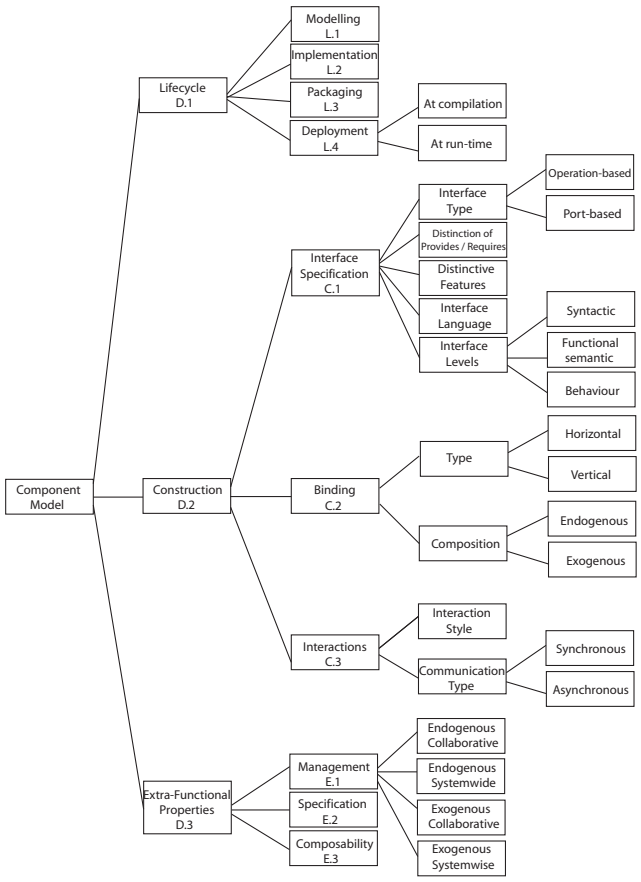

Fig. 1. Characteristics of component models [2]

The lifecycle dimension identifies the support provided by a component model in a component lifecycle. A component lifecycle covers stages from the component modelling until its integration into the systems and possibly its execution and replacement, and includes (i) modelling (e.g. using specific modelling language, or specifying models of components and component-based systems), (ii) implementation (e.g. defining implementations standards in programming languages), (iii) packaging (e.g. providing component repository), and (iv) deployment (which can be at compile- or run-time).

The construction dimension specifies (i) component connection points i.e. interfaces, (ii) mechanisms for establishing connections, i.e. binding mechanisms, and (iii) communication itself, i.e. interaction. Interfaces are means to specify provided and required component functions, which might be at the syntactic, semantic, or behavioral level. Interfaces can describe operations (i.e. functions) or ports that send/receive signals. For an interface specification specific specification languages can be used, or general-purpose programming languages with some extensions, or particular conventions. Binding defines means for establishing connection between components - they can be realised within components (endogenous binding), or outside components via connects (exogenous binding). A binding can realise component compositions in which several components can build a new component (vertical binding), or can establish connection between two components (horizontal binding). Finally, Interaction uses a particular architectural style such as pipe-filer or client-server, performed through synchronous or asynchronous communication.

The Extra-Functional properties dimension identifies support to manage, to specify, and even define compositions of certain extra-functional properties. The management part includes a support to achieve or improve a particular property. For example, reliability can be increased by inclusion of redundancy which some component models provide support for. The specification denotes an ability of a component model to specify a value of a particular property that might be important for system design and realization. Examples of such properties are memory required, execution time, reliability, etc. The composition includes a support for composing certain properties that are possible for the given component model, for example respond or execution time of interacting components, total use of memory, and similar.

III. Results

The first study proposing a component-based approach for architecting CPS is P25 which has been published in 2006 as workshop paper (see Figure 2).

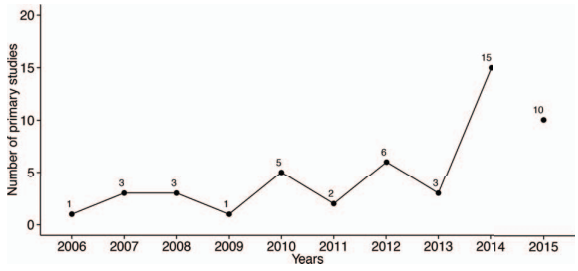

Fig. 2. Distribution of primary studies over the years (partial data for 2015)

By looking at the timeline we can notice that there is a sharp increase in the number of primary studies in 2014 and 2015[2]. Indeed, more than the half of whole primary studies has been published between 2014 and 2015, confirming that in the last years research on CBSE for CPS is gaining increasing interest and that CBSE techniques.

A. Research Contributions

This category identifies the research contribution provided by the selected primary studies. This data item is divided into

[2]Our study covers the studies published before June 2015.; nevertheless, in this year 10 studies have been already published, this result further confirms the growing attention and need of research on CBSE for architecting CPS.

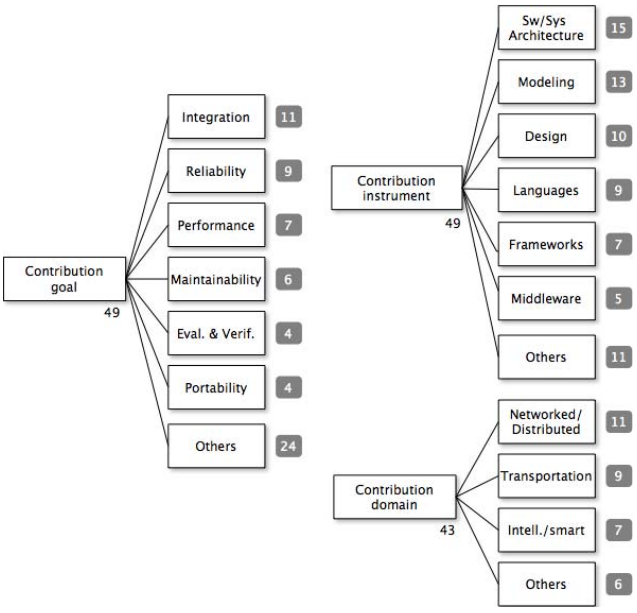

Fig. 3. Contributions

three sub-items, that are *Contribution Goals* (that identifies the concerns, such as quality attributes, process improvement, etc. to be dealt by the proposed CPS component-model), the *Contribution Instruments* (to identify which solution to achieve the contribution goals defined above), and *Contribution Domains* (CPS systems domains) we want to achieve the goals. Figure 3 summarizes information from the primary studies.

1) Contribution Goals: The contribution goals we found in the papers are quite diverse. In order to cluster them, we used the clusters provided in [8], that groups set of concerns related to CPS development into a varied set of clusters.

> *Highlights: the main concern to be managed when engineering component-based CPS are integration (horizontal, multi-view, and consistency), reliability, and performance. Multi-layer, event-driven, and service-oriented architecture represent the most frequent solution.*

Integration is considered to be the main concern to be managed when engineering component-based CPS (11 studies). Integration ranges from horizontal integration of CPS components (P4), propagation of high-level application-driven requisites towards low-level units in the design flow P22, model and multi-view integration (P28, P30), and integration consistency (P33, P40). The second most frequently referred goal is **reliability** (9 studies). Reliable and deterministic RT-components (P1), fault-propagation (P27), safety assurance (P45, P48, P49), failures avoidance (P40), properties guarantee (P41), dynamic verification of component-based real-time systems (P42), and FMEA analysis (P49) are treated by the analyzed papers. **Performance** and **maintainability** are the next most frequent goals, in seven and six cases, respectively. Some particular goals are those of energy efficiency (P12), timeliness (P20), dynamic path planning and speed setting (P46). Reusability is the main concern of papers (P14, P15, P25, P48). Other main goals are **evaluation and verification**, and **portability** (4 studies each).

2) Contribution Instruments: As illustrated in the right upper part of Figure 3, a relevant number of studies either present a novel software/system architecture (15 studies), or a model-based approach for component-based engineering of CPS (13 studies). Multi-layer, event-driven, or service-oriented architecture represent the most frequent instances of software/system architectures, while domain-specific modeling languages for CPS, metamodels, and modeling notations are the most frequent appearances of model-based approaches. Design approaches (such as the use of the IEC 61499 standard with functions blocks, multi-view design approaches, and design by contracts) are the main design techniques. Ad-hoc domain-specific languages are presented in 9 studies (i.e., allocation specification Language, typed graph matching, Time-Space pi-calculus, ModelicaML). Frameworks are presented in 7 studies and new middleware in 5 studies.

3) Contribution Domains: In order to classify the CPS domains, we followed the classification provided by the cyberphysicalsystems.org website. Accordingly, those referred as **Networked or Distributed CPS** are the most frequently associated to component-based engineering (7 studies) (see the right lower part of Figure 3. The papers contained in this category are on: Distributed CPS (DCPS) (P4), Distributed Control Systems (DCS) (P5), distributed reconfigurable embedded control systems (P7), Networked Control Systems (NCS) (P9), distributed lighting automation system (P12), networked based applications (P25), and distributed control architecture (P34). **Transportation** CPS are on the aviation domain (P11), on vehicular CPS (P13), on helicopter systems (P28), and automotive and networked ECUs (P40). **Intelligent/smart systems** CPS focus on smart OO IoT (P17), smart systems (P22), and ambient intelligence environments (P36). Independence on specific domains appears in six studies, while *others* domain cover communication, consumer, energy, infrastructure, health care, manufacturing, military, and robotics domains.

B. Components Lifecycle Support

The used classification framework identifies the lifecycle dimension of a component model as the support it provides in certain points of the lifecycle of components or component-based systems [2]: modelling stage, implementation stage, packaging stage, deployment stage.

> *Highlights: models are almost always used when specifying CPS component models: 23 different modelling languages have been found, with limited use of the UML. OO languages (specifically Java and C++) are the most commonly used to implement components. Components packaging seems to be out of the focus of research on*

TABLE I
MODELLING

No	Modelling	No	Modelling
12	ACDL	4	UML or UML profiles
11	Ad Hoc languages for CPS	3	Simulink
6	Function Block Diagrams	3	Reference models
5	State machines	2	Event models
4	Analysis models	2	Class diagrams
4	AADL 2	2	Hardware Models
15	Others		

component-based architecting for CPS. Deployment is mostly done at compilation time.

1) Modelling Stage: (Table I) Models are used in 45 over 49 studies. This is already an important indicator of the role plaid by model-based design in engineering component-based CPS. There is, however, very little convergence towards a standard model, as demonstrated by the huge variety of models used in the analyzed studies. ACDL (Architecture and component description languages, such as, AADL, SOFA, AUTOSAR, Simplex, PRISMA, AcmeStudio, DEECo, Modelica) have been used the majority of the time, and appear in 12 studies. Those studies essentially make use or extend existing architecture description languages or component languages to model CPS components and architectures. A number of ad hoc notations have been introduced as well, such as policy languages (P8), High Level Architecture Model (P9), and ad hoc domain specific languages (P15, P21, P40). The Function Block diagram, connected to the IEC 61499 standard, is used in six studies (P2, P6, P8, P13, P47, P48), always in combination with other models, except for study P46. State Machines-like notations are used in five studies. Except from P16 and P34 (using state machine diagrams), the other three instances are variations to traditional state machines. The use of UML diagrams is extremely limited (and this came a bit as a surprise) with only four studies using UML Class diagrams, or UML Profiles. The *Others* category is also very rich, including another eleven different type of models (such as Executed Control Chart, agent models, OWL, Transaction Level Modeling, and others) for a total of 23 different modelling language used in combination in 45 studies.

2) Implementation Stage: The existence of a concrete implementation of the components of the system is a precondition for running it. We have been able to extract from 26 studies information about how the components of the system are actually implemented or automatically generated (e.g., in P31, C++ code is automatically generated out of BIP specifications).

Figure 4 shows the distribution of the used programming languages across the primary studies. Here we can notice a clear predominance of object-oriented languages (Java in 12 studies and C++ in 7 studies); this result can be traced (i) to the high popularity of those languages both in academia and in practice and (ii) to the fact that object-oriented languages provide programming concepts like information hiding, encapsulation, inheritance which fit well with those of component-based software engineering. We observed that Simulink is also used in 3 studies, presumably because Simulink directly supports the concept of component, the availability of mature tool support, and for the possibility to simulate the system early during the development lifecycle.

3) Packaging Stage: Packaging refers to the means and facilities used for components storage and packaging, either in a repository or for distribution [2].

During our data analysis we have been able to extract information about components packaging for only 14 studies. This number alone may be already a first indication, as components' packaging seems to be out of the focus of research on component-based architecting for CPS. In those cases in which components packaging has been treated we can see that function blocks of the IEC 61499 standard for industrial process measurement and control systems have been used (6 studies), followed by Java Jar packages (3 studies), Simulink subsystems (2 studies), CORBA (2 studies), and approach-specific plugins (1 study). More in general, in 13 out of 14 studies components have been packaged using standard formats and archives (e.g., Jar archives or Simulink subsystems). An interesting result emerging from our analysis is that current and past research on CBSE for architecting CPS seem to do not focus at all on the definition and usage of repositories for components.

4) Deployment Stage: The deployment of components into a system can happen at two different phases of the systems' lifecycle: at compilation time and at run time. Intuitively, deployment at compile time may limit the flexibility at runtime, but on the other hand enables easier predictability, richer composition features (e.g., hierarchical composition), and more efficient reuse [2].

We can see that the majority of approaches support the deployment of components at compilation time (24 studies), whereas only 16 approaches support the deployment of components at run time. The reason for having a large number of approaches supporting only deployment at compilation time may lie in the above mentioned gains in terms of predictability, composition, and reuse. However, this result can be seen also as an indication for future research on component-based

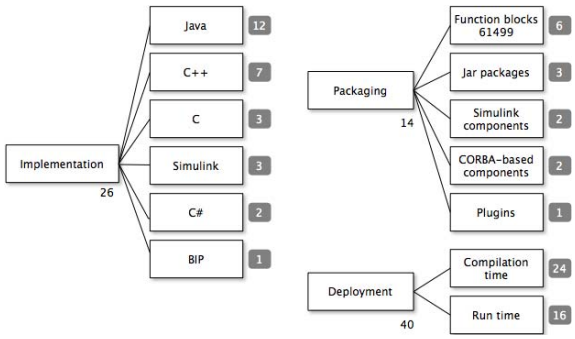

Fig. 4. Components implementation, packaging, and deployment

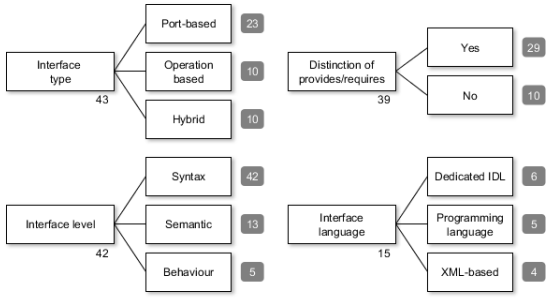

Fig. 5. Characteristics of components interfaces

approaches for architecting CPS in which components will be deployed at run-time; the possibility to deploy components at run time will be a fundamental feature for the CPS of the future in which the demands for (self-) adaptability, automatic reconfiguration and deployment.

C. Components Construction

According to the used classification framework this dimension identifies two different aspects of a component model: (i) the type of interfaces used by a component for the interaction with other components and external environment, and (ii) the means of component binding and communication [2].

> *Highlights: the majority of the studies supports port-based interfaces, with some approaches using both port-based and operation-based interface types. Provided and required interfaces are frequently distinguished. Syntactic, behavioural, and semantic level interface signatures are used by the analyzed studies. Request response and event-driven are far the most frequently used interaction styles. Components are connected directly through their interfaces, therefore, without the need of mediating connectors.*

1) Interface specification: In CBSE, interfaces can be defined as the specification of components' access points [9] in terms of a collection of provided/required operations, their descriptions, and possibly their protocols of interaction. Figure 5 summarizes information from the primary studies.

We have been able to extract from 43 studies the information about the **interface type** supported by each of them; we can notice that the majority of the studies supports *port-based* interfaces (23 studies), in which ports are used for naming entry points of the components for passing data, for triggering or receiving events, etc. *Operation-based* interfaces have been used in 10 studies, where interfaces are defined as functions with signatures, or even attributes. Interestingly, 10 primary studies adopt an *hybrid* approach in which interfaces are both port-based and operation-based; for example, the approach presented in P35 proposes a two-levels interface where ports are the main interaction points of each component, and each port has an associated interface containing the services provided or requested by the component.

Many component-based approaches allow engineers to distinguish between **provided and required interfaces**; in our study we identified 29 primary studies distinguishing between them and only 10 studies that do not make such a distinction. Such a result can be traced to the fact that distinguishing between provided and required interfaces is an accepted desiderata since it allows engineers to bind components in a more predictable manner, enabling a clear assessment of the impact that the addition, substitution, or removal of a components may have on the overall system, thus improving (above all) the system evolvability.

Further, we assessed the **level of contractualization of interfaces** across the studied approaches. the first level, *Syntactic* level enables component interacting with another one must refer to the correct "signature" of the called service, e.g., in terms of its types, fields, methods, etc.; in this case certification is quite straightforward since well-known techniques for static and dynamic type checking can be used. Secondly, we identified the *behavioural* level, where services are described also via their dynamic behaviour (e.g. in P7) each component is described as a set of nested state machines supporting all its reconfiguration forms); in our study we found 5 approaches falling into this category, where certification must take into consideration also the compatibility of the behaviours of the components within the system. Finally, the third level is the *semantic* one, where service signatures are reinforced by certifying that the values of actual parameters and the component's state variables have permitted values. In our study we found 13 approaches falling into this category. This level of interfaces enforcing can be reached via different techniques; for example in P48 interfaces are annotated with contracts specifying timing assumptions and guarantees, whereas in P34 each interface can have a set of timing constraints predicating on each operation.

Besides their name and signature (when it is supported by the component model), components interfaces can be defined via **specific languages**. We have been able to extract such information from 19 studies and classified them accordingly. As a result, we can observe that 6 studies use a dedicated Interface Description Language (*IDL*), such as the Microsoft Interface Definition Language in P14, the OMG IDL in P15, the CORBA IDL in P27, as well as a set of approach-specific IDLs like in P29, P34, and P35. Moreover, in 5 studies *standard programming languages* have been reused as interface description languages, like Java in P2, C++ in P25, C in P31, and C header files in P1 and P38, whereas *XML-based* description of interfaces have been used in P7, P8, P19, P20.

2) Components Interactions: Here we focus on how components are bound together and how they communicate. Figure 6 summarizes the information from the primary studies.

The first characteristic we focus in this dimension is the **interaction styles** (aka as architectural styles) used in the various approaches. We have been able to extract the used interaction styles for 43 primary studies; among them, the clear winners are the *Request response* and *Event-driven*

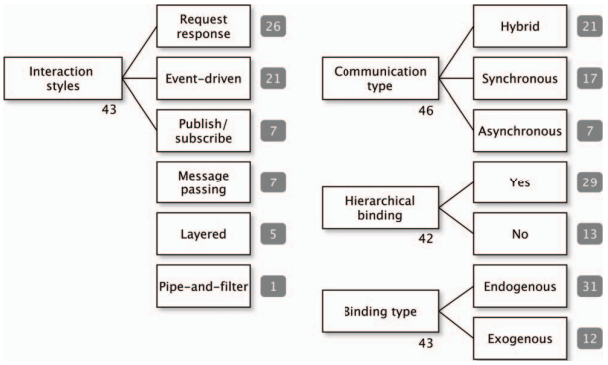

Fig. 6. Characteristics of components interaction

styles with 26 and 21 studies, respectively. This result is not surprising since the request response interaction style is commonly used in client/server architectures [2]. The event-driven interaction style represents all those systems which transfer events among their components; in those systems components can trigger or react to specific events, which are transferred among components via event channels. The event-driven style is widely used across our primary studies; this is a clear example about how the application domain and the environment of a system can influence its architectural style; indeed, it is very well-known that one of the key features of cyber-physical systems is their intrinsic reactive nature, where the behaviour of the system as a whole must react and adapt according to the ever changing dynamics of the associated physical process. Other used interaction styles are publish/subscribe (7 studies), message passing (7 studies), the layered style (5 studies), and the classical pipe-and-filter (1 study). Interestingly, we observed that there are 8 studies in which the request response and the event-driven interaction styles are used in combination (P7, P10, P11, P16, P23, P28, P39, P49); this result can be seen as (i) a concrete example about how architectural styles can be combined together for managing massively large and complex systems like CPS, and (ii) an indication that those two interaction styles are the most used and scientifically investigated for architecting cyber-physical systems via the component-based design and development paradigm.

The **communication type** characteristics details if the communication used are synchronous and/or asynchronous [2]. From obtained data we can notice that the majority of approaches (21 studies) adopt synchronous and asynchronous communication at the same time, followed by synchronous communication only (17 studies), and asynchronous communication only (7 studies). This result is interesting since it highlights the recurrent necessity of CPS engineers to reason about possible tradeoffs between the high performance and flexibility of asynchronous communication and the well-known reliability of the synchronous one (e.g., for inter-component communication across a network).

In component-based software engineering it is possible to have a **hierarchical binding** (or composition in the literature) of components; in those cases the component model considers and treats assemblies of components as single components, which then still obey to the component model rules. This is one of the most used techniques for managing the complexity of the system because engineers (and tools) can abstract a group of interconnected components as a single one, thus gaining in terms of cognitive load for engineers and computational effort for analysis engines. Our analysis is in line with this trend, the majority of selected approaches support hierarchical binding (29 studies), whereas only 13 approaches do not support it. We cross-tabulated this characteristics with the year of publication of primary studies, and we noticed that this trend is mainly constant across time.

When defining a component model one of the most crucial design decisions is whether to include connectors as first-class architectural elements (i.e., having an endogenous **binding type**) or not (thus having an exogenous binding type). More specifically, endogenous binding means that the components are connected directly through their interfaces (i.e., information concerning the binding resides inside components), whereas in exogenous binding components are mediated by connectors (i.e., components have no knowledge of which other components they are connected to). Our study reveals that the majority of selected approaches support endogenous binding (32 studies), rather than the exogenous one (11 studies). By itself, this result may not come as a surprise since it is well known that connectors are not common in many component models [2]. Interestingly, 8 out of 12 approaches supporting exogenous binding have been published in the last 5 years; this result can be interpreted as an indication that researchers are getting an increasing sensibility and interest to a more seamless evolution of the system, which is one of the most driving levers towards having exogenous binding.

D. Extra-functional Properties Support

In certain domains, e.g., in real-time embedded systems, safety-critical systems as well as cyber-physical systems, EFPs are equally important as their functional counterparts; this means that it is fundamental to precisely model, assess, and verify EFPs of systems.

Highlights: only one third of the studies support EFPs, mostly timing properties.

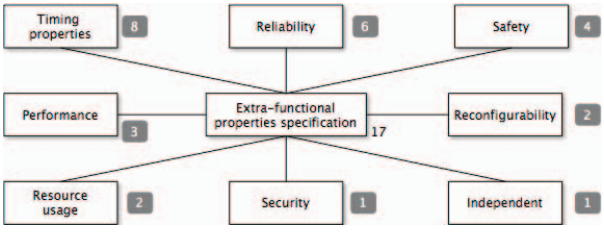

Fig. 7. Specification and analysis of EFPs

It came as a surprise that only 17 out of 49 primary studies provide a certain support for the **management of EFPs**. This result is revealing gap that actually exists in our research community, nevertheless we also noticed that recently this gap is getting narrower, as in 2015 there is a certain balance between approaches supporting EFPs (4 studies) and those who do not (6 studies). This finding is a clear indicator that studying EFPs in the context of component-based approaches for architecting CPS may be a interesting and fruitful direction for future researchers.

When considering **specification and analysis of EFPs**, we can notice that there is a certain trend in supporting timing properties (8 over 17 studies), reliability (6 over 17 studies), and safety (4 over 17 studies). We can interpret this results as a consequence of the intrinsic nature of CPSs in which software is directly interacting with the physical world, including human actors, the environment, and mission-critical equipment. Along the same lines, our analysis confirms also that properties such as performance (3 studies), resource usage (2), and security (1 study) have been (less) investigated. Differently, we noticed that reconfigurability, which is a more technical and internal extra-functional property with respect to the previously discussed ones, has been investigated in two studies (P21, P35). Finally, it emerged that in one study (P35) the proposed approach is independent from any specific EFP. In P35 a formal aspect-oriented language is used for describing the architecture of the system so that both functional and extra-functional properties are specified as aspects.

E. Domains

We distinguish *general-purpose* if its component model is not tied to any domain (e.g., the approach refers only to generic components within a CPS), and *specialized* if its component model is specifically tailored to a given domain (e.g., its modeling language presents dedicated constructs for concepts belonging to the automotive, avionics, etc.).

> *Highlights: general-purpose approaches are far the most frequent, whereas specialized are emerging.*

Given the stringent peculiarities of the CPS domain (e.g., the extremely tight timing constraints), we may expect a high number of specialized approaches; surprisingly, our analysis reveals that 33 over 49 studies are general-purpose, whereas only 16 over 49 studies are specialized. For what concerns the distribution of approaches over the years, overall general-purpose approaches are equally distributed during the years, whereas specialized approaches are strongly emerging only recently (even 10 out of 16 studies have been published between 2014 and 2015); this result can be seen as an indication of a growing interest of researchers towards specialized approaches for architecting CPS via component-based techniques in recent and future years. Specialized approaches belong to the following domains: home automation (4 studies, P2, P12, P36, P39), automotive (P14, P38, P44) and production systems (P5, P46, P47) in 3 studies, respectively, robotics (P1, P18) and avionics (P11, P28) in 2 studies, respectively, smart cities (P3) and networked control systems (P20) in 1 study, respectively.

IV. Findings

The results of our analysis clearly show that the architecting CPS emerged from Embedded Systems (ES). The component models, programming languages, interaction styles, and EFPs considered in the studies, are extensively used in ES. This is expected as the basic elements of CPS are ES. However, in CPS there are extended concerns, first of all concerning the more extensive focus on distributed communication. In selected studies the component-based principles are dominated in modelling and designing the ES part of CPS. The communication part (expressed in ”interactions” and ”bindings”) shows extensions of communications provided by the component models. In some cases this communication is expressed as extensions of component models, but in most of the cases the interaction styles (dominated by the request-response and event-driven ones) and the communication styles (hybrid) are different than in ES, and outside of scope of the used component models. Many solutions are based on different types of middleware that separate design of computing nodes (i.e. ES), from their distribution. This separation of concerns makes it easier to deal with the intrinsic complexity of CPS, but imposes challenges in the integration, modelling and verifying EFPs on the entire system level. While the middleware is successfully used in establishing interactions between different systems parts (e.g., execution nodes, or subsystems), there is a lack of support for incremental development on the system level. A typical example is the inclusion of a new feature in a CPS that requires changes across the architectural layers and different abstraction levels. For instance, adding a complex feature in a vehicular system (like a vehicle stability, or autonomous driving module) requires incremental changes across different ECUs (i.e., Electronic Computing Units) that are connected by a bus (e.g., the CAN bus). A challenge that appears here is not in the functional implementation of a feature but its deployment on several ECUs and its integration with the existing components in the ECUs and synchronisation across the entire system due to limited resources, such as memory and time. One possible direction in addressing these challenges is to develop a layered component model (as for example [10] that integrate different communication styles and interaction types, and provides support for overall system modelling and analysis, and in particular across the middleware or architectural layers).

The second finding is the *support for EFPs*. Even if it was seen in only 17 studies, percentage-wise this number is larger than in the studies of component models in general, and approximately the same as for the ES domain [11]. This indicates that EFPs questions are important, but still not yet fully explored by the community.

The third finding is an *absence of adaptation and dynamic deployment* embodied in component models. While certain component models successfully use dynamic deployment

mechanisms, they are not utilised and not further developed for CPS. This is a clear opportunity to improve the CPS architecting at run-time.

A fourth finding, a surprising one, is *the extensive use of programming languages suitable for ES* (such as in IEC 61499), but having only rudimentary support for distributed systems. Here new extensions and new languages could improve the design and implementation of the CPS of the future.

V. Related Work

In the last few years, several research efforts have been devoted to Architecting Cyber Physical Systems. In particular, CBSE is considered one of the distinguishable approaches in architecting such systems. Recently, much work had tackled CPS under different concerns. However, at best of our knowledge, this is the first work applying a systematic literature review to synthesize evidence on studies using CBSE for architecting CPS.

A systematic mapping study to map out the CBSE area in the period 1984-2012 has been presented in [11]. The authors address five features about CBSE: main objective, research topics, application domains, research intensity and applied research method. This work is a very important analysis for CBSE research, but it is not specific for CPSs.

In our previous work [12] we investigated the state-of-research on architecting CPS. That preliminary study aims to identify, classify, and understand existing research on architecting CPSs. It addresses ten venues in the last decade. It is not specific to CBSE to architect CPSs.

A simple review of CPS architectures is reported in [13]. The paper analyzes three features of CPS architecture design, real time control, security assurance and integration mechanism. They reviewed the developments of CPS from the architecture aspects without considering CBSE as an approach for Architecting CPS and without following a systematic way in their review.

VI. Conclusions and Future Work

How do component-based principles and practices are used and support the activity of architecting CPSs?

We might have not provided a definitive answer, but we have shed some light on how component models, with their principles and practices, have been applied so far to CPS. As expected, a lot of research is ongoing, with an important growth in the past two years. The trend in component-based software engineering of CPS follows the one of embedded systems, with an even increase on architectural aspects, to design the distributed nature of CPS. New component models, languages, and ad-hoc architectures are being created to deal with CPS concerns such as integration, reliability, and performance. Model-based engineering, in terms of domain-specific languages, and new metamodels is quite frequently used when engineering CPS. Integration, reliability, and performance are very important concerns covered in 27 of the 49 papers under analysis. Interface languages are formalized through dedicated IDL or programming languages, with a distinction between provided and required interfaces frequently covered.

As future work we are planning to extend this study in order to (i) enlarge its scope to architecting cyber-physical systems and (ii) perform a more thorough statistical analysis of extracted data in order to further mitigate potential biases and to better determine hidden correlations between the considered categories. Also, based on the learning of this work, our future work will be oriented to cover the gap between the needs present in CPS, and what realized so far.

Acknowledgment

This research is partially supported by the Swedish Foundation for Strategic Research through the RALF3 project.

References

[1] E. A. Lee, "Cyber Physical Systems: Design Challenges," in *ISORC*, 2008, pp. 363–369.

[2] I. Crnković, S. Sentilles, A. Vulgarakis, and M. R. Chaudron, "A classification framework for software component models," *Software Engineering, IEEE Transactions on*, vol. 37, no. 5, pp. 593–615, 2011.

[3] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015.

[4] B. A. Kitchenham and S. Charters, "Guidelines for performing systematic literature reviews in software engineering," Keele University and University of Durham, Tech. Rep. EBSE-2007-01, 2007.

[5] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*, ser. Computer Science. Springer, 2012.

[6] V. R. Basili, G. Caldiera, and H. D. Rombach, "The Goal Question Metric Approach," in *Encyclopedia of Software Engineering*. Wiley, 1994, vol. 2, pp. 528–532.

[7] M. B. Miles and A. M. Huberman, *Qualitative data analysis: An expanded sourcebook*. Sage, 1994.

[8] G. Sapienza, I. Crnkovic, and P. Potena, "Architectural decisions for HW/SW partitioning based on multiple extra-functional properties," in *2014 IEEE/IFIP Conference on Software Architecture, WICSA 2014, Sydney, Australia, April 7-11, 2014*. IEEE Computer Society, 2014, pp. 175–184. [Online]. Available: http://dx.doi.org/10.1109/WICSA.2014.19

[9] C. Szyperski, "Component software: Beyond object-oriented programming," Boston, MA, USA, 2002.

[10] S. Sentilles, A. Vulgarakis, T. Bures, J. Carlson, and I. Crnkovic, "A component model for control-intensive distributed embedded systems," in *Component-Based Software Engineering*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, vol. 5282, pp. 310–317. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87891-9_21

[11] T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. C. Cavalcanti, and S. R. de Lemos Meira, "Twenty-eight years of component-based software engineering," *Journal of Systems and Software*, vol. 111, pp. 128 – 148, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121215002095

[12] I. Malavolta, H. Muccini, and M. Sharaf, "A preliminary study on architecting cyber-physical systems," in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ser. ECSAW '15. New York, NY, USA: ACM, 2015, pp. 20:1–20:6. [Online]. Available: http://doi.acm.org/10.1145/2797433.2797453

[13] L. Hu, N. Xie, Z. Kuang, and K. Zhao, "Review of cyber-physical system architecture," in *Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW), 2012 15th IEEE International Symposium on*, April 2012, pp. 25–30.