Component Models for Embedded Systems in Industrial Cyber-Physical Systems

Luis Neto*†, Gil Gonçalves*†
*SYSTEC-FoF, Research Center for Systems and Technologies - Factories of the Future
†FEUP, Faculdade de Engenharia, Universidade do Porto
Rua Dr. Roberto Frias, s/n 4200-465, Porto, Portugal
Email: {lcneto,gil}@fe.up.pt

***Abstract*—Component Based Software Engineering is a traditional methodology that has significant advantages: reduction of production cost, code reuse, code portability, fast time to market, systematic approach to system construction and guided system design by formalization and domain specific modelling languages. This methodology is used in frameworks for enterprise systems, user interfaces, Web applications, embedded systems for Industrial Cyber Physical Productions Systems (ICPPS) and Industrial Internet of Thing's (IIoT). In this work, we surveyed component model solutions and literature applied to Industrial Cyber Physical Systems (ICPS). By conducting a reproducible systematic mapping study, we search and select results of interest. Research Questions (RQs) are formulated and addressed by applying classification schemes to the results. Finally, the classification results allow us to come up with a state-of-the art in this domain and to draw some conclusions about design considerations and research trends.**

***Keywords*–Component Based Software Engineering; Component Models; Embedded Systems; Industrial Cyber-Physical Systems.**

I. Introduction

In this paper addresses a systematic mapping study in component models for embedded systems in industrial environments. All iterations of the systematic mapping process are based in [1] and are detailed throughout the document, finishing with results that consider the research questions shown below. Heineman and Councill [2] provide a clear and unambiguous definition of *software component*, *component model* and *software component infrastructure* that we will use as reference throughout this paper.

1) **RQ1.** Which component models exist whose scope of application is ICPS and whose targets are embedded systems?
2) **RQ2.** What are the similarities and design considerations among them?
3) **RQ3.** How has research in this topic been evolving?
4) **RQ4.** What kind of contribution is given by particular papers?

A software architecture designed using the component model solution is developed as "a composite of sub-parts rather than a monolithic entity" [3]. The advantages of this approach address many objectives of software industry, such as: reduction of production cost, code reuse, code portability, fast time to market, systematic approach to system construction and guided system design by formalization and domain specific modelling languages.

The component model is the foundation of a component based design. It defines, briefly, the composition standard: how components are assembled into larger pieces, how and if they can be composed at design and/or runtime phases of component life-cycle, how they interact, how the component repository (if any) is managed, and the runtime environment that contains the assembled application. Because of all of this, component models are hard to build. Some problems like achieving determinism and real-time, parallel flows of component and system development, maintaining components for reuse, different levels of granularity [4] and portability problems [5] may occur.

Our study focuses on component models whose target are factory shop floor systems, and component models that allow to compose solutions for discrete or continuous control and automatic reasoning, the so called component-based industrial automation applications. Component based design architectures, as classified in Vyatkin's work [6], are part of traditional software engineering methodologies. From the key areas of software engineering [6], we will focus our attention on software design and construction, configuration management, tools and methods.

We are interested in covering various levels of components, from those that represent lower level parts of embedded systems (such as drivers and system kernels) to higher level (such as algorithms and services). Component models can be characterized by their capability to assemble components. These can be composed using wrapping, static and dynamic linking, and "plug-and-play" methods. Component models are, typically, thin layers that operate on top of an operating system (OS) or runtime environment (RTE), which brings portability and reuse issues. Because of the advent of IIoT and ICPS, many hardware vendors are providing heterogeneous solutions that require OS and RTE independent solutions.

The rest of this paper is organized as follows: Section II provides details of the search and selection process for articles; Section III discusses some of the results found to provide the reader with support for better interpretation of the mapping process, later explained in Section IV. Section V concludes the paper with a final discussion.

II. Primary Search

In [1], the authors present a series of steps that show how to perform a systematic mapping study. These steps are illustrated in Figure 1. The information sources for the first iteration of the study were only databases of reference: *SCOPUS, IEEExplore* and *ACM Digital Library*. The initial search string used clearly reflects the research questions: *(**TITLE-ABS-***

KEY (component model) AND TITLE-ABS-KEY (industry) AND TITLE-ABS-KEY (embedded systems))

Following the systematic mapping process, we did a first review of abstracts and selected a first set of documents based on the criteria of Table I. Because every research topic has a specific terminology unknown to the unfamiliar reader, new keywords of interest (e.g., *Software Component Framework, Component-Based Software, Component Life Cycle, Component Syntax, Component Semantics, Component Composition*) to the RQs were identified to increase the search efficiency.

TABLE I. INCLUSION AND EXCLUSION CRITERIA

Inclusion	Exclusion
• Books and papers reporting final solutions, methodologies and evaluation of component models for embedded systems in industrial scenarios. • Available and existing solutions (both commercial and academic) with documentation reporting experiments, validation and use cases. • Opinion, survey, taxonomy and classification frameworks, and philosophical findings on component models for embedded systems in industrial scenarios.	• Books and papers with less than 10 references will be excluded. • Any finding that does not discuss the three main keywords in the abstract and introduction ”component model”, ”embedded systems” and ”industry” will be excluded. • Component frameworks with exclusive application to enterprise systems, user interfaces, web-applications and others rather embedded systems for industrial domain.

A. *Search and Screening*

To the original set of steps - the blocks represented with a contiguous outline - were added those outlined by dashed lines of Figure 1. The first search query was built combining the most frequent words of the accepted papers and the RQs. To produce this set of frequent words the keywords and abstracts of all accepted documents were gathered in a spreadsheet and parsed. *RapidMiner Studio* [7] was the text analyser tool used to count frequent words. Sequentially, this tool also allowed to use an English *stopwords filter* and a *n-Grams* operator, which allows to make combinations of *n* keywords, to count frequencies of up to 4 consecutive words. After processing, the resulting set of keywords contained 44 keywords of interest. performing combinations with this set was a time consuming process, so we tried to query the selected databases with the entire set at a time but none of them accepted such a long query. After that, we decided to try the *Google Scholar* search engine. This option was viable because it accepted the long set of keywords and this resulted in very accurate preliminary results. The results from the two queries were merged to obtain an extended set of papers. At that point, we decided that to perform a pragmatic application of criteria. The number of citations considered could not be the same because *Google Scholar* takes in account citations from a wider set of sources than the other databases. To solve this issue, each individual paper of the first set was searched in *Google Scholar*. Then, for each paper found, the multiplicative factor between the number of citations in the first and second set was calculated. Finally, the average of all multiplicative factors was calculated. This average value was used to replace the minimum number of references considered in Table I. For a *Google Scholar* paper in the second set to be considered it must have a minimum number of 36 citations.

The application of the exclusion and inclusion criteria specified in Table I drastically reduced the number of documents considered in this study, as can be observed in Table II. The final set of documents was used to conduct the evaluation. For that, a specific classification scheme was combined with the mapping process. This is detailed in the next chapters.

III. Mapping Process

The following works, after a complete reading were the ones of major interest for this study and will be used throughout the mapping process: *PECOS* [8], Timing Definition Language (TDL) [9], *FORMULA* [10], *Bold Stroke* [11], *Rubus* [12], Real-Time-Linux-Based Framework enhanced with *IEC 6144*9 [13], *IEC 61449* model [14], Programming Temporally integrated distributed embedded systems (PTIDES) [15], *Kevoree* [5], [16], Automatic Reasoning [17], Critical Scenario simulation using *IEC 61449* [18] and Component Design to tackle safety analysis [19]. Some of the papers analysed did not provide enough details to fill rigorously the classification schemes adopted, but all were of the highest interest to provide insight in this study.

Figure 2 gives a concise overview of a component model. It shows two main phases, from a component creation to its usage. In the first stage, the component is built in a builder environment, which can be a code editor (mostly when developing from scratch) or in a graphical editor (mostly when using reusing built components to produce a composite component, these are normally represented by graphic shapes or diagrams). The design phase ends with the developer sending the component to a repository. In some cases, when there is no repository, the component can be directly sent to a RTE. In the deployment phase, components are fetched from the repository, composed in a graphical or code environment and finally sent to the RTE.

A. *Classification Schemes*

Four classification schemes will be taken into account to perform the mapping of papers found. The first classification scheme divides the results in: opinion, survey, taxonomy and classification frameworks, and philosophical findings. The second scheme is based on available and existing solutions (both commercial and academic) providing documentation reporting: experiments, validation and use cases. The third classification, which is based on previous ones, specifically addresses RQ3. The last classification scheme addresses RQ4 and the categories used are based in [1].

Some classification categories can not be applied to some papers from the extended set of relevant works. For examples theoretical and other survey papers does not apply to the choose taxonomy for component models. For this reason the number of references in the classification tables is not consistent.

1) Taxonomy Based: There exists literature [20, 21, 22, 23] that propose classification schemes specifically for component based software engineering. In [22], the authors provide a formal and comprehensive framework of classification that will

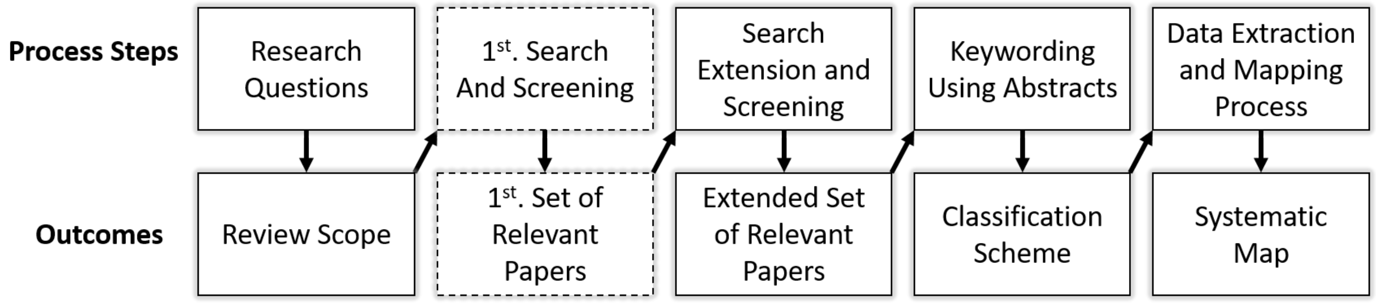
Process Steps
Research Questions
1st. Search And Screening
Search Extension and Screening
Keywording Using Abstracts
Data Extraction and Mapping Process
Outcomes
Review Scope
1st. Set of Relevant Papers
Extended Set of Relevant Papers
Classification Scheme
Systematic Map

Figure 1. Modified Systematic Mapping Process.

TABLE II. DOCUMENTS AFTER CRITERIA

	SCOPUS	*IEEExplore*	*ACM Digital Library*	*Google Scholar*	Duplicates
Initial (Duplicates)	390	206	135	913	133
>10 References (Duplicates)	42	17	14	71	10
Text Analysis (Duplicates)	5	4	4	5	1
Final Set	18				

not used because of the superficial nature of the reviewing process in systematic mapping approaches. The taxonomy that we will address is proposed in [20] and it classifies component models using the following three characteristics.

- **Component Syntax**: The syntax of components is the *component definition language*. In some cases it is a programming language, but if the solution is required to be more flexible it can be a specific language defined by the component model. In the last case, a compiler can generate code in various programming languages and make the components more versatile. Table III shows the syntax of the component models analysed.
- **Component Semantics**: The semantics of a component is what it is meant to be: it can be an object (in the sense of object oriented languages), it can be a plain piece of business logic code and also to be manipulated by a manager instance created by the container at deployment phase. In this sense, the semantic is given by the run-time environment and defined by the component model. Table IV shows the semantics of the component models analysed.
- **Composition**: Process in which components are assembled together to create new components or systems. This process can happen in two phases (Figure 2) of the software component life-cycle: at deployment phase, the builder environment is able to retrieve existing components from the repository and use them to create a new one, that in the end packaged, catalogued and sent to repository; at deployment phase, existing components in the repository can be assembled and later instantiated in a run-time environment.

TABLE III. COMPONENT SYNTAX

Component Syntax	Component Model
Object Oriented Programming Language	
IDL (interface definition language)	[12, 5]
Architecture Description Languages	[8, 9, 10, 13, 14]

TABLE IV. COMPONENT SEMANTICS

Component Semantics	Component Model
Classes	[12]
Objects	[13, 14, 5]
Architectural Units	[8, 9, 10]

TABLE V. COMPOSITION CLASSIFICATION

Category	Component Models	Characteristics				
		DR	**RR**	**CS**	**DC**	**CP**
1	[8, 10]	x	x	x	✓	x
2	[12],	x	x	✓	x	x
3	[9]	x	x	x	x	✓
4	[13, 14]	✓	✓	x	✓	x
5	[5]	x	x	✓	x	✓

Regarding the composition classification, the original taxonomy in [20] defines 5 characteristics of composition. These characteristics were mapped into categories for this study. The characteristics are: **DR**, In design phase new components can be deposited in a repository; **RR** In design phase components can be retrieved from the repository; **CS**: Composition is possible in design phase; **DC**, in design phase composite components can be deposited in the repository; **CP**, composition is possible in deployment phase. Table V shows the composition classification for the component models analysed.

2) Design Considerations: A component system capable of performing real time was a characteristic perceived as of the highest importance when reading through the chosen papers. This characteristic also introduce some concerns that are typical from the high performance computing domain. Parallelism, (a)synchronism, worst case execution time, events, threads, the mix of hard, soft and non real-time constraints are characteristics that concern to industrial control applications and that are hard to achieve altogether in component models. Integrating technologies from multiple vendors is challenging and often results in fragile tool chains that requires a considerable effort to maintain. This also touches the domain of granularity: a single component can emulate an entire system (coarse grained), benefiting from the reliability and efficiency,

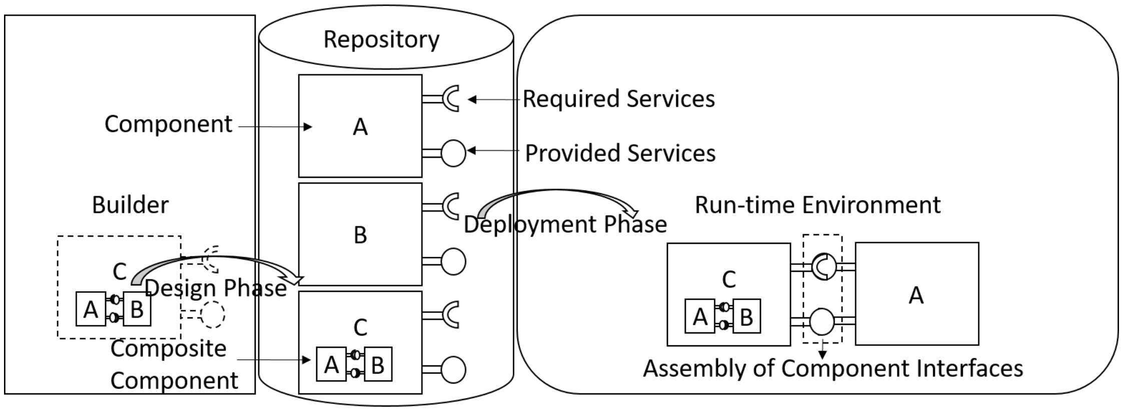

Figure 2. Component Model Overview.

but having a reduced capability of reuse. The footprint of components and its container (the run-time environment) is a recurrent concern when developing to embedded systems, which are typically very constrained. System communication refers to the application of component models to distributed systems. In scenarios where several nodes in a network are distributed physically over a production plant, the component model should be capable of making this nodes interact as components of a monolithic system.

TABLE VI. DESIGN CONSIDERATIONS

Design Consideration	Component Model
Component Granularity	[5]
Intelligent Reasoning	[17]
Real-time	[12, 21, 9, 11, 13, 14, 15, 18]
Security	[5, 19]
Footprint	[12, 5]
Portability	[10, 10, 13, 14]
Component Reuse	[11, 13, 14]
System Communication	[21, 9, 14, 15, 5]
Systematic Design	[12, 10, 10, 14, 16]

3) Design Considerations Over Time: The graph of Figure 3 shows the evolution of design considerations over the years. Despite the small population used to trace the graph, some conclusions can be drawn. This classification addresses RQ3.

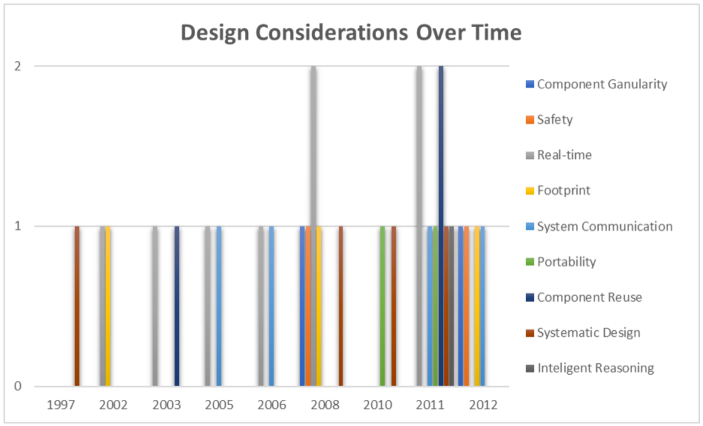

Figure 3. Design Considerations Over Time.

4) Type of Research and Contribution: According with the research type facet defined in [1], Table VII shows a classification of works presented in the previous sub chapters. This table addresses RQ4.

IV. RESULTS AND DISCUSSION

In this section we discuss some of the findings with more relevance to the topic. The objective of this analysis and the present discussion is to gain some insight about component models and about some details in the solutions found. There are some design considerations typical of industrial scenarios that this research addresses and are important to retain.

According to Lau et al. [3], components can be divided into 2 main classes, 1) objects, as in OO languages; 2) architectural units, that together compose a software architecture. According to the authors, there are no standard criteria for what constitutes a component model. Components syntax, is the language used to component definition and which may be different from implementation language. Typically the component containers and runtime environments are general purpose server computers. In this case we are interested in a particular kind of architecture in which a centralized general purpose server holds the component repository and the runtime environment is contained in physically distributed embedded systems. The taxonomy that Lau et al. [3] work defines will be used to describe the results found in the systematic mapping study. The authors conclude that a theory that supports component model process in the whole life-cycle does not exist and that a perfect component model should allow composition at design and runtime phases. A component should be deployed along with a complete information of its provided and required interfaces [2]. To enable reuse and interconnection of components, component producers and consumers must agree on a set of interfaces before the components are designed. These agreements can lead to standardized interfaces.

The authors of [21] present a survey of component frameworks for embedded systems, they point out two main difficulties in the development of component systems. The authors also present the evaluation criteria for a real-time component model for embedded systems and compare the frameworks presented against the given criteria. Component frameworks for industrial domain are also presented: *THINK* [24], *MIND* [25] (based in *THINK*) and *SOFA HI* [26]. The classification

TABLE VII. TYPE OF RESEARCH AND CONTRIBUTION CLASSIFICATION

Contribution Facet					
Metric	Tool	Model	Method	Process	
	[12, 9, 10, 11]	[12, 10, 11]	[12, 9, 11, 13, 17]	[10, 13, 16, 19]	
Research Facet					
Evaluation Research	Validation Research	Philosophical Paper	Experience Research	Opinion Paper	Solution Proposal
[12, 13, 17]	[9, 10, 13]	[11, 16]	[12, 9, 10, 11, 16, 19]		[12, 9, 10, 13, 17]

criteria and review of the frameworks are very enlightening in the sense that reading this work provides a great deal of insight into component frameworks from various perspectives of application.

Authors in [13] consider component based development as a key promising technology in embedded research domain. Here the authors point out the differences that make component model solutions for general purpose computers not viable to embedded systems. A series of component models for embedded systems in industry (both based in software engineering and control theory best practices) are pointed out. From our experience in recent European projects, industrial component models need to look into disciplines, such as IIoT and machine learning. Beyond control, embedded systems of today smart factories must analyse data, communicate with vendor independent hardware (sensors, machines, actuators, cloud systems and HMI devices) and take actions.

Rubus [12] is a component model for embedded systems. This work regards industrial requirements that were elicited considering mixed timing and resource constrained requirements. The components in this solution also have a set of modes and/or a set of states that allows the components to execute distinct code for different system states.

Authors in [8] present a good list of reasons that motivates a component model specific for field devices. A case study in which a single board computer containing the *PECOS* solution and controlling a motor speed was developed in their work. This involved a component for representing a speed sensor and others to encapsulate control algorithms that were specifically developed for this case. The board had both web-access protocols (HTTP over TCP/IP) and an interface for an industrial protocol(*ModBus*). This solution show how components can be passive, in the sense that they are invoked by a scheduler or other components; or they can be active, own a thread to process asynchronous events or perform long computations in background.

V. Conclusion

To draw more realistic conclusions commercial and other academic and non-academic solutions, which were of our knowledge, but not found during the search phase, should be considered in the evaluation and mapping. Some of them are *Matlab/Simulink* [27], *Node-RED* [28], *Scade* [29], *OSGi* [30] and *4DIAC* [31]. In addition, to make the study reproducible, it is important to mention that intuitive findings (such as when analysing papers and consulting other informal search engines and databases) were not included.

Some interesting conclusions can be taken from the design considerations over time in the graph of Figure 3. There are only two papers considering security issues, the second one [5] is about a component model designed for cyber-physical systems, in which security is a hot-topic. In the same classification line, real-time considerations are shown to prevail over the years. This finding can somewhat confirm that this is a hard subject to tackle in component architectures. Intelligent reasoning is an emergent topic of nowadays, we decided to include that design consideration in the classification scheme of Table VI, exactly to make readers perceive that only in most recent paper of interest [5] it was addressed. This also could mean that security and artificial intelligence are open topics of research in the software engineering component models domain. As we have seen, there are multiple works using *IEC 61499*, it seems to be the de facto standard for component syntax and semantics in industrial automation. Other concerns that seems to prevail are the communication, design and portability of components. Last but not least, apart from commercial and other non-academic solutions, it seems that this topic is not evolving in the recent years. This can also be a signal that the emergent software engineering methodologies for industrial automation [6] are capturing a lot of attention from the academic community.

References

[1] K. Petersen, R. Feldt, S. Mujtaba, and M. Mattsson, “Systematic mapping studies in software engineering.” in EASE, vol. 8, 2008, pp. 68–77.

[2] B. Councill and G. T. Heineman, “Definition of a software component and its elements,” Component-based software engineering: putting the pieces together, 2001, pp. 5–19.

[3] K.-K. Lau and Z. Wang, “Software component models,” IEEE Transactions on software engineering, vol. 33, no. 10, 2007, pp. 709–724.

[4] C. Maga, N. Jazdi, and P. Göhner, “Reusable models in industrial automation: experiences in defining appropriate levels of granularity,” IFAC Proceedings Volumes, vol. 44, no. 1, 2011, pp. 9145–9150.

[5] F. e. a. Fouquet, “A dynamic component model for cyber physical systems,” in Proceedings of the 15th ACM SIGSOFT symposium on Component Based Software Engineering. ACM, 2012, pp. 135–144.

[6] V. Vyatkin, “Software engineering in industrial automation: State-of-the-art review,” IEEE Transactions on Industrial Informatics, vol. 9, no. 3, 2013, pp. 1234–1249.

[7] RapidMiner, Inc. Rapidminer studio. Last accessed 2018.05.04. [Online]. Available: https://rapidminer.com/products/studio/

[8] T. Genßler, A. Christoph, M. Winter, O. Nierstrasz, S. Ducasse, R. Wuyts, G. Arévalo, B. Schönhage, P. Müller, and C. Stich, “Components for embedded software: the pecos approach,” in Proceedings of the 2002 international conference on Compilers, architecture, and synthesis for embedded systems. ACM, 2002, pp. 19–26.

[9] E. Farcas, C. Farcas, W. Pree, and J. Templ, "Transparent distribution of real-time components based on logical execution time," in ACM SIGPLAN Notices, vol. 40, no. 7. ACM, 2005, pp. 31–39.

[10] E. K. Jackson, E. Kang, M. Dahlweid, D. Seifert, and T. Santen, "Components, platforms and possibilities: towards generic automation for mda," in Proceedings of the tenth ACM international conference on Embedded software. ACM, 2010, pp. 39–48.

[11] W. Roll, "Towards model-based and ccm-based applications for real-time systems," in Object-Oriented Real-Time Distributed Computing, 2003. Sixth IEEE International Symposium on. IEEE, 2003, pp. 75–82.

[12] K. Hanninen, J. Maki-Turja, M. Nolin, M. Lindberg, J. Lundback, and K.-L. Lundback, "The rubus component model for resource constrained real-time systems," in 2008 International Symposium on Industrial Embedded Systems. IEEE, 2008, pp. 177–183.

[13] G. Doukas and K. Thramboulidis, "A real-time-linux-based framework for model-driven engineering in control and automation," IEEE Transactions on Industrial Electronics, vol. 58, no. 3, 2011, pp. 914–924.

[14] V. Vyatkin, "Iec 61499 as enabler of distributed and intelligent automation: State-of-the-art review," IEEE Transactions on Industrial Informatics, vol. 7, no. 4, 2011, pp. 768–781.

[15] J. C. Eidson, E. A. Lee, S. Matic, S. A. Seshia, and J. Zou, "Distributed real-time software for cyber–physical systems," Proceedings of the IEEE, vol. 100, no. 1, 2012, pp. 45–59.

[16] V. Tran, D.-B. Liu, and B. Hummel, "Component-based systems development: challenges and lessons learned," in Software Technology and Engineering Practice, 1997. Proceedings., Eighth IEEE International Workshop on [incorporating Computer Aided Software Engineering]. IEEE, 1997, pp. 452–462.

[17] M. Khalgui, O. Mosbahi, Z. Li, and H.-M. Hanisch, "Reconfigurable multiagent embedded control systems: From modeling to implementation," IEEE Transactions on Computers, vol. 60, no. 4, 2011, pp. 538–551.

[18] M. Khalgui, E. Carpanzano, and H.-M. Hanisch, "An optimised simulation of component-based embedded systems in manufacturing industry," International Journal of Simulation and Process Modelling, vol. 4, no. 2, 2008, pp. 148–162.

[19] D. Domis and M. Trapp, "Integrating safety analyses and component-based design," in International Conference on Computer Safety, Reliability, and Security. Springer, 2008, pp. 58–71.

[20] K.-K. Lau and Z. Wang, "A taxonomy of software component models," in 31st EUROMICRO Conference on Software Engineering and Advanced Applications. IEEE, 2005, pp. 88–95.

[21] P. Hošek, T. Pop, T. Bureš, P. Hnětynka, and M. Malohlava, "Comparison of component frameworks for real-time embedded systems," in International Symposium on Component-Based Software Engineering. Springer, 2010, pp. 21–36.

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

[23] H. J. Reekie and E. A. Lee, "Lightweight component models for embedded systems," in Published as Technical Memorandum UCB ERL M02/30, Electronics Research Laboratory, University of California at Berkeley. Citeseer, 2002.

[24] J.-P. Fassino, J.-B. Stefani, J. L. Lawall, and G. Muller, "Think: A software framework for component-based operating system kernels." in USENIX Annual Technical Conference, General Track, 2002, pp. 73–86.

[25] MINALOGIC. Mind: Assembly technology for embedded software components. Last accessed 2018.05.04. [Online]. Available: http://www.minalogic.com/en/minalogic/about-minalogic-0

[26] M. e. a. Prochazka, "A component-oriented framework for spacecraft on-board software," in Proceedings of DASIA. Citeseer, 2008.

[27] The MathWorks, Inc. Simulink - simulation and model-based design. Last accessed 2018.06.14. [Online]. Available: https://www.mathworks.com/products/simulink.html

[28] M. Blackstock and R. Lea, "Toward a distributed data flow platform for the web of things (distributed node-red)," in Proceedings of the 5th International Workshop on Web of Things. ACM, 2014, pp. 34–39.

[29] Esterel Technologies SA - A wholly-owned subsidiary of ANSYS, Inc. Scade suite - control software design — esterel technologies. Last accessed 2018.06.14. [Online]. Available: http://www.esterel-technologies.com/products/scade-suite/

[30] O. Alliance, "Osgi-the dynamic module system for java," 2009.

[31] T. Strasser, M. Rooker, G. Ebenhofer, A. Zoitl, C. Sunder, A. Valentini, and A. Martel, "Framework for distributed industrial automation and control (4diac)," in Industrial Informatics, 2008. INDIN 2008. 6th IEEE International Conference on. IEEE, 2008, pp. 283–288.