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 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 and Industrial Internet of Thing's. In this work, we surveyed Component Model solutions and literature applied to Industrial Cyber Physical Systems. By conducting a reproducible systematic mapping study, we search and select results of interest. Research Questions are formulated and addressed by applying classification schemes to the results. Finally, 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, a Systematic Mapping study in component models for embedded systems in industrial environments is addressed. All iterations of the systematic mapping process were based in [1] and are detailed along the document, finishing with results that consider the following research questions. 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 industrial Industrial Cyber-Physical Systems (ICPS) and the target are embedded systems?
2) **RQ2.** What are the similarities and design considerations among them?
3) **RQ3.** How research in this topic is evolving over time?
4) **RQ4.** What kind of contribution is given by particular papers?

A software architecture designed by component model solution is developed as "a composite of sub-parts rather than a monolithic entity" [3]. The advantages of such tackle many objectives of the software industry, some of them are: 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, that is: how components are composed 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 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.

The focus of our study are component models whose target are factory floor systems and 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], makes part of the traditional software engineering methodologies. From the key areas of software engineering [6], we will dive our attention in: software design and construction, configuration management, tools and methods.

We are interested in cover the various levels of components, from those who 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 composed using wrapping, static and dynamic linking, and "plug-and-play" methods. Component models typically are 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 Industrial Internet of Things (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 explained in Section IV. Section V concludes the paper with a final discussion.

II. Primary Search

The proposed systematic map steps in [1] are illustrated in Figure 1. The search 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 the abstracts and selected a first set of documents based on the criteria of Table I. Because every research topic have a specific terminology that is 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 research questions 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 - there was added the ones outlined by dashed lines of Figure 1. Instead of only base the research questions on the first search to build one search string, the keywords and abstracts of the accepted documents were used to find frequent words and produce a new search string. To generate the new search string, the *RapidMiner Studio* [7] tool was used to count frequent words, along with 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 that, the resulting set of keywords contained 44 keywords of interest. Because performing combinations with this set was a time consuming process, we tried to query the selected databases with the entire set at a time and none of them accepted such a long query. After that, we decided to try *Google Scholar* search engine, which accepted the long set of keywords and resulted in very accurate preliminary results. We decided to merge the last results 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 can 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, we searched for each individual paper of the first set in *Google Scholar* and calculated the multiplicative factor between the number of citations in the second set. Finally, we calculated the average of all multiplicative factors and by applying that value to replace minimum number of references considered in Table I (for this case), we came up with a minimum number of 36 references for *Google Scholar* results to be considered in the second set.

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
Abstract & Intro. Analysis (Duplicates)	5	4	4	5	1
Final Set	18				

The results of applying the Exclusion and Exclusion Criteria specified in Table I drastically reduced the number of documents, as can be observed in Table II. The final set of documents was used to conduct the evaluation. For that a classification scheme was used to combine with the mapping process, this process is detailed in the next chapters.

III. Mapping Process

The following works are the ones of 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 61449 [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 them does not provide enough details to fill all the classification schemes proposed 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 its built in a builder environment, 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 the components are fetch 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 results found. The first classification scheme is based in results found on: opinion, survey, taxonomy and classification frameworks, and philosophical findings. The second scheme is based on available and existing solutions (both commercial and academic) with documentation reporting experiments, validation and use cases. The third classification, which is based in previous ones, specifically addresses RQ3. The last classification scheme addresses RQ4, the categories used are based in [1].

From the extended set of relevant papers, not only can be applied to all classification categories. Theoretical and Survey papers does not apply to the choose taxonomy for component models.

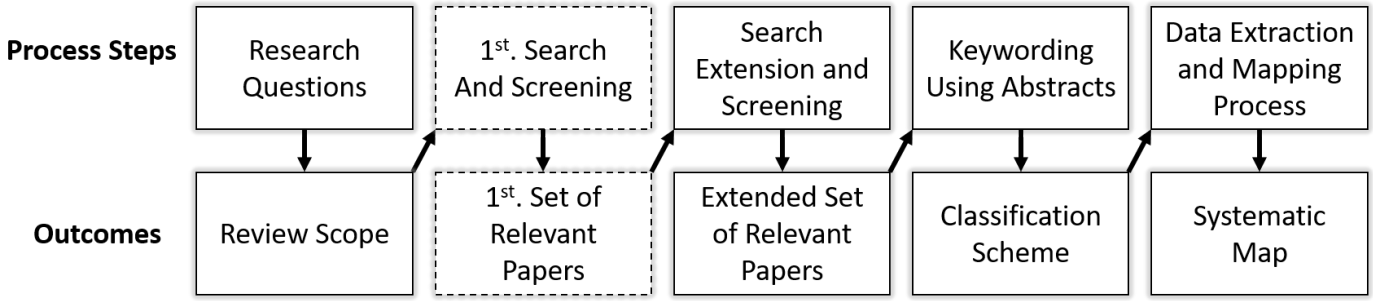

Figure 1. Modified Systematic Mapping Process.

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

- **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 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 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, 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]

Regarding the Composition classification, the original taxonomy [20] defines 5 characteristics of composition that gives origin to categories. 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.

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	✓

2) Design Considerations: Reading through the analysed papers a characteristic perceived as of the highest importance is the time characteristics of modules. Parallelism, (a)synchronism, worst case 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 results in fragile tool chains which 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, but having a reduced capability of reuse. The footprint of components and the container run-time environments is a recurrent concern when developing to embedded systems, which are typically cons. 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 monolithic system.

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,

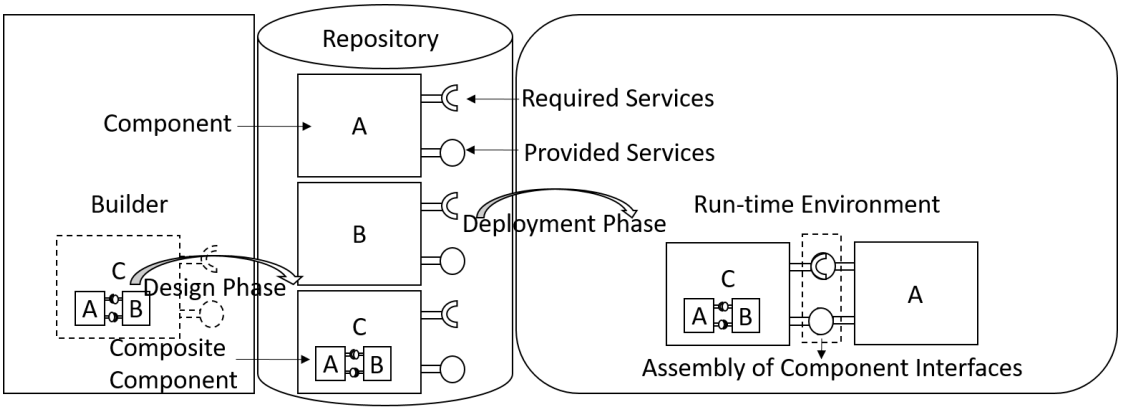

Figure 2. Component Model Overview.

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]

some conclusions can be drawn. This classification addresses RQ3.

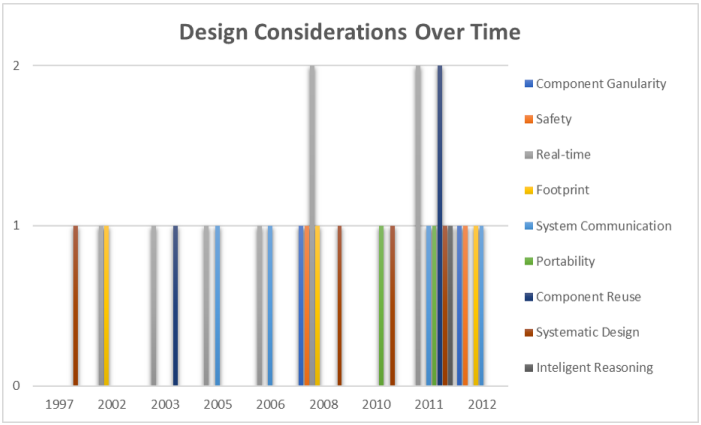

Figure 3. Design Considerations Over Time.

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

IV. Results 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 was to gain some insight in the details of the solutions found. There are some design considerations typical of industrial scenarios that this solutions address 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 result found in the systematic mapping study. The authors conclude that a theory that supports component model process in the whole life-cycle did 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 criteria and review of the frameworks are very enlightening in the sense that reading this work provides a great deal of insight in component frameworks from various perspectives of application.

Authors in [13] consider component based development as a key promising technology in embedded research domain. Here authors poit out the differences that make component model solutions for general purpose computers (other articles) not viable to embedded systems. A series of component models for embedded systems in industry, based in software engineering and control theory best practices are pointed out. From our experience in recent European projects, industrial

TABLE VII. Type 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]

component models need to look into disciplines, such as IoT and big data. 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, regarding industrial requirements that are elicited, under mixed timing and resource constrained requirements. The components in this solution have also a set of modes or a set of states, which allows the components to execute different code in different states.

Authors in [8] present a good list of reasons for motivation of a component model for field devices. In this work, a case study based on a board containing the PECOS solution and controlling a motor speed, using a speed sensor and control algorithms was developed. The board has web-access and thus is compatible with industrial (ModBus) and Web protocols (Ethernet). 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 (https://www.mathworks.com/products/simulink.html), Rational Statement (http://www-03.ibm.com/software/products/en/ratistat), Node-RED [27], Scade (http://www.esterel-technologies.com/products/scade-suite/), OSGi [28] and 4DIAC [29]. In addition, to make the study reproducible, 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 **??** 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 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 appli-

cations 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] 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.
[28] O. Alliance, "Osgi-the dynamic module system for java," 2009.
[29] 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.