2017 43rd Euromicro Conference on Software Engineering and Advanced Applications

An Ensemble-Based Approach for Scalable QoS in Highly Dynamic CPS

Vladimir Matena
Faculty of Mathematics and Physics
Charles University, Czech Republic

Alejandro Masrur
Department of Computer Science
TU Chemnitz, Germany

Tomas Bures
Faculty of Mathematics and Physics
Charles University, Czech Republic

***Abstract*—Modern cyber-physical systems (CPS) often involve distributed devices/components that closely interact with each other and their environment. In this context, operation conditions may constantly change and it is not always possible to guarantee quality of service (QoS), particularly, if resources degrade or stop being available. In addition, sometimes, one would like QoS to scale up/down with operation conditions, e.g., maximize efficiency, minimize energy consumption, etc. without compromising safety. However, traditional design and development techniques fail to capture the dynamics of modern CPS, since they rather focus on individual components/devices, and are unable to provide such QoS guarantees. To overcome this problem, we propose a design methodology based on the concept of ensemble, i.e., a dynamic grouping of components, which allows for scalable QoS guaranties. We illustrate the utility of our approach based on a case study consisting of an intelligent production line and analyze the effect on performance as communication between components degrades. Finally, our methodology can be incorporated into existing ensemble-based tools such as DEECo, Helena or jRESP to generate executable code to be deployed onto distributed devices.**

I. Introduction

We are concerned with the design and development of open-ended cyber-physical systems (CPS), i.e., where *components* may join/leave the system arbitrarily. Due to the high dynamics and complexity of such systems, there is a need for suitable programming abstractions that allow for a clean design while meeting quality of service (QoS) and safety requirements.

Traditional design and development techniques from embedded systems focus on individual components and largely fail to describe (dynamic) interactions among them. As a result, they are not really suitable for open-ended CPS. Recently, a number of approaches have been proposed within the software engineering community focusing on modeling interactions between components rather than components in isolation: DEECo [1], Helena [2], and jRESP[3]. However, these operate at a high level of abstraction making it difficult to model/express QoS on their basis.

On the other hand, usually, there are varying conditions in dynamic CPS — e.g., resources may degrade or stop being available, the number of components/devices may increase, etc. As a result, there is a need for scalable QoS metrics that can be evaluated at runtime and adapt to changing operation conditions. The idea is that some level of functionality can be provided in spite of degrading operation conditions and, in particular, without comprising safety.

Contributions. We make use of the concept of *ensemble*, i.e., a dynamic/spontaneous grouping of components, to propose a design methodology for scalable QoS. Our methodology consists in identifying safety conditions that result from both the used *cyber platform*, i.e., computation and communication processes, and the *physical world*. Similarly, we derive QoS requirements — translating into utility — that dynamically scale up/down to adapt to changing operation conditions.

We illustrate our technique on the basis of a case study consisting of an intelligent production line (IPL) where human and robot workers collaborate towards a common goal. As detailed later, safety conditions need to be derived to avoid collisions between robots and humans. QoS is characterized by the speed with which robots are allowed to move around. Clearly, the more workers there are in a robot's surroundings, the more difficult it will be to guarantee a high speed.

Both safety conditions and QoS requirements are modeled by components with well-defined roles and ensembles, i.e., rules that are evaluated at runtime, and can be incorporated into available ensemble-based tools to obtain executable code that can be deployed onto devices. In this paper, for the sake of illustration, we make use of DEECo [1]; however, our technique can also be used in combination with other similar tools such as the aforementioned Helena [2] and jRESP [3]. We further perform a number of simulations using OMNeT++ to illustrate our IPL case study and to investigate its performance with respect to degrading communication.

Structure of the Paper. The rest of this paper is structured as follows. Related work is discussed in Section II. Next, Section III introduces our IPL case study. Our proposed method is introduced in Section IV , whereas Section V shows our evaluation results and Section VI wraps up the paper.

II. Related Work

There is a clear trend to use component models for the design and development of complex systems, since they provide a comfortable level of abstraction. There exist numerous such models with different properties and application domains [4][5]. However, not all of them support dynamic architectures and open-endedness (i.e., allowing adaptivity and reconfiguration) and real-time behavior (i.e., providing deterministic running times) as required by highly dynamic CPS.

AUTOSAR [6], BlueArx [7] by Bosch, Koala [8] by Philips, and IEC 61499 [9] are component-based frameworks used in

978-1-5386-2141-7/17 $31.00 © 2017 IEEE
DOI 10.1109/SEAA.2017.62

industry. However, they do not really allow for open-endedness as they were conceived for static components with reduced support for dynamic architectures and/or self-organization.

Looking into models that do not specifically target embedded systems, the BIP (Behavior, Interaction, Priority) framework [10] enables timing analysis at the model level. BIP supports real-time aspects by using *timed components*, which allow for timing properties being specified using *timed variables* and *transitions*. These are accounted for during validation, however, composition in BIP is static not providing any support for open-ended systems.

Finally, there are a number of tools such as DEECo [1], Helena [2], and jRESP [3] that build on the concept of autonomic component ensembles. Ensembles are groups of components described by rules that form dynamically at runtime and are the basis of the proposed methodology. To the best of our knowledge, this is the first attempt towards achieving scalable QoS guarantees in the context of open-ended CPS.

III. Case Study

We consider a case study in the context of Industry 4.0, which consists of an intelligent production line (IPL) aiming to increase the level of automation and, thereby, optimizing efficiency and energy consumption. Currently, although robots already play a key role in production lines, they perform well-defined tasks and usually are mounted/installed at fixed positions. In future, however, robots are expected to gain more autonomy and potentially be able to perform a larger set of tasks among which they can spontaneously switch.

Whereas robots lift weights, manipulate dangerous objects/chemicals, among others, they usually lack versatility and ability to take *common-sense* decisions. Hence, modern production lines, rely on human workers for these tasks. Since humans and robots are expected to increasingly share the same physical space, it is important to guarantee safety and prevent collisions.

To this end, humans wear sensors that report their current positions. The system then dynamically computes a robot's trajectory depending on the positions (and speeds) of human beings and of other robots. Different QoS — translated into the speed with which robots move around to perform their assignments — can be guaranteed depending on the number of humans and robots in the surroundings and the quality of communication.

Mode of Operation. There are different regions of interest as depicted in Fig. 1. A robot is first confined to its *home zone*, where it can move freely. Each robot has its own home zone, however, home zones can overlap. In contrast to robots, humans are not restricted to any zone.

Whenever human or robot workers enter the home zone of a certain robot, this starts processing their status information (in particular, its current position) with a period p_{home}. The robot stops processing other workers' data whenever these latter leave its home zone.

A second region of interest is a robot's *proximity zone* defined in a radius of r_{prox} meters around the robot. In contrast to the home zone, the proximity zone changes with the position of the robot. Status information from any human/robot worker — the guest — entering another robot's — the host — proximity zone is updated with a higher frequency of p_{prox}.

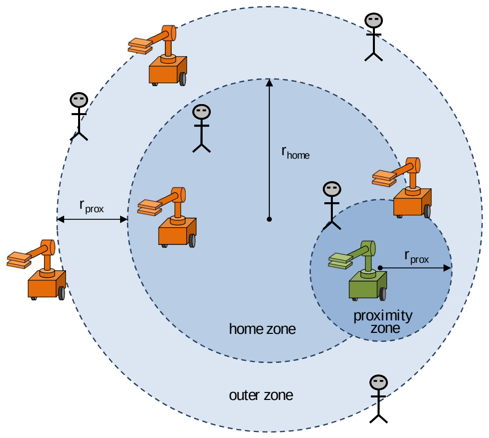

Fig. 1. Intelligent production line: home, proximity, and outer zones

An *outer zone* is an extension of the home zone by r_{prox} meters. From Fig. 1, it follows that workers in the outer zone can be in the robot's proximity zone. In this case, the robot needs to process the corresponding status information also with a period of p_{prox}.

Finally, note that workers always broadcast their status information every p_{prox} time and that a host (robot) only processes this data when workers are in its regions of interest.

IV. Proposed approach

While safety needs to be guaranteed under all circumstances, QoS can be scaled up/down to maximize utility under changing operation conditions. Our proposed design methodology consists in identifying safety conditions and scalable QoS requirements, which depend on the target application, and then formulating them as processes executed by components and ensembles that are dynamically evaluated at runtime.

In the context of the above IPL, safety is defined as the collision-free circulation of robots along the production line, whereas utility refers to the speed with which robots move to reach their desired destinations to perform their tasks.

A. Safety conditions

We now derive safety conditions which translate into proper values of p_{home} and p_{prox}. To this end, we need to consider characteristics of (i) the underlying *cyber* platform, i.e., communication and computation delay, and (ii) the *physical* world, i.e., speed and position of human and robot workers, etc.

Cyber platform. Let us consider that the 802.11b wireless standard [11] is used, which is well-established communication protocol with support for ad-hoc operation, i.e., without needing to synchronize *nodes*, and a bandwidth of 11 Mbps.

Further, $32B$ (i.e., bytes) are considered to encode a worker's current position. As a result, the time necessary to

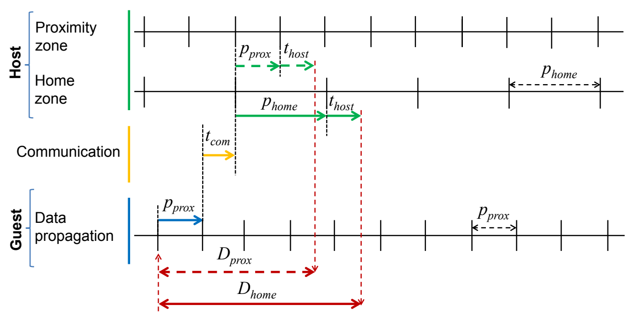

Fig. 2. End-to-end delays between host and guest workers. Delay for guests is D_{home} in the home zone and D_{prox} in the proximity zone.

deliver a data packet — denoted by t_{com} — can be computed as follows [12]:

$$\begin{aligned} t_{com} &= T_{data} + T_{SIFS} + T_{ACK} + T_{DIFS} + T_{BO} \\ &= 240\mu s + 10\mu s + 304\mu s + 50\mu s + 310\mu s \approx 1ms \end{aligned}$$

where $T_{data} = 192\mu s + \frac{(34+32)8\text{bits}}{11\frac{\text{bits}}{\mu s}} = 240\mu s$ is the transmission time for $32B$ of data under 802.11b with a $192\mu s$ header and $34B$ overhead, T_{SIFS} is the inter-frame space, T_{ACK} is the acknowledgment time, T_{DIFS} is distributed inter-frame space, and T_{BO} is back-off time.

In addition, there will a delay at the host robot denoted by t_{host} which accounts for the computation time of processes regulating speed and stopping the robot if necessary — taking actuators' reaction time into consideration. The processes computing the robot's trajectory are not included in t_{host}, since these are not relevant for safety.[1] We assume: $t_{host} = 5ms$.

We consider that processes at the host robot are executed periodically but in an asynchronous manner with respect to the environment, i.e., other workers. This allows for full flexibility and simplifies the design, since we do not need to synchronize with other events, e.g., no need to perform hand-shaking before transmitting data, etc. On the other hand, additional delay may be incurred by the system as shown in Fig. 2.

Particularly, a change in the status of one guest worker can reach the robot $t_{com} + p_{prox}$ time later . If something changes after data has just been propagated, this incurs additional p_{prox} waiting time until next propagation. At the host robot, the data might arrive after the robot has finished its processing and needs to wait until the next time processing is started. This might be a time p_{home}, if data is sent by guest in the home zone, or a time p_{prox} later, if data is sent by a guest in the proximity zone. Finally, t_{host} also needs to be considered.

This results in two possible end-to-end delays for guests, i.e., D_{home} in the home zone and D_{prox} in the proximity zone:

$$\begin{aligned} D_{home} &= t_{com} + p_{prox} + p_{home} + t_{host}, & (1) \\ D_{prox} &= t_{com} + 2p_{prox} + t_{host}. & (2) \end{aligned}$$

[1]In contrast, for safety, we are only concerned with stopping the robot to avoid collisions regardless of its trajectory, which can be computed separately.

Physical world. Wheeled robots with a maximum speed of $5m/s$ are considered. In addition, a human worker is assumed to move at a maximum speed of $1m/s$. A robot is considered to stop completely within $0.5m$ and the radius of the proximity zone is fixed to $r_{prox} = 5m$.

Regarding this setup, the worst case happens when two robots, with overlapping home zones, move towards each other at full speed. In such case the relative speed equals $10m/s$.

In the home zone, for the sake of safety, a guest is not allowed to cover a distance larger than $3m$ without sending a position update. Thus, the host needs to receive and process position updates in:

$$t_{3m} = \frac{3m}{10\frac{m}{s}} = 300ms.$$

As a result, in the worst case, the host robot detects a guest robot in the proximity zone when it is only $2m$ apart. It is necessary to guarantee that each robot can start breaking when it is at least $0.5m$ from another one to avoid collisions. This leaves time needed to cover $1.5m$ at $10m/s$ for taking the decision on breaking:

$$t_{1.5m} = \frac{1.5m}{10\frac{m}{s}} = 150ms.$$

Obtaining p_{prox} and p_{home}. It is easy to see that these two robots will collide if D_{prox} is greater than $t_{1.5m}$. As a result, we need to choose p_{prox} to be small enough that robots have time to react in this case. From (2) we have the following:

$$p_{prox} \le \frac{t_{1.5m} - t_{com} - t_{host}}{2} = 72ms.$$

The above value guarantees no collisions assuming reliable communication. However it does not leave any margin for packet losses. Lowering the value provides some resilience to this, but it also increases the communication overhead. To account for this, we select $p_{prox} = 30ms$.

With this value of p_{prox} we can compute a value for p_{home} using (2). For this, we know that D_{home} should not be more than t_{3m}, which leads to the following upper bound on p_{home}:

$$p_{home} \le t_{3m} - t_{com} - p_{prox} - t_{host} = 264.1ms.$$

This again is a upper bound on p_{home}, which leaves no margin for communication loss. Similar as before, we select $p_{home} = 200ms$ to balance resilience against packet losses and communication overhead.

B. Scalable QoS

A robot — the host — must be always able to stop before any obstacle represented by another robot or a human — a guest. To this end, the host's speed denoted by $\overline{v}_i$ needs to be adjusted accordingly. First the worst-case distance d_{ij} between the host i and a guest j in its proximity zone is computed by $d_{ij} = |dist(p_i, p_j) - t_{ij} \cdot \hat{v}_j|$, where p_i is the current position of the host, p_j the latest known position of the guest, t_{ij} is the age of the guest position, and $\hat{v}_j$ is the maximum speed of the guest. Here $dist(p_i, p_j)$ returns the shortest distance between p_i and p_j.

Now, to compute $\overline{v}_i$, the distance $\overline{d}_i$ from the host to the closest guest is used. Assuming G is set of all guest in the host's proximity zone, $\overline{d}_i$ equals: $\overline{d}_i = \min_{j \in G}\{d_{ij}\}$.

Since, in the worst case, a guest moves on a linear trajectory towards the host, $\overline{v}_i$ can be computed by $\overline{v}_i = \sqrt{2|a_i|\overline{d}_i}$, where $\overline{d}_i$ is a remaining distance to the closest guest, and $|a_i|$ is deceleration rate of the host [13].

The closer $\overline{v}_i$ is to $\hat{v}_i$, the higher the QoS/utility of the system. This depends on the operation conditions, in particular, on the number of guests and their positions which influence the value of $\overline{d}_i$, and hence dynamically scales with them.

C. Ensemble-Based Component Models

As already mentioned, the DEECo [1] is used to model the system and formulate ensembles for safety conditions and scalable QoS. In DEECo, components are runnable packages containing public data referred to as knowledge and processes that use that data. Processes are usually responsible for sensing, actuating, and running the control logic.

An ensemble is a dynamic group of components determined by a membership condition used to share parts of components' knowledge. The membership condition describes who should join the ensemble based on well-defined roles. Once components join an ensemble, they share knowledge using the ensemble knowledge exchange strategy.

In our IPL scenario, there are components representing humans and robots. With respect to ensembles, let us consider that (i) it is necessary to ensure coordination between a robot and other workers in its home zone, (ii) it is necessary to coordinate a robot with other workers in its proximity zone. This leads to ensembles *Home* and *Proximity*.

Listing 1 shows the *Robot* component — the *Human* component is omitted for the sake of brevity. A component is defined by its role using the keyword *features* in DEECo. In line 2, initial values of the knowledge fields are given. The rest of component consists of process definitions. A process can read and write knowledge fields as well as execute arbitrary code required to access sensors and actuators. The scheduling of the process can be either periodic with a fixed period or triggered by a knowledge change.

The process *UpdateTime*, at line 6 exemplify a sensor reading. The processes *EvaluateSafety* and *EvaluateQoS*, at lines 9 and 19, are responsible for evaluating safety and QoS. The *EvaluateSafety* process periodically assesses safety and stores the result in the *worstCaseDistances* knowledge field. In case of emergency the whole system can be stopped by setting the *emergencyStop* knowledge field. A change in the *worstCaseDistances* field triggers process *EvaluateQoS*, which computes the current QoS value, i.e., *maxSpeed*.

```
1  component Robot features RobotWorker
2    knowledge:
3      position, homeArea, speed, emergencyStop, guests,
4      workersInHomeArea, worstCaseDistances, maxSpeed
5
6    process UpdateTime:
7      function: time = System.getTime()
8
9    process EvaluateSafety:
10     function:
11       for worker in workersInHomeArea:
12         if now − worker.timestamp > SAFETY_TIME_THRESHOLD:
13           emergencyStop = True
14       worstCaseDistances = predictWorstCaseDistances(
15         union(guests, workersInHomeArea), position,
16         acceleration, deceleration)
17     scheduling: periodic(30 ms = p_prox)
18
19   process EvaluateQoS:
20     function:
21       maxSpeed = 5 ms^-1
22       for distance in worstCaseDistances:
23         maxSpeed = min(maxSpeed, getMaxSpeed(distance))
24     scheduling: triggered(knowledgeChange(worstCaseDistances))
```

Listing 1. Robot component

The *Home* ensemble contains only workers in the home zone of a particular robot. Each instance of this ensemble contains one robot and all the workers in its home area.

Finally, the *Proximity* ensemble defined in Listing 2 is evaluated among the components, i.e., robot or human workers, which are in the home zone of the corresponding robot. Note that the period with which *Proximity* is evaluated is the one computed in Section IV.

```
ensemble Proximity in Home:
  members: @exclusive Robot robot, @shared Worker worker
  membership:
    robot.position.distanceTo(worker.position) < PROXIMITY_DISTANCE
  knowledge exchange:
    robot.guests.put(worker)
  scheduling: periodic(30 ms = p_prox)
```

Listing 2. Proximity ensemble

V. Evaluation

In order to evaluate our approach, a simulation of the IPL case study was conducted. The focus was on analyzing the system behavior under loss of communication for which we have used OMNeT++ 5.0[2] framework and 802.11b network model provided by the INET 3.4 plugin. The source code of our simulation is available on GitHub.[3]

The simulated system encompasses all human and robot workers moving randomly without restrictions in a host robot's home zone. This represents the worst-case situation with all robots having fully overlapped home zones and human workers being present at the host robot's home robot.

Setup of the experiment. The simulated host robot's home zone has 25-meter radius and is filled with OMNeT++ *nodes* representing humans and robots. Each *node* is equipped with an 802.11b radio interface tuned to the same frequency and configured to form an ad-hoc network.

In total, four simulation runs were performed: (i) 3 humans, 7 robots, 300 seconds; (ii) 6 humans, 14 robots, 75 seconds; (iii) 12 humans, 28 robots, 20 seconds; and (iv) 24 humans, 56 robots, 5 seconds. The time interval simulated was adjusted to keep the simulation computation manageable.

Results. Figure 3 and Figure 4 show the results of simulation run (ii). In Figure 3, box-plots illustrate the communication delay as the distance between sender to receiver increases. This

[2]https://omnetpp.org

[3]https://github.com/d3scomp/scalable-reliability

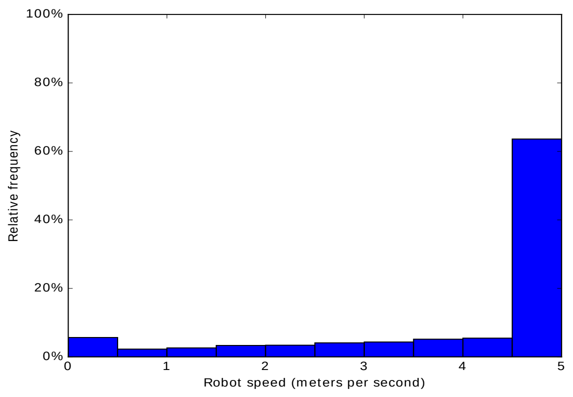

Fig. 4. Maximum allowed speed histogram, 6 humans, 14 robots

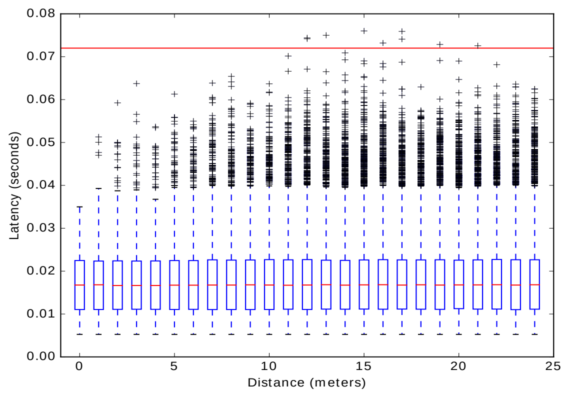

Fig. 3. Communication latency, 6 humans, 14 robots

delay becomes greater with the number of packets lost, which depend on the number of robots and humans transmitting data. The horizontal red line, stretching across the full width of the plot, represents the upper bound of p_{prox} as computed in Section IV-A, i.e., $72ms$. As it can be observed, in some situations p_{prox}'s upper bound is exceeded, which translates in a speed reduction for the corresponding robot.

Figure 4 shows histograms of the maximum speeds reached by robots in this simulation. A big number of robots and humans in the factory reduces the speed of robots in two ways. First, more radio traffic produces more data lost at the communication channel and, hence, robots need to slow down in order to deal with latency. Second, more robots and humans in the same area simply means shorter physical distances to *obstacles* resulting also in a speed reduction. On the other hand, on average, robots are able to achieve the maximum speed of $5m/s$ for about 63% of the overall simulated time.

For brevity the remaining simulation runs are summarized using system performance. Simulation runs (i) and (ii) represent slightly loaded systems with robots moving at their maximum speed 71% and 63% of the time. Simulation run (iii) is an inflexion point where operation at the maximum speed is reduced to only 30% of the time. Finally the simulation run (iv) represents a congested system where speeds close to zero dominate, however, our technique always allows guaranteeing safety, i.e., that no collisions occur between workers.

VI. Concluding Remarks

In this paper, we presented a design methodology for scalable QoS in the context of highly dynamic CPS. In this kind of systems, since it is not possible to guarantee a bound on QoS requirements, these are dynamically degraded or upgraded with changing operation conditions, e.g., new devices/components join arbitrarily, communication delay increases, etc. To this end, we build upon the concept of ensemble, i.e., a dynamic grouping of components that is evaluated based on the current state of the system. We illustrated our technique on the basis of a case study consisting of an intelligent production line and simulated it to demonstrate the utility of the proposed technique under different configuration settings.

VII. Acknowledgments

This paper has been partially supported by the Charles University Grant Agency project No. 391115 and project no. LD15051 from COST CZ (LD) programme by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] T. Bures, I. Gerostathopoulos, P. Hnetynka, J. Keznikl, M. Kit, and F. Plasil, “DEECo: An ensemble-based component system,” in *Proceedings of the International ACM Sigsoft Symposium on Component-based Software Engineering (CBSE)*, 2013.

[2] R. Hennicker and A. Klarl, “Foundations for ensemble modeling – the Helena approach,” in *Specification, Algebra, and Software*. Springer, 2014.

[3] “jRESP – runtime environment for SCEL programs,” http://www.ascens-ist.eu/jresp/.

[4] J. Feljan, L. Lednicki, J. Maras, A. Petricic, and I. Crnkovic, “Classification and survey of component models,” Project DICES, Tech. Rep., Feb. 2009.

[5] T. Pop, P. Hnetynka, P. Hosek, M. Malohlava, and T. Bures, “Comparison of component frameworks for real-time embedded systems,” *Knowledge and Information Systems*, vol. 40, no. 1, 2014.

[6] “AUTOSAR: Layered software architecture,” http://autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[7] J. E. Kim, O. Rogalla, S. Kramer, and A. Hamann, “Extracting, specifying and predicting software system properties in component based real-time embedded software development,” in *Proceedings of the International Conference on Software Engineering (ICSE)*, May 2009.

[8] R. Ommering, F. Linden, J. Kramer, and J. Magee, “The Koala component model for consumer electronics software,” *Computer*, vol. 33, no. 3, 2000.

[9] L. Lednicki, J. Carlson, and K. Sandström, “Model level worst-case execution time analysis for IEC 61499,” in *Proceedings of the International ACM Sigsoft Symposium on Component-based Software Engineering (CBSE)*, Jun. 2013.

[10] A. Basu, M. Bozga, and J. Sifakis, “Modeling heterogeneous real-time components in BIP,” in *Proceedings of the IEEE International Conference on Software Engineering and Formal Methods (SEFM)*, Sep. 2006.

[11] “IEEE standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications,” http://standards.ieee.org/about/get/802/802.11.html.

[12] J. Jun, P. Peddabachagari, and M. Sichitiu, “Theoretical maximum throughput of IEEE 802.11 and its applications,” in *IEEE International Symposium on Network Computing and Applications (NCA)*, April 2003, pp. 249–256.

[13] E. Osers and W. Westphal, *A Short Textbook of Physics: Not Involving the Use of Higher Mathematics*. Springer Berlin Heidelberg, 2012.