2014 17th Euromicro Conference on Digital System Design

Towards Component-Based Design of Safety-Critical Cyber-Physical Applications

Alejandro Masrur,† Michał Kit,* Tomáš Bureš,* and Wolfram Hardt†
*Faculty of Mathematics and Physics, Charles University, Czech Republic
†Department of Computer Science, TU Chemnitz, Germany

***Abstract*—Cyber-physical systems typically involve a large number of mobile autonomous devices that closely interact with each other and their environment. Standard design and development techniques from the embedded domain fail to accurately model the dynamics of such systems and, hence, there is an increasing need for new programming models and abstractions. Component-based design approaches are a promising solution to manage the complexity of large-scale dynamic systems. However, existing such approaches either do not accurately model transitory interactions between components – which are typical of cyber-physical systems – or do not provide guarantees for real-time behavior which is essential in many safety-critical applications. To overcome this problem, in this paper, we present a component-based design technique based on DEECo (Dependable Emergent Ensembles of Components). The DEECo framework allows modeling large-scale dynamic systems by a set of interacting components. In contrast to other component-based design approaches from the literature, DEECo provides mechanisms to describe transitory interactions between components. We introduce necessary extensions to the DEECo design flow and integrate it with real-time analysis techniques that allow reasoning about timing behavior at the component-description level. Finally, we illustrate the simplicity and usefulness of our approach on a case study consisting of an intelligent crossroad system.**

I. Introduction

Complex cyber-physical systems (CPS) can be found in many different domains such as smart traffic and transportation, intelligent buildings, smart grid, etc. A common aspect of such CPS is that they rely on a large number of autonomous, typically mobile, embedded devices that form an ecosystem. The joint cooperation of devices within this ecosystem provides functionality, which is unattainable by the individual devices in isolation.

A CPS is inherently distributed and adaptive – i.e., it constantly reacts to changes in its environment by adapting its structure and behavior. The collaborative aspect of the system as well as the necessity that its parts can function autonomously (in case the connection to other system constituents gets lost) poses a new dimension of challenges.

Typically, these challenges are regarded as separate problems of communication networks, distributed control, etc. As such, they have been addressed separately in the respective research fields. However, a significant aspect of modern CPS, which remains relatively overlooked, is that these systems can be also classified as software intensive [1].

This means that most of their functionality is embodied in the software, which in turn becomes their most complex and critical constituent. Due to being distributed and adaptive, software becomes even more complicated and the system starts exhibiting so-called *emergent behavior*. This is the situation where the system's behavior cannot be inferred any longer from its individual constituents, but their interplay and their joint influence on their environment have to be taken into account.

As a result, there is a strong need for *holistic* design and development methods that focus rather on the whole ecosystem and its overall behavior. Especially, these methods have to provide systematic software engineering practices that allow managing the increasing complexity of such systems and help controlling emergent behavior. By systematic software engineering, we envision the following four aspects:

i) the high-level (requirements-oriented) design with special focus on autonomous behavior, adaptivity and distributed collaboration;
ii) architectural design where a system is modeled by distributed components with clear responsibilities and well-defined interaction patterns;
iii) framework for implementation of components, which keeps the straightforward traceability w.r.t. (i) and (ii); and
iv) methods for design-time and runtime analysis (e.g., functional verification, timing analysis), which would predict and control the adaptivity and related emergent behavior of these systems.

The basic prerequisite for such systematic software engineering is the existence of software models providing an appropriate level of abstraction. In this respect, classical existing software models (e.g., component-based models such as AUTOSAR [2] or formal process models such as Petri Nets [3]) largely fail to address the needs of distributed adaptive systems. This is mostly because of the fact that they rely on a static structure of the system and thus are unable to model an *open-ended* system, which adapts its architecture to the current state of its environment.

On the other hand, new software models (such as DEECo [4], GCM [5]) appear gradually. They have been specifically developed to capture the nature of distributed adaptive systems and are more suitable for the design and development of CPS. However, by focusing on the cooperative aspects and dynamics of components, they *operate* at a high level of abstraction and they do not provide means to model real-time behavior – which is particularly relevant to safety-critical applications.

Contributions: In this paper, we bridge the gap between a high-level (component-based) description of the system and the analysis of its real-time behavior. In particular, we make use of

978-1-4799-5793-4/14 $31.00 © 2014 IEEE
DOI 10.1109/DSD.2014.87

DEECo (Dependable Emergent Ensembles of Components) in the context of a safety-critical cyber-physical application, viz., an intelligent crossroad. DEECo allows for a component-based design of highly dynamic CPS and provides deterministic semantics that help guaranteeing real-time behavior. Based on the crossroad scenario, we derive necessary extensions to the DEECo design flow in order to capture the real-time requirements at the component description level. Towards this, a schedulability or real-time analysis is performed taking application and implementation details (such as communication bandwidth, processing capacity, etc.) into account.

The paper is structured as follows. Section II presents the intelligent crossroad case-study that we use as the running example. Section III overviews the basic concepts of DEECo and illustrates them on the case-study. Section IV extends the DEECo model by including real-time constraints, which are then utilized to perform the schedulability analysis proposed in Section V. In Section VI, we evaluate our approach by comparing it with results obtained by an OMNet++ simulation of our case study. Section VII provides the related work, whereas in Section VIII we discuss some conclusions.

II. Case-Study

We consider an application scenario in the context of Vehicular Ad-hoc Networks (VANets) [6] and semi-autonomous driving, where an intelligent crossroad system (ICS) optimizes its *throughput* by assisting drivers in traversing a road crossing. This is illustrated in Fig. 1, where cars approach a two-lane crossing managed by the ICS.

The idea is to replace traffic lights by car-to-infrastructure (C2I) communication and to synchronize in what order cars cross the intersection. This way, the ICS takes control over approaching cars – adjusting speed and steering – to allow for an uninterrupted flow in all directions. After the intersection, control over vehicles is returned to the drivers.

The ICS adjusts the speed and direction of each car such that the highest possible *throughput* is reached provided that speed limit regulations are observed and safety of all traffic participants is guaranteed under all circumstances. Pedestrians are not considered in this setting[1].

We define the *region of influence* of the ICS by the area in which it controls all approaching cars. We assume that this area consists of a 50m radius around the intersection. In addition, vehicles are prioritized such that their priorities increase as they get closer to the focal point of the intersection and drop when they move away.

The scenario exhibits different challenges that one has to face when designing dynamic distributed systems. One of those challenges is the description of architectural changes that occur during runtime. In the scenario vehicles arrive to and leave the system at different points in time, their priorities vary according to their distance to the crossing, etc. Such details need to be properly reflected in the system design.

[1]The main issue with pedestrians is that they cannot be controlled by the ICS and, hence, hinder semi-autonomous driving at the crossroad. There might be different solutions for this problem such as planing a pedestrian underway, or using mobile phones to communicate with the ICS, etc.

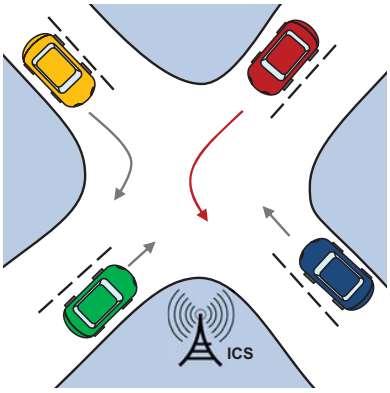

Fig. 1. Intelligent crossroad system (ICS)

Furthermore, the scenario exhibits real-time requirements imposed on the system. In particular, it is required that the *response time* between a car and the ICS is kept below a certain upper bound in order to ensure required responsiveness of the overall system. In turn, meeting those real-time requirements allows us to guarantee safety of the final application, which translates to a collision-free crossing.

III. Architectural Modeling

To address challenges related to the architecture of the above application, we propose using of the DEECo component model [4]. It features the structure of a CPS by means of components (i.e., encapsulated well-defined active entities, which perform sensing, computation and actuation) and so called ensembles, which are dynamically established groups of components that cooperate to achieve a particular goal. DEECo further provides a special requirements engineering method and traceability of requirements to components and ensembles – for further details, we refer to [7].

A. Components

To illustrate the principles behind DEECo, Listing 1 depicts a component using a DSL (Domain-Specific Language) description. In DEECo, each component consists of *knowledge* – see lines 8-15 – reflecting its current state. Knowledge is expressed by attributes organized into hierarchical data structures. Access to one or more such attributes of a component is performed through an interface – see lines 1-7 – that are featured by the component.

In addition, each component has a set of processes (essentially real-time tasks) that manipulate its knowledge. A process is characterized by a function (lines 29-35), whose parameter list consists of knowledge attributes. The scheduling and execution of processes is automatically managed by the *runtime framework*, which also takes care of knowledge retrieval before a process is executed and its update when the execution is finished. Each of the component's processes is executed in isolation meaning that it is not supposed to communicate with other processes (either of the same component or different one) in any other way than via component's knowledge.

The scheduling of a process depends on its specification (lines 20 and 35), which describes whether a process should be executed in response to a timer event (periodic execution)

or as a reaction to a value change in one of its attributes. In our example, the *Vehicle* component has a process that takes speed sensor readings and updates the corresponding attribute in its knowledge. This is repeated periodically every $5ms$ (line 20). As another example the ICS process, shown in Listing 1, determines whether there are privileged vehicles in its region of influence (lines 29-35) and is executed whenever the number of vehicles within the crossing changes (line 35).

```
1  interface MovingUnit:
2      position, route, throttle, privileged, speed
3
4  interface MovingUnitAggregator:
5      movingUnits
6
7  component Vehicle features MovingUnit
8      knowledge:
9          battery: 90%,
10         position: GPS(...),
11         route: [GPS(...)],
12         privileged: FALSE,
13         speed: 40 km/h,
14         throttle: 20%,
15         ...
16     process readSpeed:
17             out speed
18         function:
19             speed = Sensors.getSpeed()
20         scheduling: periodic( 5ms )
21     ...
22
23 component ICS features MovingUnitAggregator
24     knowledge:
25         location: GPS(...),
26         movingUnits: [...],
27         privileged: [...],
28         ...
29     process findPrivilegedVehicles:
30             in vehicles, inout privileged
31         function:
32             for (v : vehicles)
33                 if (v.privileged)
34                     privileged.add(v)
35         scheduling: triggered( changed(movingUnits) )
36     ...
```

Listing 1. DEECo component definitions based on a DSL

B. Ensembles

An ensemble in DEECo is a first class concept that defines a semantic connector between components and constitutes their *composition*. The composition in DEECo is *flat* and occurs implicitly by components dynamically joining an ensemble at runtime. When specifying an ensemble, prospective components are described by roles. One component in the ensemble has a *coordinator's* role, whereas the remaining components are *members* of the ensemble.

The roles are defined by the interfaces – in our example, *MovingUnit* and *MovingUnitAggregator* – and are matched at runtime to the actual components (i.e., their knowledge) for a structural coincidence. Later, those components that match the interfaces are considered for the ensemble evaluation process, which is composed of two steps. The first step involves checking the *membership condition*, which is expressed as a logic predicate formulated upon coordinator's and member's attributes. This determines whether two components (a coordinator and a member component) should form an ensemble. The second step depends on the results of the membership condition check and consists of exchanging attribute values between coordinator and member according to the description given in the *knowledge exchange* specification.

In the example in Listing 2, the coordinator role is determined by the interface definition *MovingUnitAggregator* and the member role by *MovingUnit*. This way, during the ensemble evaluation, only components featuring appropriate interfaces will be considered. The membership condition further constraints the number of ensemble members only to those, which are located no farther than 50m from the coordinator location. Then, according to the knowledge exchange description, the coordinator's *movingUnits* attribute is updated with information about all components that fulfill the membership condition. This way, the ICS is aware only of those vehicles, which are currently in its close proximity.

```
1  ensemble UpdateMovingUnitInformation:
2      coordinator: MovingUnitAggregator
3      member: MovingUnit
4      membership:
5          distance(coordinator.location, member.position) < 50 m
6      knowledge exchange:
7          coordinator.movingUnits.add({member})
8      scheduling: periodic( p_ens )
```

Listing 2. A DSL example of an ensemble definition.

C. DEECo's deterministic semantics

DEECo components are autonomous and rely only on data that is present in their knowledge. As mentioned before, any interaction of a component with other components is realized by ensembles, which is, in a sense, *externalized* from the component itself. This property of DEECo's component model suits very well to both design and implementation of distributed adaptive systems as all technical aspects related to communication between remote components can be abstracted away from the design phase and left to the runtime framework to deal with them.

Technically, the runtime framework addresses the distribution by periodically propagating its ensemble-relevant knowledge to all other nodes in the system[2]. In our case study, ensemble-relevant data are the car's position, its speed, and its intended direction. This is used to evaluate the distance to the crossroad and to decide whether cars are heading in its direction.

Each node then keeps relevant reference knowledge from all other nodes or components in the system. This way, since ensemble-relevant information is present at all runtime instances or nodes of the system, the DEECo runtime framework performs local evaluation of an ensemble membership condition. If it holds, the (local) reference data of the remote components involved is used for the knowledge exchange process.

IV. Real-Time Constraints

Clearly, real-time constraints stem from the application requirements. To illustrate this process, in this section, we identify those requirements in our ICS case study and analyze the implications for the component description in DEECo.

In order for the ICS to control a car, knowledge needs to be exchanged from the car to the ICS and from the ICS to the car at least once per meter of the car's trajectory. (For this, clearly, a maximum speed of the car needs to be forced by

[2]Note that gossip or broadcast algorithms might be used for this purpose.

the ICS). However, the knowledge exchange happens based on *local data* when the corresponding ensemble condition is evaluated to true at both the ICS and the car node separately.

As discussed above, the knowledge propagation and the ensemble membership check happen periodically. Let us denote by p_{pro} the period with which knowledge is propagated and by p_{ens} the period with which ensembles are evaluated on all nodes. For simplicity, we assume that p_{pro} and p_{ens} are equal for all nodes in the system. Note the presented analysis can be easily extended to the more general case where p_{pro} and p_{ens} vary from node to node.

Since these two processes are not synchronized with one another, we have the following conditions in the worst case:

i) A car propagates its knowledge to the ICS immediately after a membership evaluation has been performed at the ICS. As a result, data is received p_{ens} time later at the ICS, when the next membership evaluation is performed.
ii) In a similar manner, the ICS propagates its knowledge to the corresponding car just after a membership evaluation has been performed in the car. As a result, data is received p_{ens} time later at the car too.
iii) The knowledge of the car changes immediately after knowledge has been propagated to the ICS. As a result, the *current* knowledge is propagated with a delay p_{pro} from the car to the ICS, when a new propagation is performed.
iv) The ICS's knowledge changes immediately after knowledge has been propagated to the car. Hence, the *current* knowledge is not propagated until a new propagation is started p_{pro} time later.

As a result, in the worst case, we have a delay due to the asynchronous nature of the DEECo framework which is given by: $2 \times p_{pro} + 2 \times p_{ens}$.

In addition, there is also a process running at the ICS which computes the new values of speed and steering for the car and is triggered when a knowledge exchange is executed at the ICS. We denote by r_{ICS} the worst-case response time (WCRT) of this process. Analogously, there is a process running in the car, which applies the new speed and steering values to the *physical* car. This process is also triggered when a knowledge exchange happens in the car and its WCRT is r_{car}[3].

As a result, the worst-case delay D_{max} for a closed-loop reaction in DEECo, i.e., a reaction of the car to an input of the ICS – which is computed based on the car's current knowledge – is given by:

$$D_{max} = 2 \times p_{pro} + 2 \times p_{ens} + c_{ICS} + c_{car} + r_{ICS} + r_{car}, \quad (1)$$

where c_{car} is the communication delay from a car to the ICS and c_{ICS} is the communication delay from the ICS to the car. Since there is interference by other messages (from other cars), c_{car} is the maximum possible communication delay in the network. However, from the ICS to the car, there is no interference – assuming a full-duplex communication channel – and the communication delay c_{ICS} is equal to the transmission time, since the ICS does not compete for accessing the network.

[3]The purpose of this paper is to illustrate the use of DEECo to model timing constraints at a component level. The implementation details of the different processes running at the ICS and in different cars are beyond scope.

Now, to guarantee safety, a car needs to be provided with new speed and steering values at every single meter of its trajectory. If the car's speed is at maximum 50Km/h (assuming an urban scenario and considering that the ICS controls the speed in its region of influence), we need to know the time $t_{1\text{m}}$ that it needs to cover 1m of its trajectory:

$$t_{1\text{m}} = \frac{1\text{m} \times 3600\text{s/h}}{50 \cdot 10^3\text{m/h}} = 72\text{ms}. \quad (2)$$

As a result, the following condition must hold, in order that the ICS system can be safely implemented in DEECo:

$$D_{max} \leq t_{1\text{m}}. \quad (3)$$

Note that in (1), r_{ICS} and r_{car} strongly depend on the workload at the ICS and in the car respectively. Similarly, c_{car} and c_{ICS} are given by the message scheduling and transmission on the network. In contrast to this, p_{pro} and p_{ens} can be configure to meet (3). In the next section, we explain how to compute r_{ICS}, r_{car}, c_{car}, and c_{ICS}.

V. Schedulability Analysis

In order to design safety-critical applications in a component-based fashion, the implementation of the runtime framework has to allow a deterministic behavior. Clearly, the runtime framework needs to rely on specific technologies that make real-time scheduling and real-time communication possible. In this section, we specify what conditions have to be fulfilled in order to guarantee a correct timing behavior of the overall system.

A. Processing Nodes

The DEECo runtime framework is executed on each node in the system and is in charge of scheduling component's processes. For this, a real-time scheduling algorithm needs to be provided by the operating system. Since processes are triggered periodically, we assume the scheduling is performed in line with the *rate monotonic* policy [8]. That is, processes are assigned fixed priorities according to the following rule: the shorter the period the higher the priority.

Now let us denote by $\mathbf{T}$ the set of processes on a given node. Further, τ_i is a process that belongs to $\mathbf{T}$ where e_i denotes its worst-case execution time (WCET) and p_i denotes its period of repetition. Without loss of generality, let us further assume that all τ_i in $\mathbf{T}$ are sorted in order of non-decreasing p_i (and hence non-increasing priority), i.e., if $j < i$, then $p_j \leq p_i$. For $\mathbf{T}$ to be schedulable, the following has to hold for each τ_i and $1 \leq i \leq |\mathbf{T}|$, where $|\mathbf{T}|$ is the number of elements in $\mathbf{T}$:

$$\sum_{j=1}^{i} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \leq p_i. \quad (4)$$

This expression means that for each process τ_i to be schedulable (and, hence, for $\mathbf{T}$ to be schedulable), the sum of all executions of higher-priority processes in a time interval equal to p_i plus its own execution e_i should be less than its deadline p_i. Note that (4) is sufficient but not necessary. A sufficient and necessary test can be achieved by response time analysis

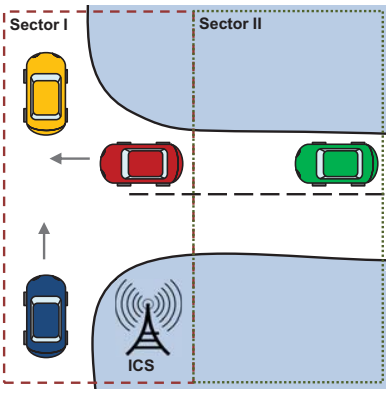

Fig. 2. Priorities are given according to the proximity to the intersection

[9]; however, the sufficient test of (4) is enough for the purpose of this paper. Now, since the following holds:

$$\sum_{j=1}^{i} \left\lceil \frac{p_i}{p_j} \right\rceil e_j \quad \leq \quad \sum_{j=1}^{i} \left(\frac{p_i}{p_j} + 1 \right) e_j,$$

we can reshape (4) to:

$$\sum_{j=1}^{i} \frac{e_j}{p_j} + \frac{\sum_{j=1}^{i} e_j}{p_i} \leq 1. \tag{5}$$

B. Communication Network

A number of techniques have been already proposed to realize collision-free communication in VANets [10], [11], [12], [13]. In addition, since VANets are usually based on wireless communication [14], we assume that Ethernet IEEE 802.1Q is the underlying protocol. This provides mechanisms to prioritize *messages* [15].

In general, we will have a number of access points (AP) which are connected to a full-duplex switch via Ethernet. Again, the communication to the AP is collision-free. Assuming that wireless network provides 100Mbps and that messages are at most 1Kbit (1024 bits), then the transmission time c_W on the wireless network is at most – considering a 144-bit protocol overhead:

$$c_W = \frac{1024 + 144}{100Mbps} = 11.68\mu\text{s}.$$

However, the switch then sends messages to the ICS according to their priorities. Let us consider that the ICS's region of influence is divided into sectors with different priorities – see Fig. 2. Cars that are in the first sector (e.g., within 10m from the intersection) have higher priority than cars in the second sector (e.g., from 10m to 20m) and so on. The switch then sends messages to the ICS according to these priorities.

Let us now analyze the communication segment between the switch and the ICS. To this end, let $\mathbf{M}$ denote the set of all messages being sent to the ICS over the switch. Further, m_i denotes one such message in $\mathbf{M}$ where c_i is its transmission time – note that c_i is constant for a given i which results from the amount of bits to be sent and the bandwidth of the communication channel – and z_i denotes the minimum inter-arrival time between two consecutive such messages. The deadline of a message is also given by z_i.

Let us assume that all m_i in $\mathbf{M}$ are sorted in order non-increasing priority, i.e., if $j < i$, then m_j has higher priority than m_i. For all messages in $\mathbf{M}$ to meet their deadlines, the following has to hold for $1 \leq i \leq |\mathbf{M}|$, where $|\mathbf{M}|$ is the number of elements in $\mathbf{M}$:

$$b_i + \sum_{j=1}^{i} \left\lceil \frac{z_i}{z_j} \right\rceil c_j \leq z_i, \tag{6}$$

where b_i denotes *blocking time* on the network. That is, whenever a message needs to be sent, a lower-priority message might eventually be using the communication channel. Since this lower-priority message cannot be interrupted, there is a blocking time on the bus. Clearly, in worst case, b_i is given by the maximum transmission time among all lower-priority messages:

$$b_i = \max_{l=i}^{|\mathbf{M}|}(c_l). \tag{7}$$

Similar as before, we can approximate (6) as follows by removing the ceiling function:

$$\sum_{j=1}^{i} \frac{c_j}{z_j} + \frac{b_i + \sum_{j=1}^{i} c_j}{z_i} \leq 1. \tag{8}$$

Let us assume that the bandwidth of the Ethernet link between the switch and the ICS is equal to 1Gbps. Now since we consider a message length of at most 1Kbit (1024 bits) and a protocol overhead of 144 bits, we have that $c_i = c_E$ for all messages m_i:

$$c_E = \frac{1024 + 144}{1Gbits} = 1.168\mu\text{s}.$$

Finally, in addition to the transmission time, there is always a delay at the AP and at switches in Ethernet, which accounts for buffering and routing tasks. This is typically in the order of 2μs.

C. Computing c_{car}

We compute c_{car} as the maximum delay necessary to send a message from a car, over the network, to the ICS. For this let us analyze the delay of the lowest priority message in the system. This is the message of a car being still away from the intersection. In addition to all higher priority messages, we assume that all messages of the same priority are sent before, which leads to the greatest possible communication delay. Note that messages are sent by components to the ICS every p_{pro} time units and also from ICS to the cars at the same rate – we consider a full-duplex communication, such that it is possible to send and receive messages simultaneously in both directions.

To know how many messages are sent at maximum over this network, we need to compute the maximum number of cars in the system. To determine this number, let us assume that a car is at least 2m long and that there is a least a 1m distance between any two cars. As a result of this, in the worst

possible case, the number of cars n approaching the crossing from all directions is given by the following equation:

$$n = 4 \times \left\lceil \frac{50\text{m}}{3\text{m}} \right\rceil = 68. \quad (9)$$

Now we can compute the maximum communication delay between ICS and switch denoted by r_E. This can be obtained from (6) or its approximation (8). For simplicity, we choose using (8) as shown below:

$$r_E = p_{pro} \times 68 \times \frac{1.168\mu\text{s}}{p_{pro}} + 68 \times 1.168\mu\text{s} = 158.85\mu\text{s}, \quad (10)$$

where c_i and z_i have been replaced by c_E and p_{pro} respectively, and b_i is zero according to (7). Recall that we have two Ethernet links: one from the AP to the switch and another from the switch to the ICS. Finally, being $e_{AP} = e_{SW} = 2\mu$s the delay on the switch and AP respectively, c_{car} can be obtained by:

$$c_{car} = 2 \times r_E + c_W + e_{SW} + e_{AP} = 333.38\mu\text{s}. \quad (11)$$

Clearly, since c_{ICS} is the transmission delay on the network without interference, this is given by:

$$c_{ICS} = 2 \times c_E + c_W + e_{SW} + e_{AP} = 18.02\mu\text{s}. \quad (12)$$

D. Obtaining p_{ens} and p_{pro}

There will at most 68 different ensembles (between the ICS and each of the cars) on the ICS – see (9). In addition, there will be 68 processes to compute speed and steering values for each car. Since the ensemble membership check triggers a knowledge exchange when evaluated true, we can assume that in worst case all 68 ensemble processes trigger their corresponding computation processes simultaneously. In addition, there will be one knowledge propagation process for the ICS[4].

Assuming that all the processes have a WCET $e_i = 25\mu$s (note that most these processes consist in checking logic conditions, assigning pointers to given memory spaces, etc., or are simple control algorithms), we can use (5) applied to the ICS so as to compute p_{ens} and p_{pro}:

$$\frac{0.025\text{ms}}{p_{pro}} + 68 \cdot \frac{2 \times 0.025\text{ms}}{p_{ens}} + \frac{(0.025\text{ms} + 68 \cdot (2 \times 0.025\text{ms})}{p_{ens}} \leq 1. \quad (13)$$

Finally, we obtain the following value for p_{ens} assuming $2 \times p_{pro} = p_{ens}$, i.e., that knowledge propagation is done twice as frequently as any ensemble membership check[5]:

$$p_{ens} \geq 6.88\text{ms}.$$

We choose $p_{ens} = 14$ms – twice as much to relax the workload on the ICS – and hence $p_{pro} = 7$ms.

[4]Note that we assume that the knowledge propagation from cars does not produce any overhead at the ICS, but in the respective cars.

[5]This is a design decision that needs to be taken. In general, since ensemble membership checks rely on local knowledge, it is meaningful that knowledge be updated as often as necessary to guarantee desired functionality.

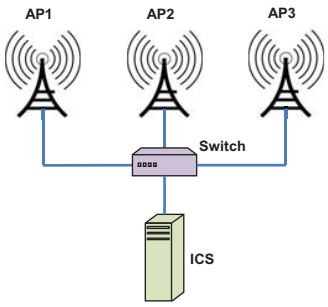

Fig. 3. Simulated network consisting of three access points (AP) and a switch

E. Testing Schedulability

To test schedulability of the system, we need to verify that (3) holds. For this it is necessary to compute r_{ICS} and r_{car}, i.e., the WCRTs of the ICS and the car process respectively. With the values of p_{ens} and p_{pro} that we have obtained before, we can compute r_{ICS} using again (5).

$$r_{ICS} = 14\text{ms} \times \left(\frac{0.025\text{ms}}{7\text{ms}} + 68 \cdot \frac{2 \times 0.025\text{ms}}{14\text{ms}} \right) + 0.025\text{ms} + 68 \cdot (2 \times 0.025\text{ms}) \approx 7\text{ms}. \quad (14)$$

Similarly, we can compute r_{car} using (5). In the car, there are only one ensemble process, one process to update the speed and steering values with those values obtained from the ICS, and a knowledge propagation process. Again, we assume the ensemble process triggers the update process at cars. Assuming that $e_i = 25\mu$s holds, we obtain:

$$r_{car} = 14\text{ms} \times \left(\frac{0.025\text{ms}}{7\text{ms}} + \cdot \frac{2 \times 0.025\text{ms}}{14\text{ms}} \right) + (0.025\text{ms} + 2 \times 0.025\text{ms} = 0.18\text{ms},$$

and, finally, we have from (1):

$$D_{max} = 2 \times 7\text{ms} + 2 \times 14\text{ms} + 0.3334\text{ms} + 0.0181\text{ms} + 7\text{ms} + 0.18\text{ms} \approx 50\text{ms},$$

which is less than $t_{1m} = 72ms$ – see (2). That is, our ICS is able to meet all deadlines in the worst case.

VI. Evaluation

In this section, we validate the analysis presented above by simulation. To this end, we created an OMNet++ simulation [16] using *INET* hardware models. Our network topology consists of one ICS host connected by a full-duplex switch to three AP – see Fig. 3.

At the AP we consider a collision-free communication protocol as per some VANet technique [10], [11], [12], [13]. The communication from the switch to the ICS host is performed under message prioritization according to 802.1Q. The simulation scenario spans different numbers of vehicles (20, 50 and 70 correspondingly) exchanging Ethernet packets with the ICS. Vehicles connect dynamically to the AP adjusting message priorities as they get closer to the intersection. The following list summarizes the most important simulation parameters considered in our evaluation:

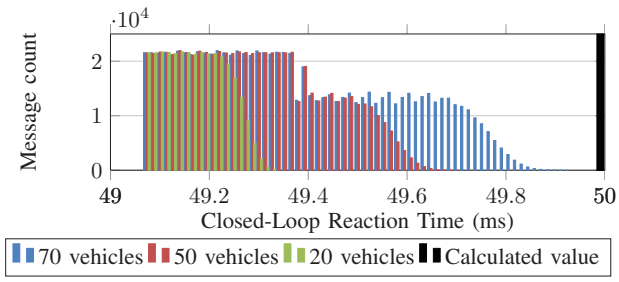

Fig. 4. Car-ICS-Car closed-loop reaction times

Priority levels	7
Message length	1024bits
Packet send interval	$7ms$ (p_{pro} from the analysis)
ICS response delay	$p_{pro} + p_{ens} + r_{ICS} = 28$ms
A car response delay	$p_{pro} + p_{ens} + r_{car} = 21.18$ms
Bandwidth (Car to AP)	100Mbps
Bandwidth (ICS to AP)	1Gbps

	20 vehicles	50 vehicles	70 vehicles
Mean	49.1714	49.2961	49.5333
Std. Dev.	0.06058	0.1411	0.1912
Variance	0.00367	0.01992	0.0365
1st Quartile	49.1161	49.1829	49.4368
Median	49.1696	49.2898	49.5704
3rd Quartile	49.223	49.3967	49.6773
Max	49.3299	49.704	49.9311

Fig. 5. Closed-loop reaction time statistics

Figures 4 and 5 show the results of the simulations with respect to the closed-loop reaction time. These indicate that the computed D_{max} is safe, i.e., that all delay values in the system are always less than D_{max} even for 70 cars, i.e., two more cars than what it is considered and allowed by the analysis presented in the above sections.

VII. Related Work

To position the presented approach among a multitude of the existing component models, we constrain our focus to those that enable analyzing timing aspects.

The most prominent example is certainly AUTOSAR [2], which is of common use in the automotive industry. AUTOSAR serves as a specification for different layers (i.e., application software, runtime environment and basic software) of a system constituted by hierarchical components. AUTOSAR itself does not provide any means to perform timing analysis and for that reason it has been enriched by the TIMing MOdel (TIMMO) [17], which builds on the Timing Augmented Description Language (TADL).

Another widely used model supporting timing analysis is AADL (Architecture Analysis and Design language)[18]. It relies on Real-Time Calculus (RTC) [19], which is a formalism that allows for system-level performance analysis of stream-processing systems constrained by hard real-time requirements. Essentially RTC models are extracted from AADL and subsequently the RTC tools can be employed.

Similarly, timing analysis enabled at the model level are supported by the BIP (Behavior, Interaction, Priority) framework [20]. BIP supports real-time aspects by using *timed components*, which allow for timing properties being specified using *timed variables* and *transitions*. Those are accounted for during the validation within the real-time engine implementing operational semantics of the BIP models.

An architectural approach to modeling systems is also taken by SysML, which integrates with MARTE [21] to enable modeling non-functional properties such as power consumption, performance and timing.

However, AUTOSAR, AADL, SysML and BIP assume static component architectures, which effectively prevents their application in case of modern CPS development. In contrast, our approach targets at open-ended CPS where the architecture changes continuously (e.g., cars appear and disappear without anticipation) and leads to emergent behavior in the system.

Another example of a component model that allows for timing estimates of a system is the Palladio Component Model (PCM) [22]. The strongest point of PCM is the extra-functional property prediction framework that allows estimating overall system performance. It relies on different models, depending on what is required to be analyzed (e.g., reliability, performance, throughput, etc.). These are decorated by non-functional properties specification, which serves as an input for model analysis. Similar to our approach PCM builds on simulation-based techniques for model validation, which allows exploring different designs for a given system or application. However, in a similar manner as the related approaches above, PCM does not support dynamic architectures, which again limits its applicability in today's complex CPS.

VIII. Concluding Remarks

In this paper, we have presented DEECo as a special-purpose, component-based, design and development framework for open-ended CPS. DEECo specifically targets at dynamic distributed systems and, thus, provides systematic software engineering mechanisms to describe and analyze such complex application scenarios. These mechanisms mainly consist in modeling transitory interactions between one or more components in the system.

We extended DEECo's design flow by a technique to estimate worst-case, closed-loop, response times between DEECo components. This effectively allows guaranteeing real-time requirements from high-level DEECo-based designs provided that the underlying platform supports real-time, e.g., real-time operating system, priority-based communication protocols, etc. Clearly, if the underlying technologies are nondeterministic, then it is not possible to provide any timing guarantees and, as a consequence, no safety-critical applications can be implemented on their basis.

We illustrated our proposed technique based on an intelligent crossroad scenario. Towards this, we derived the worst-case delay D_{max} of a DEECo-based system – see Eq. (1). This analysis is general enough and can be used for other applications. Note that the term $2 \times p_{pro} + 2 \times p_{ens}$ is the

overhead by DEECo, whereas $c_{ICS} + c_{car}$ and $r_{ICS} + r_{car}$ stand for the communication and the computation overhead respectively. DEECo's overhead is configurable by properly choosing p_{pro} and p_{ens} which again need to be in accordance with the application requirements. The communication and computation overhead will depend on the used technologies such as communication protocols, scheduling algorithms, etc.

We envision integrating the proposed technique into the existing ensemble development life cycle [23], which provides a systematic approach (i.e., methodology) towards engineering (ensemble-based) CPS. The presented work fits into the modeling part of that cycle, which is followed by verification performed on the basis of simulation techniques – similar to the procedure shown in this paper. An important aspect is also the requirements engineering part, which should besides functional properties also account for non-functional, in particular, real-time aspects.

Overall, the technique presented in this paper allows reasoning about real-time requirements at the component level and constitutes a necessary step towards holistic software engineering methods for modern cyber-physical systems.

ACKNOWLEDGEMENTS

This work was partially supported by Charles University institutional funding SVV-2014-260100. The research leading to these results has received funding from the European Union Seventh Framework Programme FP7-PEOPLE-2010-ITN under grant agreement n°264840.

REFERENCES

[1] K. Beetz and W. Böhm, "Challenges in engineering for software-intensive embedded systems," in *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*. Springer, 2012.

[2] "AUTOSAR: Layered software architecture," http://autosar.org/download/R4.0/AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf.

[3] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013.

[4] 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.

[5] F. Baude, "A perspective on the CoreGRID grid component model," in *Euro-Par 2011: Parallel Processing Workshops*. Springer, 2012.

[6] S. Zeadally, R. Hunt, Y.-S. Chen, A. Irwin, and A. Hassan, "Vehicular ad hoc networks (VANETS): status, results, and challenges," *Telecommunication Systems*, vol. 50, no. 4, 2012.

[7] J. Keznikl, T. Bures, F. Plasil, I. Gerostathopoulos, P. Hnetynka, and N. Hoch, "Design of ensemble-based component systems by invariant refinement," in *Proceedings of the International ACM Sigsoft Symposium on Component-based Software Engineering (CBSE)*, 2013.

[8] C. Liu and J. Layland, "Scheduling algorithms for multiprogramming in hard real-time environments," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, 1973.

[9] N. Audsley, A. Burns, M. Richardson, K. Tindell, and A. Wellings, "Applying new scheduling theory to static priority pre-emptive scheduling," *Software Engineering Journal*, vol. 8, no. 5, 1993.

[10] N. Shah, F. Bastani, S. Kumar, and I.-L. Yen, "Real-time car-to-car communication protocol for intersecting roads," in *Proceedings of the International Conference on ITS Telecommunications (ITST)*, 2008.

[11] S.-Y. Pyun, H. Widiarti, Y.-J. Kwon, D.-H. Cho, and J.-W. Son, "TDMA-based channel access scheme for V2I communication system using smart antenna," in *Proceedings of the IEEE Conference on Vehicular Networking (VNC)*, 2010.

[12] S.-Y. Pyun, H. Widiarti, Y.-J. Kwon, J.-W. Son, and D.-H. Cho, "Group-based channel access scheme for a V2I communication system using smart antenna," *IEEE Communications Letters*, vol. 15, no. 8, 2011.

[13] N. Shah, S. Kumar, F. Bastani, and I.-L. Yen, "Optimization models for assessing the peak capacity utilization of intelligent transportation systems," *European Journal of Operational Research*, vol. 216, no. 1, 2012.

[14] "IEEE 802.11p standard: Wireless LAN MAC and PHY specifications amendment 6: Wireless access in vehicular environments," http://standards.ieee.org/findstds/standard/802.11p-2010.html.

[15] "IEEE 802.1Q standard: LANs and WANs – MAC bridges and virtual bridged LANs," http://standards.ieee.org/findstds/standard/802.1Q-2011.html.

[16] A. Varga and R. Hornig, "An overview of the OMNeT++ simulation environment," in *Proceedings of the International Conference on Simulation Tools and Techniques for Communications, Networks and Systems (SIMUTOOLS)*, 2008.

[17] K. Klobedanz, C. Kuznik, A. Thuy, and W. Mueller, "Timing modeling and analysis for AUTOSAR-based software development - a case study," in *Proceedings of Conference on Design, Automation, and Test in Europe (DATE)*, 2010.

[18] O. Sokolsky and A. Chernoguzov, "Performance analysis of AADL models using real-time calculus," in *Foundations of Computer Software: Future Trends and Techniques for Development*. Springer, 2010.

[19] L. Thiele, S. Chakraborty, and M. Naedele, "Real-time calculus for scheduling hard real-time systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS)*, 2000.

[20] 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)*, 2006.

[21] H. Espinoza, D. Cancila, B. Selic, and S. Gerard, "Challenges in combining SysML and MARTE for model-based design of embedded systems," in *Model Driven Architecture - Foundations and Applications*. Springer, 2009.

[22] S. Becker, H. Koziolek, and R. Reussner, "The palladio component model for model-driven performance prediction," *Systems and Software*, vol. 82, no. 1, 2009.

[23] T. Bureš, R. D. Nicola, I. Gerostathopoulos, N. Hoch, M. Kit, N. Koch, G. V. Monreale, U. Montanari, R. Pugliese, N. Serbedzija, M. Wirsing, and F. Zambonelli, "A life cycle for the development of autonomic systems: The e-mobility showcase," in *Proceedings of the Workshop on Challenges for Achieving Self-Awareness in Autonomic Systems (AWARENESS)*, 2013.