Cyber-Physical Software Systems for Smart Worlds: A Case Study of Intelligent Transportation System

Kaliappa Ravindran

City College of CUNY and Graduate Center, Department of Computer Science, 160 Convent Avenue, New York, NY 10031, USA ravi@cs.ccny.cuny.edu

Abstract. The paper discusses the design of cyber-physical systems software around *intelligent physical worlds* (IPW). An IPW is the embodiment of control software functions wrapped around the external world processes. The IPW performs core domain-specific activities while adapting its behavior to the changing environment conditions and user inputs. The IPW exhibits an intelligent behavior over a limited operating region of the system — in contrast with the traditional models where the physical world is basically dumb. To work over a wider range of operating conditions, the IPW interacts with an intelligent computational world (ICW) to patch itself with suitable control parameters and rules/procedures relevant in those changed conditions. The modular decomposition of a complex adaptive system into IPW and ICW lowers the overall software complexity, simplifies the system verification, and promotes an easier evolution of system features. As an intelligence functionality, a network system in our approach employs redundant sensing as a means to improve the quality of detection & aggregation of events occurring in the environment. The paper illuminates our concept of IPW with case study of vehicular traffic management network.

1 Introduction

A cyber-physical system (CPS) allows the computational processes to interact with the physical world processes in a way to impact how the latter is structured and designed, and vice versa. We elevate the definition of an embedded system by eliminating the hardware-centric boundaries of physical processes. An application \mathcal{B} that is traditionally viewed as a non-embedded system because of its heavy software leaning can now be brought into the fold of CPS with a notion of *intelligent physical world* A_p . Here, A_p can be an embodiment of diverse software functions, with the embedded hardware instantiating the raw physical processes (RPP). The RPPs are dumb physical component ensembles through which a system interacts with its (hidden) external environment, such as: steering linkages to turn a car on the road, network link/router to transport data packets, and conveyor belt to move assembled parts.

The sub-system \mathcal{A}_p is more than a collection of physical components RPP, but instead consists of a software wrapper that controls RPP in such a way to infuse a self-contained and intelligent behavior¹. From a programming standpoint, the RPP is abstracted as a function $g^*(I, O^*, s^*, E^*)$ that takes an input I and responds with an output O^* , where s^* is the current state of RPP and E^* is the uncontrollable external environment incident on RPP. Here, O^* depicts the observation of a transition in the state of physical processes s^* , with the timescale of response hidden as part of the abstraction. For e.g., $q^*(\cdots)$ may represent the end-to-end path in a data network, where I and O^* denote the injection of a packet flow and its delivery respectively, s^* is the available bandwidth, and E^* depicts a packet-loss phenomenon impacting the flow. As another example, $g^*(\cdots)$ may be the motor in an industrial control system, where I and O^* denote the electrical signal and rotational speed respectively, s^* is the residual motor torque, and E^* depicts an electrical and/or mechanical disturbance impacting the motor speed. Our idea is to extend $g^*(I, O^*, s^*, E^*)$ into a coherent intelligent physical world A_p that is self-aware and can repair itself (in a limited way) from the damages caused by environment conditions E^* . A_p is augmented by an intelligent computational world A_c that manages the overall operations of A_p . Their composition to yield an adaptive application system \mathcal{B} is denoted as:

$$\mathcal{B} \equiv A_p \oplus A_c,$$

where A_p is wrapped around $g^*(I, O^*, s^*, E^*)$ and the operator ' \oplus ' depicts the inter-module flow of signals between A_p and A_c : which includes a managementoriented feedback from A_p to A_c . The signal flow is at a meta-level, while the A_c - A_p concrete interactions are determined by their programming boundaries. We allude to a 'monitor-and-control' interaction (M&C) initiated by A_c on A_p , and vice versa.

An example of A_p is a smart home that sets the heating and cooling parameters based on the occupancy, ambient conditions, comfort level, and the like. Here, A_c may be a Home Service Center outsourced with the task of managing the intelligent home remotely by setting the right parameters and operating procedures (say, different procedures for winter and summer operations). In a target tracking system as another example, the radar units reporting the images of objects in a terrain to a data fusion center may also notify the terrain characteristics to enable the choice of image processing algorithms: say, to meet the target detection accuracy needs. Here, A_p is the group of radar units implanted with parameter-adjustable image processing algorithms (say, track resolution) and A_c is the fusion center deciding on the right set of algorithms suitable for the terrain.

¹ The physical world A_p in our CPS view includes software functions that were hitherto a part of the control system software external to the RPP.

We extend the functional boundary of physical world A_p to infuse the intelligence for a limited repair capability. The remaining part of system, assigned with a comprehensive repair capability, constitutes an intelligent computational world A_c . The paper describes the software engineering issues in supporting a harmonious co-existence of intelligent sub-systems: A_c and A_p . The off-loading of domain-specific core adaptation functions into A_p enables the infusion of new functionalities and features in applications with less software complexity. The ease of verification and testing of such modularly structured systems lowers the development cost of distributed control software for complex systems.

The paper is organized as follows. Section 2 rationalizes the structuring of complex systems with intelligent physical worlds. Section 3 advocates the use of redundant sensing as a means to improve the quality of event detection (and hence the control actions therefrom). Section 4 provides a communication structure suitable for vehicular networks (say, in a city area). Section 5 discusses the existing frameworks for CPS. Section 6 studies a vehicular traffic management network using our CPS framework. Section 7 concludes the paper.

2 Our CPS view of complex systems

A traditional embedded system (TES) employs an asymmetric control relationship with the RPP: i.e., only the computational processes initiate the M&C interaction with RPP but not vice versa. The TES underscores an integrated software structure where the core adaptation functionality is entwined with high-level application features — which precludes rapid incremental software changes/configurations. In contrast, the CPS employs a modular software structure where a self-aware physical world A_p that is wrapped around the RPP communicates with a set of computational processes A_c to coordinate supervisory control by A_c . Figure 1 illustrates the difference between CPS and TES.

2.1 Existing designs cast through CPS view

Computational intelligence in the physical world requires the components to be self-aware, i.e., a component needs to be able react to its external environment — and possibly repair itself. Such an ensemble of self-aware components in A_p need to work together to provide a coherent interface to A_c . In this light, existing works on embedded control systems [1, 2] use an integrated structure (i.e., TES) that assigns intelligence for adaptation and reconfiguration only to the computational world, which exercises control on the physical world to cause effects in the external environment.

Given an adaptive application system \mathcal{B} , the TES-based design depicts a composition:

$$\mathcal{B}(tes) \equiv [A'_p \oplus g^*(I, O^*, s^*, E^*)],$$

where A'_p refers to the computational processes (implemented in software) that interface with the RPP function $g^*(\cdots)$. The composition \oplus depicts a M&C type of interaction, where A'_p invokes $g^*(\cdots)$ with a computed actuator signal I and



Fig. 1. TES versus CPS

observing the output response O^* . The TES structure assigns intelligence to A'_p , with the latter interfacing with $g^*(\cdots)$ through signaling hooks to actuate the trigger mechanisms, thereby moving the RPP move from one operating point to another. Thus, CPS-based design depicts an alternate system composition:

$$\mathcal{B}(cps) \equiv [A_c \oplus A_p] \supseteq \mathcal{B}(tes),$$

where $A_p \equiv [A_p'' \oplus g^*(I, O^*, s^*, E^*)]$ depicting the CPS software functions that wrap local intelligence around the raw physical world process over a limited operating region, such that $A_p'' \subseteq A_p'$. A_c is the computational process to infuse a broader intelligence to the operations of \mathcal{B} that are otherwise difficult in a TES-based design. $\mathcal{B}(cps)$ can easily be infused with new features and/or have its existing features augmented by algorithm plug-ins to modify the functionality of A_p , as orchestrated by policy-based mechanisms and adaptation logic programmed in A_c . For example, the QoS feature for packet transport over a network data path that hitherto allows controlling the mean packet delay can be augmented with delay jitter control as well, by implanting a modified packet scheduling algorithm along the path. The raw physical world $g * (\cdots)$ is itself considered as dumb, providing only the basic functional components. An invocation of these components comes from the upper layer processes: A_p'' in the CPS approach and A_p' in the TES approach.

Due to the underlying state-machine complexity of the system as a whole, the TES-based integrated approach does not lend itself well for a seamless addition/removal of automated system features, entails difficulty in incremental software changes, and makes the testing/maintenance of system software a laborintensive activity.

2.2 CPS-based structure of complex systems

We employ the principles of *piece-wise linearity* and *separability* of functions describing the system model [3], to determine the operating regions of A_p where the system-level computations of future trajectories (in a control-theoretic sense) are simpler and fall within the ambit of local intelligence. When the system behavioral changes satisfy linearity/separability, A_p can repair itself. The selfrepair can be via a local built-in mapping function that is instantiated with the parameters supplied by A_c for that operating region. An example is the adjusting of TCP flow control window size based on small changes in packet round-trip delay (RTT) over the transport network. On the other hand, if the behavioral changes are larger taking the system into non-linear regions, A_p may report the changes to A_c for the latter to adjust the parameters for the new region of system operations: say, by using *domain-specific policy* functions. In the TCP example, the protocol itself may be changed to aggressively adjust the window size when the RTT swings are large. A_c is wired with domain-specific policies and rules to evaluate the linearity and separability conditions, and then patch A_p with appropriate parameters and procedures².

 A_p operates over a much faster time-scale than A_c . This is because the control loop in A_p is self-contained to react to the smaller changes that typically occur frequently in the external environment. Whereas, A_c steps in only when larger changes occur in the external environment — which are less frequent (e.g., a network suffering a DOS attack, a car tire losing air due to a puncture). A_p embodies the core domain-specific functionality, and A_c is delegated with an external management role using parameterized procedures and rules specific to the domain. See Figure 2 for an illustration of the functional blocks to realize the hierarchical control relationship between A_c and A_p . Our software engineering approaches orchestrate such a delineation of A_p and A_c .

The true model of RPP may not be known to A_p , i.e., it is difficult to express $g^*(I, O^*, s^*, E^*)$ in a closed-form. So, the determination of I is governed by a computational model of RPP, denoted as $g(I, O^*, s, E)$, that is programmed into the controller module of A_p . This localized incremental adaptation strategy employed in A_p allows determining the final input I needed to attain a stable output P' — where $P' = O^*(L)$ with L depicting the control round when A_p reaches convergence. Any mismatch between P' and P_{ref} is then notified to A_c for appropriate recovery. The intelligent behavior of A_p is however feasible only over a limited operating region, as determined by A_c .

The system output O^* , which is of interest to the controller modules in A_p and A_c , is often easier to measure (e.g., packet transfer latency on a network path). The uncontrolled external environment E^* , which impacts the system output in complex ways, is however hard to measure (e.g., bandwidth depletion along the path). Our partitioning of observation space into O^* and E^* arises from these considerations. We assume a finite world where the parameter values

² The update of controller sub-systems in A_p during run-time is known as *patching* [4]. It enables a hierarchical control with simple controllers programmable at lower levels (such as automotive ECUs supplied by OEM vendors).



Fig. 2. Hierarchical control in our CPS structure

of E^* and O^* are bounded. A system designer may reduce output observation errors by exactly measuring $M^*(O^*)$ with suitable tools. Environment observation however is error-prone i.e. the observable environment space is: $E \subset E^*$.

2.3 Situational assessment feedback

Our approach is distinct from the well-known supervisory control methods [5]. The incorporation of a management-oriented feedback from A_p to a situational assessment module (SAM) housed in A_c allows the latter to adjust the control laws employed by A_p . The feedback is a notification about how successful A_p is in realizing the control delegated by A_c . A_p obtains a control reference parameter P_{ref} from A_c , along with domain-specific operating parameters and computational mapping functions (e.g., a rule to change the packet transmission window size for delay-adaptive flow control in TCP). A_p then generates appropriate inputs I to the RPP over multiple act-and-observe steps until the output O^* becomes stable: possibly, with a close match to P_{ref} .

That a final control error is stable but is not at the minimum depicts a controller with limited repair capability. When it is determined A_p has exceeded its repair capability at the current operating point, A_p seeks the services of A_c for a comprehensive repair, i.e., to bring down the error to a minimum. The comprehensive repair may involve, say, changing the plant and/or the controller parameters — and even the controller algorithm itself. Thus, a repair is collectively realized by A_p and A_c , with the invocations from A_p occurring infrequently on A_c in comparison to that on the RPP.

We focus on *action errors* in the controller of A_p , i.e., the deviations in actual RPP output from expected output $|O^* - O|$, arising from the inexact knowledge of controller about the computational model $g^*(I, O^*, s^*, E^*)$ of RPP. Regardless

of an error-prone or error-free output observation, action errors do occur, i.e., the output of RPP O^* as a result of executing an action I may deviate from the controllers belief about the effect of I, as captured by the model g(I, O, s, E). An³ error-free output observation, which we assume in this paper, yields an exact measurement of action errors — thereby allowing A_c to precisely evaluate the efficacy of controller rules/policies implanted in A_p , and install any changes therein.

2.4 Advantages of our approach

The observe-adapt cycle executed by A_p is at the machine-level time-scales pertinent to the RPP. The operations of A_c occur at much slower time-scales. The separation of time-scales in the operations of A_c and A_p makes it easier to assert the correctness of application behavior with a high degree of confidence. In TES-based design, the time-scale separation is not easily extractable from a trace-analysis of the state-transitions in application software, which lowers the designer confidence in making correctness assertions. In this light, our CPSbased modular techniques purport to reduce the overall system development cost during the evolutionary and operational stages of system designs, in the face of increasing complexity of system operations (both hardware and software) to meet the enhanced demands for new and better functionalities.

The patching of A_p from A_c enables the autonomic switching of control algorithms (at run-time) as the system operating points change. A_p can be supplied by designers with domain-knowledge (such as OEM vendors for in-vehicle electronic systems and network platform developers for inter-vehicle communications). A_p is designed to be programmable, with appropriate signaling hooks, while meeting the inter-operability requirements. Whereas, the designers of A_c are software engineers with more expertise on the management functions (instead of the domain itself). Some of the computational intelligence in A_p are enabled by new applications. A_p may also realize some of the functions hitherto in the TES-based computational world. The migration is possible due to the availability of data processing and storage capabilities in the physical components.

3 Management of distributed intelligent systems

From a service specification standpoint, the system performance, fault-tolerance, and timeliness goals can be unified into a single set of application-level QoS objectives. How well the application-level QoS specs are met in the presence of hostile external conditions depicts the dependability of the system.

³ Action errors in a complex system arise as an artifact of system modeling inaccuracy, which are different from the ones caused by software-induced bugs and failures [6].

3.1 Failure impact of system components

Given an ensemble of K devices in the infrastructure, A_c chooses N devices to participate in the algorithm execution of A_p for a collaborative task, where $2 \leq N \ll K$. An example is the reaching of consensus about an event occurrence. The choice of N is tied to an assumption made by A_c that at most f_m devices can fail at run-time and an attacked device exhibits a fault severity of r — where $1 \leq f_m < N$ and $0 < r \leq 1.0$. A failure may be benign or malicious, which may be (partly) captured in the fault severity parameters.

An intruder potentially targets f_a of the K devices for attacks to disrupt the system-level output, where $0 \leq f_a \ll K$. Furthermore, an attacked device exhibits a fault severity of r'', which depicts the probability of misbehavior by an attacked device when an input trigger occurs (r'') may be quantified in terms of how many operations the attacked device performs correctly before responding maliciously to an input trigger). The intruder does not have knowledge of systemlevel algorithm parameters $[N, r, f_m]$ (i.e., this information is protected in A_c): where N is the number devices participating in algorithm $(2 \leq N \leq K), f_m$ is the assumed number of faulty devices $(1 \leq f_m < \lceil \frac{N}{2} \rceil)$, and r is the assumed aggressiveness of a faulty device $(0 < r \leq 1)$. So, the intruder randomly targets the attacks on f_a devices and infuses a fault severity-level of r'' on an attacked device, in the hope of damaging the system output. The choice of $[f_a, r'']$ is based on the computational and other assets available at the intruder's disposal to orchestrate attacks and his/her empirical knowledge about the anticipated system-level damage caused by attacks.

Since $[r'', f_a]$ is not known to A_c , the algorithm designer needs to model the intruder's capability and profile to get a probabilistic estimate of $[f_a, r'']$. In general, the designer's decision about $[N, r, f_m]$ is based on his (domainspecific) knowledge about the overall system: namely, the operating environment of infrastructure and the control loops implemented by A_p .

3.2 Managing sensor redundancy and heterogeneity

Resorting to sensor heterogeneity in system measurements interplays with the control functions that rely on the accuracy and timely detection of events. A device D may run different algorithms $o_a = g_{(D)a}(M), o_b = g_{(D)b}(M), \cdots$ on a raw input data M (sequentially or concurrently), and then extract an accurate information o_a, o_b, \cdots about an event occurrence therefrom: say, by voting or outlier analysis on o_a, o_b, \cdots . Furthermore, D may survive against software errors and/or targeted attacks on a specific algorithm, say, $g_{(D)a}(.)$, because the other functions $g_{(D)b}(.), \cdots$ may continue running [20]. To survive against severe device-level failures (such as machine crashes and multiple attacks), a spatial replication of the device-functions is employed: such as $o_a = g_{(D_1)a}(.), o_b = g_{(D_2)b}(.), \cdots$. Figure 3-(A) illustrates the temporal and spatial redundancy to infuse survivability of the sensing process. If H is the number of heterogeneous devices, the system-level design complexity is $\mathcal{O}(H^2)$. Figure 3-(B) shows the cost of device replication from a system designer perspective.

Voting among N-replicated sensor devices provides an overall confidence level Γ that is higher than the per-device confidence level in the system, i.e., $\max(\{p_i\}_{i=1,2,\dots,N}) < \Gamma < 1.0$, where p_i is the confusion probability of D_i in



Fig. 3. Redundancy of sensor functions (temporal and spatial), and its cost

reporting an event. In an example of collision avoidance system for automobiles, a combination of sensors may be employed to detect the presence of road obstacles and fuse their results by voting (for improved vehicle safety) [19]. Likewise, multiple measurement tools enhance the accuracy of available bandwidth estimation on an end-to-end network path for a better video transport QoS. A voting-based improvement in the quality of event sensing is expressed mathematically as:

$$\left(1 - \binom{N-1}{l_1} \left[1 - p_i(e)\right]^{l_1 + 1 - l_2}\right) > \Gamma, \tag{1}$$

where l_1/l_2 are the number of consents/dissents about an event occurrence $o \in \mathcal{O}$ generated by D_i — assuming that all sensors have the same capability for event detection. For instance, $p_i = 0.85$ and N = 10 can achieve a confidence level of 98% with replica voting. In the absence of exact knowledge about the ground truth on system measurements, the confidence measured in the above manner can be used as an indicator of sensing accuracy. More generally, the operating point of A_p determines the weights assigned to the various replicated sensors for accurate determination of event o.

 A_c embodies computational intelligence methods [1] to implant the desired control algorithms in A_p that handle sensing errors as well: such as learning from past behaviors, sensor classification and calibration, and optimal control allocation to system components. The management of sensor heterogeneity is also handled by A_c .

4 Data aggregation in on-tree nodes

In this section, we describe the high level aggregation operations carried out by the on-tree nodes⁴. There are two reasons for the on-tree aggregation of events as they surface, instead of aggregating all the events at the root node. First, it enhances the scalability of event reporting system when large amounts of data are collected. Second, it entails a faster reaction to the events by overlay nodes as soon as a composite situation emerges that warrants an action (e.g., responding to traffic congestion events).

See Figure 4 for an illustration of the communication structure event notification. The overlay node at leaf point of the event aggregation tree maintains information about the capability of devices serviced by that node (such as encoding format, CPU speed, and display size). The node may, for instance, transcode the multimedia data describing an event for device-level rendering. The on-tree aggregation capabilities of overlay nodes is quite useful for vehicular networks (instead of doing only at end-point nodes).

4.1 Aggregation using syntactic rules

Let Θ_1 and Θ_2 be the confidence intervals of the data delivered at an overlay node O from its two downstream segments. With only a syntactic processing of the two distinct events, a confidence measure associated with the combined data sent by O to its upstream node is: $\min(\{\Theta_1, \Theta_2\})$.

Similarly, other types aggregation operators can be implemented in O such as addition, maximum, average, median, set union & intersection, selection, and the like. For instance, the congestion reports from two segments along the planned route of a car with projected delays d_1 and d_2 will simply lead to an estimate of the combined delay as $d_1 + d_2$ in traversing this route. Scalability considerations require that the syntactic composition operators satisfy the commutativity and associativity properties [17]. These properties allow an efficient examination of the events arriving asynchronously from various downstream nodes (by reducing inter-event synchronization delays).

An aggregation of events at various nodes in the tree typically affects the time-scale of changes in the resulting macro-level data. An example is to determine if there is a sustained packet loss in a multi-hop network (with k hops), based on the spatially separated per-hop measurements. The end-to-end loss is:

$$[1 - \prod_{i=1}^{k} (1 - l_i)],$$

where l_i is the measured packet loss in i^{th} hop. Since the 'loss composition' operator combines a set of fluctuating per-hop loss rates with independent modes

⁴ The data aggregation functions in on-tree overlay nodes and the communication functions between overlay nodes can be structured independent of that in the adhoc network segments at leaf nodes.



Fig. 4. Communication structure for event aggregation

at any given time, the end-to-end loss rate varies with a time-scale as determined by the highest mode in the per-hop loss rates. A spatial scale of changes may also be associated with event aggregations — such as the vehicular traffic congestion on a given route being the combination of the reported congestion levels in various stretches of roads along that route.

A domain-specific interpretation of the events in different regions cannot be adequately captured with the standard mathematical operators of aggregation — as argued in [2]. For example, the effect of a vehicle accident in one region on traffic congestions in the adjoining regions cannot be expressed through simple syntactic connectives. This motivates the need for a semantic knowledge in interpreting events.

4.2 Aggregation using semantic knowledge

Vehicular network applications often require abstracted measurements of the diverse environment phenomena (or events) in various geographic regions. These measurements need to be interpreted using a semantic relationship between the events (which may take into account the weak consistency and the temporal correlation among events [18]). Typically, the confidence level in the reporting of a combined event can be increased with a semantic knowledge that interconnects the two independently reported events.

As an example, consider the detection of a plane (in terms of speed and location) by the devices in region 1 followed by the detection of a plane by the devices in an adjacent region 2 after a certain time interval T. If the geographic distance between regions 1 and 2 depicts a flight time close to T at the given speed, then it is highly likely that the object detected in regions 1 and 2 refers to the same plane. So, when the detection reports from regions 1 and 2 arrive

at the overlay node O, the latter may aggregate them into a single report with a confidence measure higher than $\max(\{\Theta_1, \Theta_2\})$. The timing correlation in the two reports increases the confidence level of the combined report to higher than that of the individual reports.

Where semantic knowledge is used, the aggregation operations on two events may have to be carried out in a certain sequence (i.e., the operations may not satisfy the commutativity and/or associativity properties). Typically, each overlay node may implement the required synchronization between the arrival of various data items from its downstream nodes, based on the sequencing relationship between the data items — such as the causal relationship between events.

In a way, replica voting on fuzzy data (where the device-level confusion probability p_i satisfies the condition: $0.5 \ll p_i < 1.0$) may be viewed as a knowledgebased 'data aggregation' procedure executed at a leaf node. Here, the goal is to generate a single event notification with a base confidence measure that is higher than p_i . The semantic knowledge is that when two devices report the same datum with confidence levels of p_{i_1} and p_{i_2} , the leaf node can accept the datum with a confidence level higher than min($\{p_{i_1}, p_{i_2}\}$).

Latency measurements for voting-based data collection can provide the baseline timing information to enforce the synchronization of data, while meeting the overall timeliness constraints Δ . This however requires knowledge of the overlay tree topology and the data delays incurred in the various path segments.

5 Existing paradigms for CPS

At an abstraction level meaningful for applications, today's embedded systems embody both adaptation behaviors and functional behaviors. The former deals with adjusting the system operations according to the environment conditions (e.g., reducing the video send rate to deal with bandwidth congestion in the network). Whereas, the latter deals with requirements such as fault-tolerance, security, and timing. For system specification and analysis purposes, We treat the adaptation and functional behaviors separately. In this light, we categorize the existing works as dealing with:

- Systems engineering for the control-theoretic aspects of adaptation (such as stability, convergence) [7, 8];
- Software engineering for the verification of application requirements (including para-functional ones) [9, 10].

There have also been system-level tools developed to aid these studies: such as probabilistic monitoring and analysis [11], controlled fault-injection [12], and plug-in based model-solvers (e.g., SYSWeaver) [13].

Our work falls in a distinct category of model-based engineering of complex embedded systems. Our CPS model treats the adaptation processes in a target system as a black-box: A_p . The I/O mapping is procedurally realized by a sequence of sense-and-act steps executed by A_p on the RPP. A_c then incorporates the enhanced management functionality needed for complex systems (such as QoS assurance).

6 Case study: Vehicular traffic management

Vehicular networks often consist of computational devices, i.e., ECUs, that collect data representing the road traffic conditions and then generate traffic alerts for use by drivers. The data may include road traffic volume, terrain scenarios (e.g., hill tracks, slippery road), weather conditions, and vehicle motion tracking (e.g., car speed, inter-car spacing). These data, some of which constitute the external environment parameters E, are collected by various sensors mounted on the cars and the roadside, and then processed to generate corrective actions: say, traffic alerts and traffic re-routing.

6.1 IPW in vehicular traffic-flow system

The physical world is the road infrastructure itself, through which vehicular traffic flows. The topological parameters of infrastructure describe the interconnection of various road segments: such as the number of lanes along a road segment, posted speed limits, and traffic signal intersections, and the merge/branch points of different road segments. Such a road infrastructure is augmented with traffic monitoring and alert functions to enable an intelligent behavior:

- 1. Drivers may be notified of prevailing or anticipated congestion levels (via roadside displays, radio broadcasts, and SMS to phone subscribers);
- 2. Road crew may reduce congestion by opening and/or closing selected road segments and lanes (with a quick setup of dividers and road-blocks)⁵.

Infusing a capability for congestion notification and (limited) relief is based on computational models of the traffic-flow system, as executed by the local transportation hubs of crews that collectively manage the road infrastructure.

Given the above delineation of IPW functions, supervisory control functions can then be assigned to other units in the traffic-flow system higher in the management hierarchy: such as regional transportation centers. The latter, which constitutes the ICW, enforces policy decisions on traffic flows such as road closures and traffic prioritization. The ICW takes cognizance of the effectiveness of current infrastructure in adapting to various congestion levels, and takes recovery actions therein (e.g., authorizing the conversion of a two-way lane to a one-way lane). Such computational intelligence functions of ICW supply the configuration inputs to the IPW functions that invoke the traffic-flow system.

The traffic data collected is prone to errors for two reasons: First, the processing algorithms in sensor devices may often have only limited capabilities, and also exhibit diversity due to vendor-specific implementations. The traffic reports generated therein may be fuzzy, providing an imprecise representation of the ground truth: namely, the congestion state. Second, some of the devices may be maliciously faulty mis-reporting the traffic flow. In such a setting, *replication* of devices and *voting* on the traffic data collected by them enhances the trust-worthiness of congestion reports generated.

⁵ Closing a road or lane may sometime reduce congestion if the traffic merge from the offending road/lane onto a main road creates local vortex effects at the intersection.

In terms of our CPS-based design approach, the voting/fusion component is a part of the observer module M(O) which maps the traffic reports from various sources onto composite descriptors of congestion events. These event notifications are annotated with quantifiers that depict the quality of congestion reports q as a percentile scale: i.e., $q \in (0, 1)$. Figure 5-(a) illustrates how the event-report accuracy q impacts the decision-making process of controller module C.

6.2 Improving the accuracy of traffic reports

We employ k-out-of-N consensus voting [14] to decide on an accurate congestion report, where N is the number of replicas reporting traffic data and k is the level of consensus needed among replicas. A higher k yields a better accuracy of the congestion report, with the parameter N set to meet the condition: $1 < k \le N$. This is a case of reaching approximate agreement in sensor data fusion applications [15].

Consider a case of traffic monitoring on roads with the sensing devices mounted on police vehicles. One device may report a 80% traffic congestion on the road, whereas, another device may report a 75% congestion. The difference may arise in their traffic sampling rates and observation intervals. Besides, a malicious intruder device that poses as a police vehicle may report a traffic congestion when there is none. In the presence of such error-prone traffic reports, a central monitoring station should be able to take adequate measures to relieve the congestion — such as controlling the traffic inflow into the congested area by diverting the traffic in the upstream feeder roads. Here, a control measure taken based on incorrect reports can lead to traffic chaos — such as admitting more traffic on the feeder roads when a mis-reported congestion is acted upon by the monitoring station. Figure 5-(b) illustrates the role of replica voting in improving the accuracy of traffic reports.

With replica voting as the building-block, a data fusion mechanism based on semantic composition of the traffic reports from different regions may be employed to further improve the quality of inference about congestion. The data fusion may be based on *tree-structured overlays* set up over a vehicular network [16]. In a tree overlay, the root node is attached to a data dissemination station and the leaf nodes are attached to the data collection devices in different geographic regions (similar to [17]). The fusion architecture allows incorporating two complementary functionalities: i) sanitization of data collection by voting among replicated devices at the leaf nodes, and ii) secure propagation of the sanitized data upstream towards the root node for control actions. An intermediate node, often attached to a stable station (e.g., a police control vehicle, an airborne platform), may also carry out aggregation functions on the data arriving from its downstream tree segments and then forwarding the aggregated data upstream. Where necessary, the intermediate nodes may also be equipped with functions to initiate (limited) control actions in the local regions.

Event quality q is a parametric input to the computational model executed by the controller C. Thereupon, C assimilates the parameter q as part of its



Fig. 5. M & C in vehicular traffic management

decision-making on the traffic management $\operatorname{actions}^6$. Typically, the degree of sensor replication N and the consensus level k for voting on traffic data are controllable parameters, with $1 < k \leq N$. While improving the accuracy of traffic reports, a higher k lowers the time to generate a report due to the increased parallelism among sensor units but increases the network bandwidth consumption B to exchange synchronization messages.

The choice of [N, k, B] is aided by a calibration of the sensors vis-a-vis their event reporting quality and a computational model of the voting sub-system therein. The calibration data is maintained by the ICW for dynamically loading into the IPW. The parameter patching enables IPW to reconfigure its operations under various environment conditions.

7 Conclusions

As embedded systems become complex, there is a need to explicitly incorporate diverse physical computing systems (both hardware and software) in a coherent abstraction. Removing the explicit hardware-centric boundaries as part of the currently prevalent definitions of an embedded system, our paper introduced a concrete notion of *intelligent physical world* (IPW), and an intelligent computational world (ICW) therein, as the modules of an embedded system.

⁶ In a multi-agent based realization of C, q is viewed as the *belief probability* of an agent about the existence of a reported congestion. With epistemic reasoning about the belief states of agents, a traffic control action over different geographic regions can be realized by various agents with a certain confidence level. Study of how the accuracy parameter q impacts traffic flow-related decisions of C, and the underlying epistemic reasoning process, is deferred as a future work.

The paper described the software engineering issues in orchestrating a harmonious co-existence of the IPW and ICW. With the aid of a software structural model of a CPS, the paper studied a complex network application: viz., vehicular traffic congestion monitoring in a transportation network, through the prism of ICW-IPW partitioning.

The advantages of our CPS-style structure of an application are that it reduces the development cost of distributed control software via software reuse and modular programming. The CPS-style structure also enables easier system evolutions in the form of adding and/or modifying the controller functionalities in applications without weakening the software correctness goals.

References

- R. C. Eberhart and Y. Shi. Computational Intelligence. In chap. 2, Computational Intelligence: Concepts to Implementations, Morgan Kaufman Publ, 2007.
- S. Kabadayi, A. Pridgen, and C. Julien. Virtual Sensors: Abstracting Data from Physical Sensors. *Tech. Rep. 2006-01*, Univ. of Texas Austin, 2006.
- F. S. Hillier and G. J. Lieberman. "Non-linear Programming" and "Metaheuristics". Chap. 12, 13, *Introduction to Operations Research*, McGraw-Hill publ. (8th ed.), pp.547-616, 2005.
- J. Love, J. Jariyasunant, E. Pereira, M. Zeenaro, K. Hedrick, C. Kirsch, and R. Sengupta. CSL: A Language to Specify and Re-specify Mobile Sensor Network Behaviors. In proc. *RTAS'09*, 2009.
- Y. Diao, J. L. Hellerstein, G. Kaiser, S. Parekh, and D. Phung. Self-Managing Systems: A Control Theory Foundation. In *IBM Research Report*, RC23374 (W0410-080), Oct.2004.
- N. G. Leveson. Software Challenges in Achieving Space Safety. In Journal of the British Inter-Planetary Society, 2009.
- B. Li, K. Nahrstedt. A Control-based Middleware Framework for Quality of Service Adaptations. In *IEEE JSAC*, 17(9), Sept.1999.
- C. Lu, Y. Lu, T. F. Abdelzaher, J. A. Stankovic, S. H. Son. Feedback Control Architecture and Design Methodology for Service Delay Guarantees in Web Servers. In *IEEE TPDS*, 17(7), Sept. 2006.
- I. Schaefer and A. P. Heffter. Slicing for Model Reduction in Adaptive Embedded Systems Development. In Workshop on Software Engineering for Adaptive and Self-managing Systems (SEAMS), 2008.
- J. Yi, H. Woo, J. C. Browne, A. K. Mok, F. Xie, E. Atkins, and C. G. Lee. Incorporating Resource Safety Verification to Executable Model-based Development for Embedded Systems. In *IEEE Real-time and Embedded Technology and Applications Symp.*, 2008.
- T. Mikaelian, B. C. Williams, and M. Sachenbacher. Probabilistic Monitoring from Mixed Software and Hardware Specifications. In Proc. ICAP'05 Workshop on Verification and Validation of Model-based Planning and Scheduling Systems, 2005.
- P. E. Lanigan, P. Narasimhan, T. E. Fuhrman. Experiences with a CANoe-based Fault Injection Framework for AUTOSTAR. In IEEE/IFIP Conf. on *Dependable* Systems and Networks (DSN'10), 2010.

- A. Rowe, G. Bhatia, and R. Rajkumar. A Model-Based Design Approach for Wireless Sensor-Actuator Networks. In proc. workshop on *Analytic Virtual Integration* of Cyber-Physical Systems (AVICPS'10), Nov. 2010.
- M. V. Erp, L. Vuurpijl, and L. Schomaker. An overview and comparison of voting methods for pattern recognition. In proc. IEEE Intl. Workshop on *Frontiers in Handwriting Recognition* (WFHR02), 2002.
- R. R. Brooks and S. Iyengar. Chap. on Sensor Fusion and Approximate agreement. In *Multisensor Data Fusion*, Prentice-Hall Publ., 1998.
- K. Ravindran. Replica Voting based Architectures for Reliable Data Dissemination in Vehicular Networks. In proc. Intl. Conf. on Telecommunications for Intelligent Transport Systems (ITST-2011), IEEE, St. Petersburg (Russia), Aug. 2011.
- R. Stadler, F. Wuhib, M. Dam, and A. Clemm. Decentralized Computation of Threshold-crossing Alerts. In proc. conf. on *Distributed Systems: Operations and Management*, IEEE/IFIP, Barcelona (Spain), Oct. 2005.
- W. Hu, A. Misra, and R. Shorey. CAPS: Energy-Efficient Processing of Continuous Aggregate Queries in Sensor Networks. In proc. 4th Intl. conf. on *Pervasive Computing and Communications*, IEEE-PerCom'06, pp.190-199, June 2006.
- D. A. Amditis and et al. Multiple Sensor Collision Avoidance System for Automotive applications using an IMM approach for obstacle tracking. In proc. *Fusion'02*, Intl. Society of Information Fusion, 2002.
- S. Forrest, A. Somayaji, and D.H. Ackley. Building Diverse Computer Systems. In proc. 6th Workshop HotOS-VI, IEEE, 1997.