Proceedings of the 2018 Winter Simulation Conference
M. Rabe, A. A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, eds.

DATA-CENTRIC CYBER-PHYSICAL SYSTEMS DESIGN WITH SMARTDATA

Antônio A. Fröhlich
Davi Resner

Federal University of Santa Catarina
Software/Hardware Integration Lab
88040-900 – Florianópolis, SC, BRAZIL

ABSTRACT

Timeliness is a fundamental property of Cyber-Physical Systems that has been intensively investigated within the scope of real-time and critical systems. The advent of the Internet of Things, however, brings an intense communication flow between devices and the Internet. In this scenario, modeling time requirements in terms of data, rather than the tasks that manipulate them, may be advantageous as a data-centric design can promptly encompass other first-order requirements, such as geolocation, security, and trustworthiness. In this paper, we propose a strategy to design complex CPSs by modeling their data using the SmartData construct, which, besides encompassing means to handle the aforementioned requirements, also defines the concept of data expiry to guide scheduling decisions. With SmartData, local tasks are scheduled to produce the freshest data and to manipulate them before expiration. Likewise, network packets are scheduled prioritizing data whose expiry is close. We validate the strategy through simulations using Castalia.

1 INTRODUCTION

Timeliness is a fundamental property of *Cyber-Physical Systems* (CPS) that has been intensively investigated within the scope of reactive real-time and critical systems, mostly around the concepts of tasks and deadlines. Tasks emerge naturally from the functional aspects of the system and deadlines capture the associated time constraints. Operating systems, development tools, and methodologies proposed around these ideas are now mature and sustain most of the embedded systems behind CPSs. Nevertheless, a growing demand for advanced functionality by the users of such systems and their integration into the *Internet of Things* (IoT) are starting to challenge the establishment.

Advanced functionality often requires powerful microcontrollers or even ordinary general-purpose processors running general-purpose operating systems. The speculative, out-of-order execution of instructions in such superscalar processors makes it difficult for developers to determine the Worst-Case Execution Times (WCET) estimates required by most real-time algorithms in order achieve schedulability confidence (Nowotsch et al. 2014). At the same time, the advent of the IoT brings along an intense communication flow between devices in a CPS as well as between them and the Internet, calling for network-wide resource management, particularly time, energy, and bandwidth. The *Time-Triggered Architecture* (TTA) (Kopetz and Bauer 2003) addresses many of these issues, but it does that through a strictly synchronous design in which every computation and communication is planned a priori and statically scheduled – assumptions that are incompatible with the envisioned scenario.

We believe that modeling time requirements in terms of the data, rather than the tasks that manipulate them, may be advantageous as a data-centric design can promptly encompass other first-order requirements, such as geolocation and security. In this paper, we explain how timeliness is handled in the context of *SmartData* (Fröhlich 2018), a high-level Application Programming Interface (API) for CPSs that aims at leveraging the myriad of features available on the embedded platforms that support them while delivering a common interface for sensing, actuating, and controlling based on the associated data. SmartData

978-1-5386-6572-5/18/$31.00 ©2018 IEEE

encapsulates data with metadata that makes it self-contained in terms of semantics, spatial location, timing, and trustfulness. It is meant to be the only application-visible construct in the platform and, therefore, implicitly mediates all system-level services, including communication, synchronization, and the interaction with transducers and actuators. SmartData objects are tagged with an *Expiry* representing the last moment in time their values can be used without misleading associated computations to produce wrong results.

A CPS built on SmartData is comprised of components that interact with each other through messages that are implicitly exchanged over a communication channel to ensure that the application-visible data are consistent. Components and channels can be either physical, such as Engine Control Units (ECUs) on a Controller Area Network (CAN) bus, or logical, such as tasks on a multicore. The discussions in this paper use an implementation of SmartData for the Embedded Parallel Operating System (EPOS) and its Trustful Space-Time Protocol (TSTP) (Resner et al. 2017). We detail the SmartData concept in Section 2, addressing each of its roles on a CPS along with its major facets: semantic data, network and processing abstraction, timing, georeferencing, and security. Section 3 discusses the *timeliness* of systems designed and implemented on SmartData, including the local scheduling of service tasks along with user tasks, the timely routing of messages pertaining SmartData updates, and premises for design-time reasoning on the temporal behavior of the whole system. Section 4 is devoted to a simulation of a SmartData-based CPS aiming at substantiating the discussions about its timeliness. Section 5 discusses related work and Section 6 finishes the paper with our final remarks.

2 SMARTDATA

SmartData was conceived to be the primary (if not the only) abstraction used by application programmers to interact with the physical world on a network of sensors and actuators supporting a CPS. A SmartData instance is a piece of data enriched with enough metadata to make it self-contained regarding semantics, spatial location, timing, and trustfulness. The semantic aspects of SmartData are described using a strategy inspired by the Transducer Electronic Data Sheets in the IEEE 1451 standard (IEEE 1451.0 2007). Each piece of data is tagged with a 32-bit type identifier designating either an *SI Physical Quantity* or plain digital data. Physical quantities are identified by the corresponding unit of the International System (SI). Derived SI units are expressed in terms of SI basic units. Digital data are simply classified at this level, with actual encoding being specific to each defined class (Software/Hardware Integration Lab 2018).

Besides a `Value` and a `Unit`, each piece of SmartData carries along the coordinates of the location where it was produced (`Origin(x,y,z)`), a local enumerator of the device that produced it (`Origin(d)`), a high-resolution timestamp identifying the moment in which it was produced (`Origin(t)`), the validity of the data expressed relatively to the time it was produced (`Expiry`), and a message authentication code that can be used to verify its authenticity (`MAC`). The local enumerator `Origin(d)` is not a device Id. It is a simple counter that is only used to disambiguate multiple devices producing the same kind of data (`Unit`) at the same place and time (`Origin(x,y,z,t)`), for instance, a 3-axis accelerometer. Figure 1 depicts SmartData from the perspective of communication protocols.

The SmartData application interface is depicted in Figure 2. Although it only exports few and simple methods, this interface encapsulates a powerful mechanism to abstract any sort of sensor and actuator on a system. Instances of SmartData can be created using one of three constructors: the first one is used to abstract local transducers, the second is used to create local proxies of remote transducers, and the third is used to model controllers. In all cases, the `Transducer` class parameter binds a SmartData object with the corresponding sensor, actuator or controller. Every transducer is supposed to declare a constant named `UNIT`. SmartData uses this constant to personify the corresponding SI quantity. For instance, a SmartData object instantiated with a transducer specifying *K* (Kelvin) as `UNIT` represents the SI quantity Temperature.

Origin (x,y,z,d,t)	Unit	Value	Expiry	MAC

Figure 1: SmartData encapsulated in a network packet.

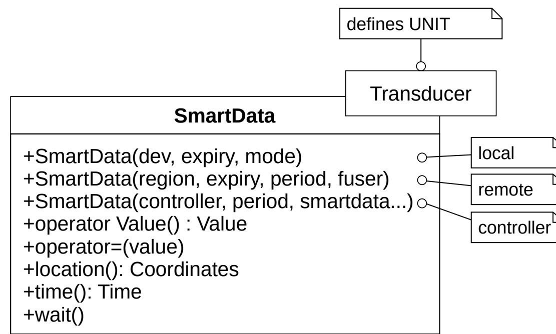

Figure 2: SmartData interface.

This SmartData can abstract either a temperature sensor, an air conditioner (i.e., a temperature actuator), or a temperature controller.

2.1 Local Transducers as SmartData

Local transducers are abstracted using SmartData objects instantiated with the first constructor depicted in Figure 2. The constructor's first parameter, `dev`, is used to differentiate multiple instances of the same transducer available on a given node. The network carries it as the `Origin(d)` parameter, presented in Figure 1. The second parameter, `expiry`, represents the latest point in time when functions can use the data without a correctness compromise. This concept is further discussed in Section 3. The last constructor parameter, `mode`, designates one of the three possible SmartData modes of operation: private, advertised, and commanded. *Private* SmartData are not advertised to the network and, therefore, cannot be monitored or controlled remotely. *Advertised* SmartData are advertised on the network and, therefore, can be remotely monitored. *Commanded* SmartData are advertised on the network and can be remotely controlled. A commanded SmartData defines a data-centric mechanism to interact with cyber-physical environments as it replaces explicit actuation commands by an implicit setpoint represented by the SmartData value. By writing a value to a SmartData, users enable actuation until the setpoint is reached.

Figure 3 shows examples of SmartData representing local transducers. The one at line 1 defines an object whose value represents the electric current sensed by an Ammeter (SmartData and Transducer are bound by the Ampere SI unit). The object is not advertised to the network. It is meant to be used by a stand-alone component of the CPS. Reading the value of this object implies in sampling the transducer whenever the last sample gets older than 10 *ms*. At line 2, `voltage` defines a SmartData connected to a Voltmeter through the Volt SI unit, with samples lasting for 1 *ms* and that can be remotely accessed. The object at line 3 represents a remotely accessible on-off switch with a 300 *ms* expiry. The unit binding this SmartData to the Transducer is not SI; it is a digital convention named `ON_OFF`. The SmartData at lines 4 and 5 are both bound to the Kelvin SI unit, and also advertised to the network. However, the former represents a sensor (i.e., the second thermometer in the platform, since `dev = 1`), while the latter represents an actuator (i.e., air conditioner). Actuators interpret the expiry parameter as an indicator of the time allowed for the actuation to take effect (2 *min* in the example).

2.2 Remote Transducers as SmartData

Remote transducers are also abstracted using instances of SmartData. Objects created with the second constructor depicted in Figure 2 are local proxies to remote transducers. All the system-level procedures needed to ensure a consistent semantic for such objects are transparently carried out by the API's implementation, including communication, synchronization, and scheduling. The constructor's first parameter is used to specify a space-time region of interest as:

$$\textit{Space-Time Region}(x, y, z, r, d, t_0, t_f)$$

```
1  SmartData<Ammeter> current(0, 10ms, PRIVATE);
2  SmartData<Voltmeter> voltage(0, 1ms, ADVERTISED);
3  SmartData<Switch> key(0, 300ms, ADVERTISED);
4  SmartData<Thermometer> current_temperature(1, 1s, ADVERTISED);
5  SmartData<AC> desired_temperature(0, 2min, COMMANDED);
```

Figure 3: Examples of SmartData representing local transducers.

where x, y, z designate the center of the interest region, r designates its radius, and $[t_0, t_f]$ designates the interest's time interval. The interest region is used in combination with the SI physical quantity designated by `UNIT`. The instantiation of a remote SmartData implies in the announcement, over the network, of interest for a given SI quantity in a given region of space for a given interval of time. SmartData objects matching the criteria will reply to the interest according to parameters `period` and `fuser`. A zero period means that all updates on the remote nodes will be reported, determining an *event-based* mode of operation. Other values require matching objects to periodically send a response every `period` units of time, from t_0 until t_f, defining a *time-triggered* operation for the network. If d is given, then this local device enumerator is used as a filter to express interest for specific transducers.

If `fuser` is not provided, then objects matching the criteria will respond to the interest if they have not yet listened to a response from another node. This mechanism does not ensure a single response: hidden nodes might concurrently respond to the same interest. The value of the SmartData will be that of the last response received, and expiry will be adjusted accordingly. Otherwise, when a `fuser` function is given, all transducers matching the criteria across the network will respond to the interest. Every new response will be handed over to the function for data fusion, and the function's return value will define the SmartData value.

Similarly to local transducers, remote ones abstracted as SmartData also have an *Expiry*. This Expiry is kept as client-side information, so several SmartData objects acting as proxies to the same transducer can have different expiration times. Nodes with SmartData exported as `ADVERTISED` will manage proper response messages to match all interests. The shortest of the expiries is used in the network packet (see Figure 1).

Some examples of SmartData representing remote transducers are given in Figure 4. The first line declares the space-time region of interest. It represents a sphere with a radius of 15 *m* centered at coordinates (150 *cm*, 300 *cm*, 125 *cm*), and a time interval from the current time to one hour in the future. The first SmartData, in line 2, will cause remote transducers at the given space-time region to sense and send voltage data every 10 *ms*, with a 10 *ms* expiry. The second example, `key`, omits the period, assuming an event-driven behavior. It prompts remote transducers that personify switches to send the state of the switch whenever it changes, with a 300 *ms* expiry. Since there may be more than one sensor of the requested type in the desired region, each of the SmartData instances might receive data from multiple sources. In the third example, a `fuser` function is passed to the SmartData constructor to aggregate these possible multiple readings into a single, meaningful value. The last example represents a temperature actuator, which can be manipulated either by calling the assignment operator or with a controller SmartData, presented next.

2.3 Controllers as SmartData

The third constructor in Figure 2 is used to instantiate a SmartData controller. It takes as parameters an arbitrary number of SmartData sensors and actuators `(smartdata...)`, a `controller` function and, optionally, the `period` within which the function must be executed. The controller function's signature must match the types of the SmartData parameters in the constructor. SmartData is currently implemented in C++, and this constructor is implemented using Variadic Templates and a function pointer with the same signature. This function implements the actual control algorithm that transforms the data read from the SmartData sensors into commands that are issued to the SmartData actuators. Writing to the controller

```
1 Region region(Coordinates(150cm, 300cm, 125cm), 15m, now, now + 1h);
2 SmartData<Generic<Volt>> voltage(region, 10ms, 10ms);
3 SmartData<Generic<ON_OFF>> key(region, 300ms);
4 SmartData<Generic<Kelvin>> temperature(region, 1s, 500ms, &fuser);
5 SmartData<Generic<Kelvin>> ac(region, 2min, 10min);
```

Figure 4: Examples of SmartData representing remote transducers.

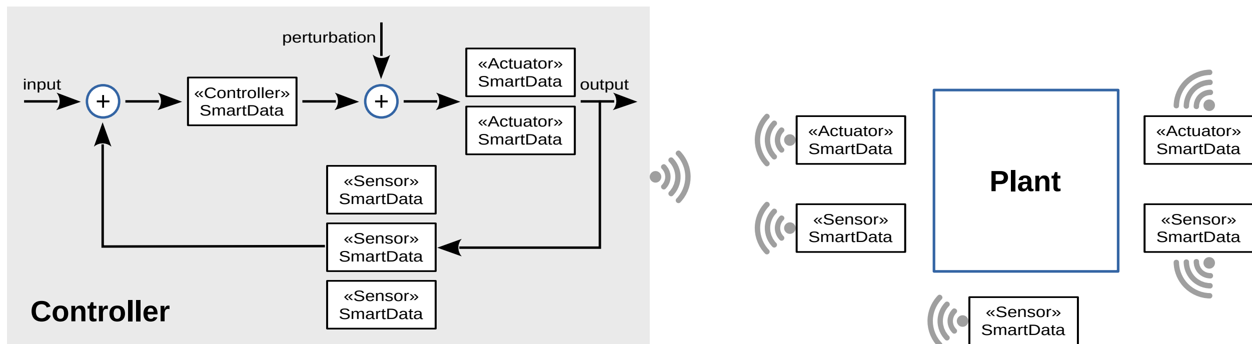

Figure 5: CPS control based on SmartData, with local proxies of sensors and actuators integrated into a controller and remote sensors and actuators interacting with a plant.

SmartData changes the setpoint for the control function. The general idea of a system controlled using SmartData is depicted in Figure 5.

Controllers that do not define `period` are *event-driven*. The control function in these controllers is invoked whenever an aggregated SmartData gets updated (either through a local sampling of the associated transducer or the reception of a response message over the network to an announced interest). Controllers that do define a period are *time-triggered* and are periodically invoked independently of the associated SmartData. An intermediate case can be achieved through the use of the `wait` method of the aggregated SmartData by the control function. This method causes the caller to block until the SmartData value gets updated. In this case, the control function of an event-driven controller becomes periodic with a period defined by the SmartData with the longest period. The control function can apply `wait` selectively to some SmartData only, using the remainder based on their expiries.

The exact mechanisms used to propagate events from Transducers to SmartData and among them have been previously described in details by Ludwich and Fröhlich (2015). They combine lightweight Semaphores with the Observer design pattern to encapsulate low-level interrupts as notifications of observed objects to higher level observers up to the application.

An example of SmartData representing a controller is given in Figure 6. The first three lines declare, respectively, the space-time region of interest, a remote temperature sensor SmartData, and a remote temperature actuator SmartData. This sensor-actuator pair is then tied to a controller SmartData in line 4. It will cause the `controller` function to be called every 5 minutes with the two SmartData instances as parameters. The controller function implements the specific logic for changing the air conditioning intensity based on the data read by the temperature sensors to reach the setpoint represented as the value of the `control` SmartData.

```
1 Region region(Coordinates(150cm, 300cm, 125cm), 15m, now, now + 1h);
2 SmartData<Generic<Kelvin>> temperature(region, 1s, 500ms, &fuser);
3 SmartData<Generic<Kelvin>> ac(region, 2min, 10min);
4 SmartData<Generic<Kelvin>> control(&controller, 5min, temperature, ac);
```

Figure 6: Examples of SmartData representing controllers.

2.4 SmartData Value, Location, Time, and Security

Independently of how a SmartData is created, five common methods can always be invoked on it: a native type conversion operator, the assignment operator, `location()`, `time()`, and `wait()`. The conversion operator, summarized as `operator Value()` in Figure 2, enables a SmartData to be used just like the native type used to hold its value. The assignment operator (`operator=()`) issues commands to the Transducers and Controllers associated with the SmartData. The method `location()` returns the coordinates of the location where the data were produced, while the method `time()` returns the time in which the data were produced. The method `wait()` causes the caller to block until the SmartData value gets updated.

The security mechanisms used to protect a SmartData are based on the Poly1305-AES cryptographic MAC (Bernstein 2005), on keys generated with the Elliptic Curve Diffie–Hellman (ECDH) algorithm (Raso et al. 2015), and on the precise time synchronization delivered by the Speculative Precision Time Protocol (SPTP) (Resner and Fröhlich 2016). Messages containing SmartData values are time-stamped, encrypted, and authenticated to ensure the desirable properties. The fields `MAC` and `t` in Figure 1 are used for this purpose. These security mechanisms have been previously described in details by Resner et al. (2017).

3 SMARTDATA TIMELINESS

Timeliness in CPS is usually achieved by modeling the embedded systems that support them with a combination of decomposition and real-time scheduling (Hu et al. 2012). On the one hand, system functionality is decomposed into simple tasks that run isolatedly on their own microcontrollers, and that interact with each other over a statically modeled interconnect (e.g., ECUs on a CAN bus). On the other hand, classic real-time scheduling algorithms are applied to run multiple tasks on more powerful processors interconnected using off-the-shelf networks (e.g., CIM over Ethernet). In both cases, *deadlines* are the main concept used to express the timing requirements of each CPS's part or component. Unquestionably, deadlines define a solid foundation atop of which CPS can be consistently built. However, the growing complexity of such systems is now challenging this concept (Choi et al. 2017; Phan et al. 2017).

As introduced in the previous sections, SmartData objects are tagged with an *Expiry* representing the last moment in time their values can be used without the risk of misleading associated computations to produce wrong results. This concept is the primary element to represent timeliness in a CPS modeled around SmartData. It is implicitly used to derive parameters, including deadlines, for the local scheduling of the tasks supporting SmartData, to route packets containing SmartData globally, and to estimate the overall system load at design time. SmartData Expiries are means to express timeliness through a more compartmented reasoning about how long a piece of data lasts. Besides improving domain decomposition on the same premises as object-orientation improves on pure functional decomposition (Tan et al. 2006), expiries can be directly derived from the sampling frequencies in the physical models used during the design of a CPS, with actual task scheduling being automatically derived thereof.

Before explaining how a CPS based on SmartData is scheduled, it is important to recall that each client component using a SmartData can define its own particular view of Expiry. For one component, the data produced by a sensor can be valid for an amount of time that is different from that defined by another component. Therefore, Expiry is a per-client property of SmartData that is expressed through the `expiry` parameter in the SmartData constructors depicted in Figure 2. Expiry is refreshed on every SmartData update; hence it is a relative value. As also mentioned earlier, a CPS modeled around SmartData can operate in one of two modes: event-driven or time-triggered. An event-driven SmartData has its value updated asynchronously – usually by invoking getter-link methods on the associate Transducer – whenever it is accessed, while the value of a time-triggered SmartData is periodically updated. Therefore, a time-triggered SmartData requires the specification of a Period in addition to the Expiry (through the `period` parameter in Figure 2). Specifying an Expiry smaller than the Period is invalid since the attempt to access an expired SmartData immediately triggers an update, which would disrupt its periodic operation. If distinct SmartData

objects using the same Transducer are created with different periods, then the greatest common divisor is used for sampling, but the `wait()` method applied to each object respects their originally specified periods. Dissonant periods, such as prime numbers, can impose a high penalty on energy consumption and network bandwidth utilization and should be avoided (Nasri and Fohler 2015).

3.1 SmartData Expiries and Local Scheduling

The API defined by the SmartData construct is sustained by a complex run-time support system, which in many cases implicitly creates service tasks. These tasks are scheduled along with user-defined ones using a specific real-time scheduling algorithm. The instantiations of SmartData that can result in the creation of service tasks are discussed below:

1. *Event-driven SmartData encapsulating local Transducers that operate via polling* are updated on-demand whenever an attempt to access an expired value occurs. For most cases, the update operation is limited to reading a few registers and the incurred time is computed as a constant latency in the access time of the SmartData. For a complex sensor, however, demanding considerable processing for updates, an implicit periodic task with Period equal to the SmartData Expiry is created, so when a component accesses the SmartData, its value is already updated.
2. *Event-driven SmartData encapsulating local Transducers that operate via interrupts* get their values updated by interrupts raised whenever a meaningful event is observed at the associated Transducers. If the handling of such interrupts for a given Transducer is not trivial and requires considerable processing time, then an aperiodic task is created for this purpose and conditioned to a light semaphore signalized by a simple, constant-time, generic interrupt handler (this scheme was originally modeled as the *Concurrent Observer* design pattern, Ludwich and Fröhlich 2015). The aperiodic service tasks are prioritized according to the priority defined for the associated Transducers at design-time.
3. *Time-triggered SmartData encapsulating local Transducers that operate via polling* achieve periodic operation through the implicit creation of periodic service tasks by the Transducers. Such tasks are created with a Period that is equal to the Expiry of the associated SmartData. Note that local SmartData never define Period (it is not a parameter of the constructor that yields local objects). It is the SmartData's Expiry that locally defines the sampling period of a Transducer, which, in turn, established a time-triggered mode of operation through the creation of a periodic service task.
4. *Time-triggered SmartData encapsulating local Transducers that operate via interrupts* inherit their periodic operation by the interrupts that are periodically triggered by the associated Transducers to update the SmartData value. The time elapsed at each update is modeled as system's overhead that causes jitter to other tasks. If the update function execution time is significant, or if the CPS is jitter-sensitive, then the Transducer must model the update function as a task that is just released by the interrupt. This task would be aperiodic and would be synchronized with the interrupt via a light semaphore just like in 2. The same considerations about harmonic periods for multiple SmartData are valid here. The mapping of interrupts into task activations (viz., job releases) is a constant-time operation that is computed as system's overhead, while the service task execution time is considered along with other tasks.
5. *Advertised SmartData that respond to periodic Interests* create service tasks for each matching interest. When a node (i.e., a CPS component) receives an Interest message that matches its capacity – for instance, a request for sensing data of a given SI Unit that can be measured by the node – it creates a periodic service task with the same Period as expressed in the Interest message. A response message to that interest is sent at each Period to update the SmartData's proxy value.
6. *Time-triggered SmartData encapsulating periodic controllers* implicitly create a periodic service task whose Period is that of the controller to invoke the associated control function.

These tasks are locally scheduled using a combination of the Modified Least Laxity First (MLLF) algorithm (Oh and Yang 1998) and the Modified Maximum Urgency First (MMUF) (Salmani et al. 2007), such that objects closer to becoming invalid are updated first. Proving the schedulability of a system using SmartData is out of scope for this work, but provided WCET of the tasks are available, the proof would be directly accomplished through the schedulability analysis of MMUF. Application tasks in a system using SmartData can be mostly aperiodic, with a periodic behavior being implicitly accomplished by the `wait()` blocking method that is used to wait for a SmartData update. Indeed, many of the existing applications consist solely of SmartData instantiations.

3.2 SmartData Expiries and Network Routing

As seen in Section 3.1, SmartData Expiries can instruct local scheduling decisions to make sure that data are always fresh. When working with remote SmartData, however, network delays can prevent correctly-scheduled SmartData from honoring their expiries. In this section, we show that the space-time semantics of SmartData can also guide network routing decisions to better meet expiries. As an example, we investigate a SmartData implementation on a Wireless Sensor Network (WSN) using the Trustful Space-Time Protocol (Resner et al. 2017) for communication.

TSTP is a cross-layer protocol that has access to the SmartData fields (Figure 1) and makes nodes aware of their spatial coordinates. For data messages, TSTP employs an all-to-sink, greedy geographic routing strategy. Messages carry only the final destination address (as coordinates). At the link level, messages are broadcasted, and forwarders are distributively selected based on their position and possibly other metrics of interest.

When a TSTP node receives a message and detects that it is closer to the destination than the current sender, it becomes a relay candidate and immediately sleeps for a time offset δ:

$$\delta = \alpha \times \frac{R-(D_m-D)}{R} \times S \tag{1}$$

where D is the current node's distance to the message's destination, D_m represents the message sender's distance to the destination, R is a network-wide parameter corresponding to the radio range of the nodes, and S is an upper bound derived from the idle listening duty cycle of nodes (Resner et al. 2017). α is a distortion coefficient, explained next. This equation makes the relay candidate closest to the destination wake up earlier, and when it does, it wins the contention round and proceeds to forward the message.

The distortion coefficient $\alpha \in [0,1]$ is used to prioritize messages by reducing the offset δ according to other metrics besides geographic distance. TSTP uses SmartData expiry times to prioritize messages that are close to expiring. The coefficient is determined as:

$$\alpha = \frac{e-t}{e-t_0} \tag{2}$$

where e is the time at which the message expires, t is the current time, and t_0 is the message's time at the origin. The offset δ is, thus, reduced in proportion to the time left until the message's expiration, increasing the message's likelihood of being forwarded earlier. A similar logic can be used with any other communication protocol that lets upper layers inform message priorities.

3.3 SmartData Expiries and CPS Design

As discussed in Section 3.1, the instantiation of some kinds of SmartData implies the creation of service tasks. The task set implicitly created by the SmartData run-time support system for a given CPS, in some cases, can be modeled and evaluated at design-time. If the system is comprised only of time-triggered SmartData, or if it combines time-triggered SmartData with event-driven ones bound to timed Transducers, then the characterization of the task set in terms of periods and deadlines can be directly derived from the

parameters supplied to the constructors depicted in Figure 2. Table 1 summarizes the possible combinations of SmartData and Transducers from the perspective of a local node hosting advertised SmartData. If the SmartData is only used locally (column Local in Table 1), then it inherits the temporal behavior of the associated Transducer: either periodic with a period defined by the SmartData Expiry (a PCM audio encoder is an example of a timed polling transducer), or aperiodic on a fully asynchronous, event-driven mode of operation. If an advertised SmartData bound to an untimed, polling Transducer is used remotely, then a periodic task is created to sample the Transducer at each Expiry units of time. This task ensures a periodic operation for the remote SmartData, even if a period is not defined for the local one. Time-triggered remote SmartData all define a periodic behavior through the creation of a service task at the node hosting the Transducer. That task gets its period from the SmartData Period and its deadline from the SmartData Expiry. However, at design-time, it is not possible to reason about SmartData expiration (and deadline misses of the associated service tasks) for the case involving a time-triggered SmartData bound to an untimed, interrupt-based Transducer. A Transducer of this kind only refreshes the SmartData value when a meaningful event is observed. Conjectures about the maximum period between two such events and SmartData expiration can only be done for particular cases.

Table 1: Summary of service tasks implicitly created by a local SmartData as a function of Transducer types and remote SmartData. A *Periodic(E)* task has period and deadline equal to the SmartData's Expiry. A *Periodic(P,E)* task has period and deadline defined by the time-triggered SmartData's Period and Expiry. *Send(X)* designates the additional time needed to send a response with a SmartData update every X units of time. Combinations in gray have deterministic local behavior and predictable global behavior. The one in light gray can suffer data expiration if events are sparser than Expiry. Those in white are not suitable for real-time scenarios.

			SmartData		
			Local	Remote	
			Event-Driven	Event-Driven	Time-Triggered
Transducer	Timed	Polling	Periodic(E)	Periodic(E)+Send(E)	Periodic(P,E)+Send(P)
		Interrupt	Periodic(E)	Periodic(E)+Send(E)	Periodic(P,E)+Send(P)
	Untimed	Polling	Aperiodic	Periodic(E)+Send(E)	Periodic(P,E)+Send(P)
		Interrupt	Aperiodic	Aperiodic	Periodic(P,E)+Send(P)

If WCET estimates can be obtained for sensing, processing, controlling, and actuating, then a CPS can be subjected to schedulability analysis using the periods and deadlines shown in Table 1 and a real-time scheduling algorithm such as the MMUF currently used in the implementation of the SmartData API for EPOS (Software/Hardware Integration Lab 2018).

4 EVALUATION

In order to substantiate the discussions of Section 3, we modeled and implemented a CPS around a Wireless Sensor Network (WSN) with a controller at a central sink. The system was first implemented for EPOS on a small scale and then validated through simulations on the OMNeT++ simulator with the Castalia framework. SmartData and their Expiries were modeled considering the principles discussed earlier in the paper, thus yielding a schedulable system. However, to confirm the expected impact of prioritizing messages based on the Expiries of the SmartData they carry, we subsequently varied the SmartData period, eventually saturating the network. The modeled network covers a 500 *m* by 500 *m* field with 116 nodes on a regular deployment. Each node has at most six neighbors with equal distances. 115 of these nodes instantiate SmartData (and the associated service tasks) to respond to an Interest message propagated by a central sink node through the instantiation of a time-triggered, remote SmartData. Table 2 summarizes the simulation parameters.

Number of nodes	116	Radio range	$\approx 60\ m$
Field size	$500\ m$ x $500\ m$	Avg. # of hops	3.957
Simulation time	$2\ h$	Avg. # of neighbors	5.200
Deployment	Regular	Data period	$20\ s$ to $140\ s$
Channel model	Free space	Data expiry	same as period
Transmission power	$0\ dBm$	MAC duty cycle	1%

Table 2: Simulation parameters.

Figure 7 presents the simulation results. Each data point represents the average of 10 replications. Figure 7a shows the impact on the timely delivery of messages, comparing the original TSTP distance-based routing metric ((1) with $\alpha = 1$) against the expiry-enriched metric ((1) with α as defined in (2)). A TSTP network will drop messages that do not arrive at their final destination before their Expiry, and, therefore, the delivery ratio in the figure represents the proportion of messages that were delivered in time with each strategy. In every case, the delivery ratio was improved by the expiry routing metric, as messages that are closer to expiring get higher priority and are delivered in time more often. However, for TSTP, increasing the priority of messages comes with an energy cost, as shown in Figure 7b. The best relative energy to delivery ratio gain is at the 60 s period, with a 10.4 % energy consumption increase leading to a 30.42 % increase in delivery ratio. When the network is most saturated, the energy consumption grows by 21.31 % for a 6.37 % increase in delivery ratio. For the higher data periods, energy consumption is increased by 0.43 % for a 0.88 % increase in delivery ratio. In summary, using SmartData expiries to guide routing decisions improves TSTP's timely delivery of messages, even if at an energy cost.

The performance of TSTP has been extensively evaluated in previous work (Resner et al. 2017), considering several parameters, scenarios, and techniques. The present analysis is only meant to illustrate that expiry-based expedite routing can improve timely delivery of messages in existing protocols.

5 RELATED WORK

SmartData and *Interfaces* in the Time-Triggered Architecture (TTA) (Kopetz and Bauer 2003) share many common aspects. Both constructs are used to mediate data access by Actors, decoupling locality while preserving timeliness. TTA Interfaces, however, are defined in the context of a strictly synchronous design, where every computation and communication is planned a priori in a static time-division multiple access scheme. SmartData Expiries aim at fostering time determinism through a less-strict strategy that is more suitable to IoT systems, which cannot cope with TTA assumptions. SmartData also addresses data semantics, spatial location, security, and trustworthiness in addition to timeliness. Both approaches, however, present a common limitation: it is very difficult to define the activation time of the jobs responsible for updating the data abstracted by Interfaces and SmartData when such update requires complex computation. If update jobs are released too early, data might get used when they are already worn-out. If they are released only at the specified periods, they might reach the client components too late. In both scenarios, SmartData and Interface, this problem is delegated to programmers, who must account for the update time while estimating the WCET of components.

Frincu (2017) discusses the needs of IoT-connected CPSs and identifies the need for three levels of control, which are discussed at a higher level. The *dew* level consists of low-power devices with limited memory and processing power, which run local data aggregation based on their local history. At the *fog* level, the data from these devices are further aggregated to reduce bandwidth consumption when sending it to the *cloud* level. At the latter, global-knowledge decisions are made and an offline process selects appropriate control models based on the current state of the system. Such system could be implemented with SmartData. The dew level aggregation algorithms are implemented at the Transducer level. The fog level communication and aggregation is naturally achieved by remote SmartData with a fuser function. Furthermore, in previous work, we have explored the utilization of light prediction models to greatly reduce communication overhead (Huegel et al. 2017) that are directly applicable to SmartData both at dew-fog

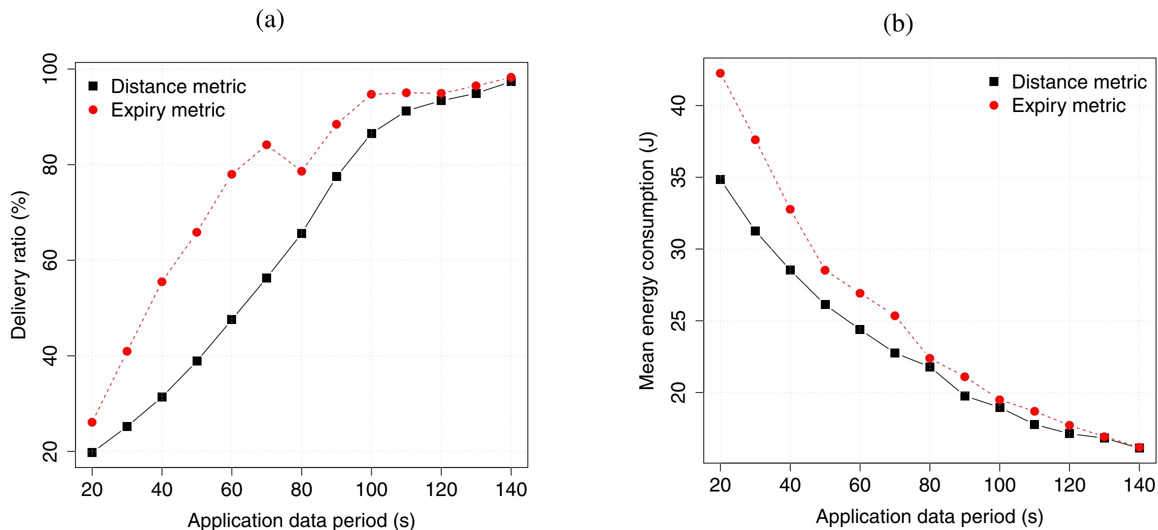

Figure 7: Simulation results; (a) Delivery ratio; (b) Energy consumed.

and fog-cloud levels. Cloud communication is out of scope for the present work, but can be employed at SmartData-Internet gateways.

6 FINAL REMARKS

In this paper, we discussed the timeliness of SmartData while considering how it can be used to guide a data-centric CPS design. We recapitulated the concept and the API, describing how it abstracts most of the run-time support system, including synchronization, coordination, and communication, while delivering a lean interface for sensing, actuating, and control based on the data produced and consumed by the system. Timing was considered from the perspective of local components, globally for the set of interconnected components, and at design-time. An implementation of SmartData for EPOS/TSTP supported the discussion. A simulation of a SmartData-based CPS also substantiated the discussions about the timeliness of systems designed around the concept, demonstrating how a high level of temporal determinism can be achieved.

REFERENCES

Bernstein, D. J. 2005. “The Poly1305-AES Message Authentication Code”. In *Fast Software Encryption*, edited by H. Gilbert and H. Handschuh, 32–49. Berlin: Springer.

Choi, S., A. Chavez, M. Torres, C. Kwon, and I. Hwang. 2017. “Trustworthy Design Architecture: Cyber-Physical System”. In *2017 International Carnahan Conference on Security Technology*, October 23rd–26th, Madrid, Spain.

Frincu, M. 2017. “Architecting a Hybrid Cross Layer Dew-Fog-Cloud Stack for Future Data-Driven Cyber-Physical Systems”. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, May 22nd–26th, Opatija, Croatia, 399–403.

Fröhlich, A. A. 2018. “SmartData: an IoT-Ready API for Sensor Networks”. *International Journal of Sensor Networks*. In press.

Hu, L., N. Xie, Z. Kuang, and K. Zhao. 2012. “Review of Cyber-Physical System Architecture”. In *2012 IEEE 15th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops*, April 11th, Shenzhen, Guangdong, China, 25–30.

Huegel, C., G. Gracioli, and A. Fröhlich. 2017. “Space-Time Derivative-Based Prediction: A Novel Trickling Mechanism for WSN”. In *Brazilian Symposium on Computing Systems Engineering*, November 7th–10th, Curitiba, Brazil, 47–54.

IEEE 1451.0 2007. *IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Common Functions, Communication Protocols, and Transducer Electronic Data Sheet (TEDS) Formats*.

Kopetz, H., and G. Bauer. 2003. "The Time-Triggered Architecture". *Proceedings of the IEEE* 91(1):112–126.

Ludwich, M. K., and A. A. Fröhlich. 2015. "Proper Handling of Interrupts in Cyber-Physical Systems". In *2015 International Symposium on Rapid System Prototyping (RSP)*, October 8th–9th, Amsterdam, The Netherlands, 83–89.

Nasri, M., and G. Fohler. 2015. "An Efficient Method for Assigning Harmonic Periods to Hard Real-Time Tasks with Period Ranges". In *2015 27th Euromicro Conference on Real-Time Systems*, July 7th–10th, Lund, Sweden, 149–159.

Nowotsch, J., M. Paulitsch, D. Buhler, H. Theiling, S. Wegener, and M. Schmidt. 2014. "Multi-core Interference-Sensitive WCET Analysis Leveraging Runtime Resource Capacity Enforcement". In *2014 26th Euromicro Conference on Real-Time Systems*, July 8th–11th, Madrid, Spain, 109–118.

Oh, S.-H., and S.-M. Yang. 1998. "A Modified Least-Laxity-First scheduling algorithm for real-time tasks". In *Fifth International Conference on Real-Time Computing Systems and Applications*, October 27th–29th, Hiroshima, Japan, 31–36.

Phan, D., J. Yang, M. Clark, R. Grosu, J. Schierman, S. Smolka, and S. Stoller. 2017. "A Component-Based Simplex Architecture for High-Assurance Cyber-Physical Systems". In *17th International Conference on Application of Concurrency to System Design (ACSD)*, June 25th–30th, Zaragoza, Spain, 49–58.

Raso, O., P. Mlynek, R. Fujdiak, L. Pospichal, and P. Kubicek. 2015. "Implementation of Elliptic Curve Diffie Hellman in Ultra-Low Power Microcontroller". In *2015 38th International Conference on Telecommunications and Signal Processing (TSP)*, July 9th–11th, Prague, Czech Republic, 662–666.

Resner, D., and A. A. Fröhlich. 2016. "Speculative Precision Time Protocol: Submicrosecond Clock Synchronization for the IoT". In *21st IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2016)*, September 6th–9th, Berlin, Germany.

Resner, D., G. M. de Araujo, and A. A. Fröhlich. 2017. "Design and Implementation of a Cross-Layer IoT Protocol". *Science of Computer Programming*. In press.

Salmani, V., S. T. Zargar, and M. Naghibzadeh. 2007. "A Modified Maximum Urgency First Scheduling Algorithm for Real-Time Tasks". *International Journal of Computer, Electrical, Automation, Control and Information Engineering* 1(9):2827–2831.

Software/Hardware Integration Lab 2018. "Embedded Parallel Operating System Project". https://epos.lisha.ufsc.br, accessed May 1st, 2018.

Tan, H., Y. Yang, and L. Bian. 2006. "Systematic transformation of functional analysis model into OO design and implementation". *IEEE Transactions on Software Engineering* 32(2):111–135.

AUTHOR BIOGRAPHIES

ANTÔNIO A. FRÖHLICH is Full Professor for CS at the Federal University of Santa Catarina (UFSC), where he leads the Software/Hardware Integration Lab (LISHA) since 2001. With a Ph.D. in Computer Engineering from TU-Berlin, he has coordinated several R&D projects on embedded systems, including the ALTATV Open, Free, Scalable Digital TV Platform, the CIA^2 research network on Smart Cities and the Internet of Things, and the Smart Campus project at UFSC. He is a senior member of ACM, IEEE, and SBC. His email address is guto@lisha.ufsc.br.

DAVI RESNER has M.Sc and B.Sc degrees in CS by the Federal University of Santa Catarina (UFSC). He has worked with the Software/Hardware Integration Lab (LISHA) since 2013, where he has been involved in R&D projects on IoT infrastructure and communication protocols. His email address is davir@lisha.ufsc.br.