2016 IEEE World Congress on Services Computing

On Monitoring Cyber-Physical-Social Systems

Muhammad Z.C. Candra, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
{m.candra,truong,dustdar}@dsg.tuwien.ac.at

***Abstract*—Recent developments of computing systems allow humans to participate not only as service consumers but also as service providers. The interweaving of human-based computing into machine-based computing systems becomes apparent in smart city settings, where human-based services together with software-based services and thing-based services (e.g., sensor-as-a-service) are orchestrated for solving complex problems, leading to the creation of the so-called Cyber-PhySical-Social Systems (CPSSs). Monitoring such CPSSs is essential for system planning, management, and governance. However, due to the diversity of the involved building blocks, it is challenging to monitor such systems. In this paper, we present metric models and the associated Quality of Data (QoD) to elastically monitor the execution metrics of a centralized coordinated CPSS. We develop a monitoring framework for capturing and analyzing runtime metrics occurring on various facets of the coordinated CPSS. Furthermore, we present the implementation of our monitoring framework, and showcase monitoring features in a simulated system using real world infrastructure maintenance scenarios.**

***Keywords*—cyber-physical-social system, service monitoring, smart city management, event processing**

I. Introduction

Modern computing systems do not only include machines (e.g., software and *things*) but also humans as active building blocks (e.g., [1]), which we refer to as *compute units*. Although we have relied on machines (e.g., software-based services with cognitive capabilities) to solve complex problems, many challenges (e.g., requiring creativity and intelligence) require humans working with machines to provide efficient solutions [2]. Moreover, with easier provisioning techniques for human capabilities, we have seen an increasing integration between human capabilities into existing computing systems to create the so-called Cyber-Physical-Social Systems (CPSSs) [3].

Such CPSS can be seen in Internet of Things (IoT) and Cloud computing adopted for vital applications in smart cities (e.g., [4]), where Cloud services can be leveraged to process data from IoT systems with rather limited capabilities (e.g., with respect to storage, processing, and energy). Furthermore, by utilizing services virtualization (e.g., [5]), these IoT Cloud services can be fused with human-based services, hence enabling the orchestration of capabilities of humans, software, and *things* for solving a complex problem.

Monitoring and analyzing metrics for CPSSs in such a smart city setting are inevitable steps for effective smart urban services management and governance. Monitoring and analysis tools provide insights to plan, manage, and adapt the systems to fulfill quality requirements. However, the diversity of involved units in CPSSs introduces challenges for monitoring such systems. Existing monitoring systems traditionally deal with homogeneous units, for example, infrastructure/platform monitoring systems (e.g., [6]), software-based services monitoring systems (e.g., [7]), and IoT monitoring systems (e.g., [8]). To the best of our knowledge, currently no monitoring system exists that deals with thing-based, software-based, and human-based compute units in an integrated manner.

Moreover, different types of compute units have different lifecycles, which require different measurement techniques, monitoring cycles and events. Different unit types may also have similar metrics, but different semantics interpretation, which needs to be correlated. Furthermore, monitoring CPSSs require us to interface the underlying diverse resources as well as the application platforms. Our work presented in this paper tackles to above-mentioned issues. Our goal is to provide a generic monitoring framework for CPSSs, which captures and processes metrics from diverse compute units, e.g., sensors, actuators, gateways, software services, and human-based compute units. Our framework provides a necessary foundation for optimizing smart services. For characterizing behaviors of units in various CPSSs, we propose a metric model to handle metrics with different semantics. Furthermore, we introduce a framework for consolidating different underlying monitoring interfaces.

The salient contributions of our work are as follows:

a) We propose metric models that are necessary to understand and monitor various facets of CPSSs.
b) We bring into effect the notion of Quality of Data (QoD) for monitoring data enabling effective monitoring in CPSSs.
c) We propose a framework and implement a prototype of a monitoring system for coordinated CPSSs.

Furthermore, we evaluate our framework by exemplifying our model on a real-world scenario in smart city infrastructure maintenance and conduct experiments for monitoring such system in a simulated environment.

The rest of this paper is organized as follows: Section II discusses the background, the motivation, and related work. In Section III we present our proposed models that describes CPSS metrics, and the concept of Quality of Data (QoD) for CPSS monitoring. Section IV presents our approach to consolidate various monitoring sources and discusses our proposed monitoring framework. Section V presents our prototype implementation and discusses some experiments based on real-world scenarios to exemplify our approach. Finally, we conclude our paper and outline some future work in Section VI.

II. Background, Motivation, and Related Work

A. Background

In this work, a Cyber-Physical-Social System (CPSS) is a system comprising of three intertwining subsystems: (i) *the human-based systems*, i.e., the social system containing human actors and their interconnected devices/agents and/or social platforms providing human-based services, (ii) *the software-based systems*, i.e., the cyber world providing software-based

The work mentioned in this paper is partially supported by the EU FP7 FET SmartSociety (http://www.smart-society-project.eu/). The first author of this paper is partially supported by the Vienna PhD School of Informatics.

services including the underlying infrastructures and platforms, either on-premise or in the Cloud, and (iii) *the thing-based systems*, i.e., the physical world that includes sensors, actuators, gateways and the underlying infrastructures at the edge.

Technologies used in CPSSs have evolved from the traditional Cyber-Physical Systems (CPSs) [3], which consist of smart embedded systems interconnected to the cyber world. Recent developments of CPS intensively include human actors in a socially connected world, hence showing the advent of Cyber-Physical-Social Systems (CPSSs). A representative use-case of such CPSSs for smart cities can be seen in participatory sensing platforms [9], where citizens are involved for sensing and collecting data for various domains, such as urban planning, e.g., [1], and environmental monitoring, e.g., [4].

The emergence of such mixed human-machine settings can also be seen from different directions. In typical Process-Aware Information Systems (PAISs), the construct of human-based tasks are used to integrate human-based and software-based services into processes, e.g., [10]. In the cloud, we have also seen various sources of human-based compute units such as crowdsourcing marketplaces, and social networks being utilized as active compute units, e.g., [2]. The unification of human-based computing technology and Cloud technology has been carried out using various approaches, such as by abstracting human as a programmable unit, e.g., [11], or by utilizing crowdsourcing APIs, e.g., as in [12]. Furthermore, technologies integrating sensors and actuators into processes, e.g., [13], enable the orchestration of thing-tasks together with software-tasks and human-tasks.

We employ the notion of Cyber-Physical-Social System (CPSS) to approach these diverse systems to make more homogeneous abstractions and conceptualize similar characteristics of these systems so that we are able to develop a general monitoring techniques for these systems.

B. Architecture of Centralized Orchestration of CPSSs

We focus on a class of CPSSs employing a centralistic approach, i.e., systems that have a role of orchestrators to control and manage tasks distribution and execution. Such a class can be found in coordinated CPSs and coordinated IoT Cloud systems, as well as in PAIS with human-based and/or thing-based tasks. Furthermore, we envisage CPSSs as systems virtualizing the capability of humans, software, and *things*, as services [5]. As shown in Fig. 1, in such a service-oriented architecture, an orchestrator coordinates the assignment and distribution of software-, human-, and thing-based tasks to the available and suitable services.

While the system is running, IoT gateways send sensor data from the *things* through a sensor data bus (e.g., a messaging bus using CoAP, MQTT, or XMPP) to software-based services, or to human-based services utilizing a human-friendly dashboard shown by a UI agent. In a typical scenario, during the execution of the tasks, a human- or software-based service may decide to make adjustments on the thing-based systems (e.g., sensor update rates). Such adjustment requests can then be translated (e.g., by an orchestrator) into thing-based tasks and sent to the corresponding machine.

Our monitoring framework proposed in this paper operates on events and metrics captured from all CPSS subsystems through the underlying monitoring tools. Some of such monitoring tools are discussed in Section II-D.

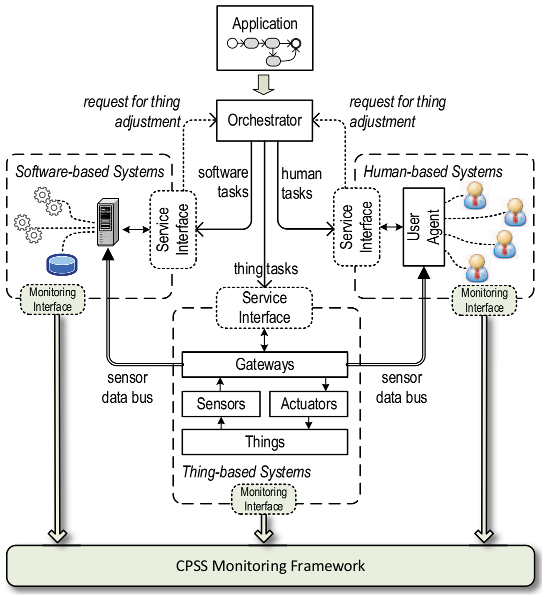


Fig. 1: An Architectural View of orchestrated CPSSs

C. Motivation

Let us consider a scenario in a smart city setting where facilities' maintenance is conducted pro-actively by predicting a possible facility breakdown. For this infrastructure maintenance scenario to work, sensors are installed to monitor facilities. These sensors capture events occurring on the facilities, which are then streamed to a cloud-based data processing center. Software-based data analytic services are configured on the data processing center to analyze the event streams. When the software-based analyzers detect a suspicious event, the event is escalated to a group of human experts who analyze the event further. In the case of the occurrence of an actual incident, an incident ticket is created. In some cases, the human experts may also decide to reconfigure the IoT-subsystem, e.g., changing sensor data rate, deploying more sensors, etc.

Our goal is to provide a monitoring tool for such systems, which is a crucial element for smart services development in cities. Monitoring such complex CPSSs deals with a multitude of problems. Our work is motivated by the following problems:

Problem 1: Each subsystem in a CPSS brings along various metrics, which have corresponding metrics from other subsystems albeit having different definitions. For example, we could define the availability, utilization, and cost metrics for humans; however, their interpretation and measurement differ from the corresponding metrics for software- and thing-based systems. To enable system-wide monitoring, we need models and methods to relate the corresponding metrics and bring them together as a unified metric. In Section III-A2, we employ different classes of metric measurement to address this issue.

Problem 2: Metrics from different subsystems of a CPSS, although having related definitions, may have different qualities, e.g., with respect to the data rate and accuracy. Furthermore, different monitoring clients may require different quality of monitoring data. For example, a human-based client may prefer non-intrusive data (e.g., low data rate), while a software-based client may require much more frequent data. In Section III-B, we bring along some of the concepts of Quality-of-Data and apply them in the context of CPSS monitoring.

Problem 3: A CPSS involve diverse underlying technologies, e.g., with respect to the compute units types and their communication, which have to be taken into account by the monitoring system. We propose an agent-based multi-tier approach to deal with such heterogeneity, shown in Section IV.

D. Related Work

a) Monitoring Framework: Many techniques and tools for machine-based computing units have been developed for monitoring on various layers [14]. Monitoring tools on traditional distributed systems, i.e., grids and clusters, e.g., [6], have been extended to cope with the Cloud characteristics [14]. Unlike the machine-based counterpart, there are not so many works carried out for monitoring the execution of human-based computing. The focus of existing research in this area is to develop quality improving techniques, which are typically domain-specific [15]. In PAIS with human tasks support, some tools, e.g., [16], are provided to monitor human tasks and their execution states, and to allow administrators to perform manual actions when necessary. However, still there is a lack of generic tools for online monitoring of various aspects of a collective of human-based compute units and human tasks executions. We leave this issue for future work. We position these machine-based and human-based monitoring tools as the underlying interfaces for capturing events and metrics to be used in system-wide CPSS monitoring.

b) Characterizing CPSS: Metrics in software-based systems, including the underlying infrastructures, have been extensively studied, e.g., in [17]. Although much less studied, metrics for people as computing units have also been proposed, e.g., in [15]. In thing-based systems, many works center around streams of metrics produced by the sensors (e.g., temperatures, locations, etc.). However, not many papers have been published to define the metrics which represent the qualities of the system itself. Some papers, e.g., [18], propose metrics for improving the quality of the thing-based systems.

Quality of Data (QoD) plays a crucial role, especially in systems such as CPSSs, where electronic data are ubiquitous. The majority of authors in the domain of QoD typically consider QoD from a basic set of quality dimensions: accuracy, completeness, consistency, and timeliness [19]. In our work, we apply the accuracy and timeliness (data rate and freshness) dimensions of QoD to enable more efficient data delivery in CPSS monitoring.

The notion of composable metrics have also been proposed, e.g., in [20]. These works focus on composable metrics in homogeneous system. Our work presented here deals with correlating and composing metrics from thing-based, software-based, and human-based systems.

III. Metrics and Quality of Data

A monitoring system centers around metrics, which need to be captured, analyzed, and delivered to the clients. Existing metric constructs in monitoring systems need to be extended in order to engage with the dynamics of CPSSs. Furthermore, the concept of Quality of Data (QoD) can be leveraged to deal with the problem of different metrics qualities, as well as different monitoring requirements in CPSSs. In this section, we discuss various metric suitable for CPSSs, how to measure them, and the application of QoDs in CPSS monitoring.

A. Metrics

1) Useful Metrics for CPSSs: Metrics of a CPSS may contain aggregation of metrics from machine-based units (i.e., from the thing-based, and software-based systems) and human-based units. Metrics traditionally found for machine-based units may have the equivalent for human-based units with similar meaning. In Table I we show several metrics and some possible definitions of the metrics for machine-based and human-based units as well as the aggregation definition for CPSS, where u is a unit and t is a task. Other definitions may also be employed according to the problem domain.

2) Metric Measurement: To capture the dynamics of a CPSS, we define four classes of metrics, namely *raw metrics*, *composite metrics*, *state metrics*, and *correlation metrics*. The first two classes of metrics are commonly found in monitoring system, e.g., [20], [21]. However, due to the diversity of CPSSs, we introduce the state metrics and correlation metrics.

a) Raw metrics: are metrics that capture information from the underlying resources. These metrics can be collected using different means depending on the underlying monitors. For example, typical raw metrics from a cloud service can be obtained using an exposed API. Generally, raw metrics can be obtained in two manners, by pulling periodically, or by using a publish-subscribe approach. In human-based computing platforms, some raw metrics may be directly provided by the platform (e.g., human-based unit acceptance rate and location). However, some other raw metrics may not be so straightforward to obtain but can be inferred from the events generated by the platform. For example, to obtain an assignment count metrics for each human-based unit, a monitoring agent can subscribe to task assignment events, and count the number of assignments for each unit.

b) Composite metrics: can be defined using an arithmetic expression, an aggregate function, or a custom composite function of other metrics. For example, the utilization of a thing-based system containing a set of sensors can be measured by aggregating the number of sensors actively sending streams of data in a particular time frame, e.g., using moving average aggregation on a sliding window.

c) State metrics: define measurements related to the state transitions during runtime. In typical human-task and process-based systems, we often deal with the underlying monitoring tools that are capable of capturing events representing transitions from one runtime state to another, e.g., [16]. We use state metrics for capturing metrics related to the state transitions, such as how many times a compute unit enters a particular state, how long a compute unit stays in a state for a particular time window, and so on. Consider, for example, a human-based task running on a process-based system [10]. On a particular process instance, the human-based task may be transient from one state to another, e.g., *created*, *assigned*, *running*, *paused*, and *finished* states. Using a finite-state automata model, we can then define some primitives for a given entity e and state s, e.g., $time(e, s)$, $count(e, s)$, $duration(e, s)$, which define the last timestamp e enters s, the number of times e enters s, and the total duration of e staying in s respectively. We can also extend these primitives to perform the measurement on a particular time window, e.g., the last 24 hours.

d) Correlation metrics: allow computing together metrics with different semantics from different sources. We support this type of metrics to tackle the problem of combining together metrics from diverse compute units. To correlate diverse metrics we need to specify three things: the sources of metrics or events that we want to correlate, the normalization

Metrics	Machine-based Definitions	Human-based Definitions	CPSS Aggregation	Description
$Util(u)$	$CPUUsage(u)$	$\frac{Active(u)}{\textsc{maxActive}}$	$\frac{\sum_{\forall u \in \mathcal{U}} Util(u)}{\lvert\mathcal{U}\rvert}$	$Util(u)$ = utilization of unit u, $Active(u)$ = the total duration of active time in the past 24 hours of unit u, $\textsc{maxActive}$ = threshold for maximum active time per human unit, $\mathcal{U}$ = the set of units in the CPSS
$RT(t,u)$	$FT(t,u) - AT(t,u)$	$FT(t,u) - AT(t,u)$	$\max_{\forall u \in \mathcal{U}} FT(t,u) - \min_{\forall u \in \mathcal{U}} AT(t,u)$	$RT(t,u)$ = response time for task t by unit u, FT = finish time, AT = assignment time
$Cost(t,u)$	$CostT(u) \cdot RT(t,u)$	$CostA(t,u)$	$\sum_{\forall u \in \mathcal{U}} Cost(t,u)$	$Cost(t,u)$ = cost for executing task t by unit u, $CostT$ = cost per time unit, $CostA$ = cost per task assignment

TABLE I: Metric Examples

function that should be applied so that the source metrics have uniform semantics before we combine them together, and the aggregation function to calculate the value of the new correlated metric.

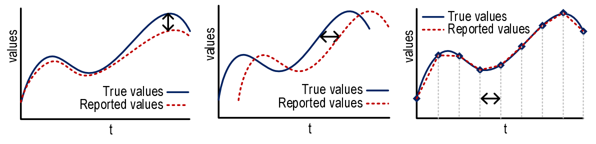
3) Complex Metric Samples: To demonstrate these metric classes, consider how we can measure the *utilization* metrics of the system for the sensor data collection and analytic in infrastructure maintenance scenario discussed in Section II-C. For example, on the thing-based and software-based systems, the utilization of sensor i, $SensorUtil(i)$, and the utilization of machine j running a data analytic service, $MachineUtil(j)$, can be obtained using raw metrics, typically using a certain API exposed by the platform.

However, for human-based systems, the utilization measurement can be more complicated. First of all, we have to define what *utilization* means for human-based compute units. In the case of humans, there is no notion of CPU usage as traditionally found for software-based systems. Without loss of generality, let us, for example, define the utilization of a human-based compute unit as the time the unit spent for executing all assigned tasks in a given time window w. Hence, using the state metrics, we can the measure the utilization of human-based compute unit k as $HumanUtil(k) = duration(k, Running, w)$.

Often we need to monitor system-wide correlated metrics instead of metrics for a particular unit. For example, it can be necessary to see the overall average utilization from all the three subsystems. Or we can monitor the *top-k* units with the highest utilization, regardless they are thing-based, software-based, or human-based units, in order to identify bottlenecks. To obtain such metrics, we need to combine together $SensorUtil$, $MachineUtil$, and $HumanUtil$ metrics, and resolve any semantics differences among them. This is where our correlation metric model becomes practical. Firstly, we could normalize the *HumanUtil* metric so that it has the same value range and it has an acceptable similar meaning compared to the $SensorUtil$ and $MachineUtil$ metrics. One reasonable normalization of the human utilization against the machine utilization is to set a maximum threshold of working time that a human-based unit may work in the past 24 hours, that is $HumanUtil'(k) = duration(k, Running, 24hours)/\textsc{max}$. This definition surely is not the sole definition of human utilization, different definitions may be applied according to

B. Quality of Data

Collecting and processing monitoring data on large scale systems such as an IoT Cloud system introduces an inherent problem, that is, a huge number of monitoring data lead to



(a) Accuracy (b) Freshness (c) Data Rate

Fig. 2: Quality of Data in CPSS Monitoring

high network utilization and heavy data processing. On the contrary, the human-based computing counterpart is typically running in a much slower pace due to longer life-cycles, e.g., assignments to a single human unit may take place in the order of minutes, hours, or even days.

We apply the concept of *Quality of Data* (QoD) [19] allowing monitoring clients to specify monitoring requirements as a trade-off of quality and costs. Such QoD-aware monitoring solves the above-mentioned problems in two ways: (i) it allows the monitoring clients/providers to request/produce monitoring data on a lower quality level to reduce costs, and (ii) it allows interweaving monitoring requests on different subsystems having different QoD into similar QoD, hence it becomes reasonable to correlate metrics from those subsystems. We discuss the interpretation of QoD in the context of CPSS monitoring and some use-cases of such QoD as follows.

1) QoD Interpretation for CPSS Monitoring: We focus on three QoD measures, namely *accuracy*, *freshness*, and *data rate* (or *rate* for short) as defined in the following paragraphs and illustrated in Fig.2.

a) Data Rate: The data rate of a monitoring data, $Rate(d)$, represents the frequency on which a monitoring agent should report the data. Many techniques can be used to obtain data on a particular time point, e.g., to use last actual retained data or to use moving average values. When the real data has a lower data rate, the monitoring agent may perform techniques, such as interpolation, estimating the data in-between.

b) Accuracy: The accuracy of monitoring data is derived from the difference between the true value with the value last reported to the client, i.e., given a data, d, the accuracy of the data is defined as $Acr(d) = |v(d') - v(d)|$, where $v(d)$ is the actual value of d and $v(d')$ is its last reported value.

c) Freshness: The freshness of monitoring data defines the timing skew between the true timestamp of the data and the timestamp when the data is reported, i.e., it defines as $Frs(d) = t(d') - t(d)$, where $t(d)$ is the actual timestamp of d and $t(d')$ is the report timestamp.

More formally, given a QoD requirement, $\mathcal{Q} = (R_R, R_A, R_F)$, where R_R is a data rate requirement, R_A is an accuracy requirement, and R_F is a freshness requirement, the monitoring tool must deliver a set of reported data $\mathcal{I}$ from the actual set of data $\mathcal{J}$, that fulfils the following constraints:

$$\begin{aligned}\forall d' \in \mathcal{I}, \forall d \in \mathcal{J}, \quad & t(d'_{i-1}) < t(d_j) \le t(d'_i) \implies \\ & \Big(t(d'_i) - t(d'_{i-1}) \le R_R \quad \wedge \\ & |v(d'_i) - v(d_j)| \le R_A \quad \wedge \\ & t(d'_i) - t(d_j) \le R_F\Big), \end{aligned} \tag{1}$$

where $t(d)$ is the time when the data d is sent, and $v(d)$ is the value of the data d.

2) QoD-aware Monitoring Usages: The usages of QoD-aware monitoring can be seen from two perspectives. First, from the perspective of a monitoring provider, QoD-aware monitoring helps increasing efficiency on resource usage, e.g., data bandwidth. Second, it allows a monitoring client to define more precisely the quality of data they need.

Consider, for example, a human client who wants to monitor system utilization, but he does not want the monitoring reports to be intrusive. Hence, he may want to request utilization data for one hour intervals. However, he does not want to miss rapid changes on the system utilization. Hence, he puts in a data quality requirement that the accuracy of the data he received should not be more than 0.10 points. In this case, the monitoring system delivers the data on (maximum) an hourly rate, but also makes sure that the last reported data does not differ more than 0.10 points from the real value.

3) QoD-Aware Data Delivery: Based on QoD requirements, a monitoring provider may provide a QoD-aware data delivery by optimizing monitoring resources while still fulfilling the constraints as described in Eq. 1. There are many ways to achieve such QoD-aware data delivery. We present an example of such QoD-aware data delivery algorithm (see Algorithm 1), which minimizes the number of messages (i.e., the number of sent monitoring data), while still honouring the QoD requirements. This algorithm defers the sending of data, retain it, and calculate the right time to send the data according to the data rate, R_R, accuracy, R_A, and freshness, R_F, requirements. Here, the RECEIVE function is executed when a monitoring consumer (see Section IV) receives data, and the SEND function sends data to the subscriber.

IV. Distributed Monitoring Framework

Our monitoring framework consists mainly of *monitoring agents* (or agents for short), which provide events and metrics for other monitoring agents, as shown in Fig. 3 (here, a metric is another type of event, in the remainder of this paper we use them interchangeably). Such a distributed and recursive nature of the monitoring agents structure allows our framework to scale according to the scale of the CPSS.

Our framework adopts an event-based approach using the publish/subscribe pattern. Each agent publishes topics that contain metric values for other agents. Each agent can either subscribe to certain topics from other agents, or retrieve metrics from their own adapters connecting to the underlying monitoring tools. Eventually, a client application (or a client, for short) can then consume metrics from one or more agents and use it in the application logic. Fig. 3 also represents an example of an agents topology.

In the following subsections we discuss the construct of monitoring agents and the communication protocol between

Algorithm 1 Algoritm for QoD-Aware Data Delivery

```
Input:
    $\mathcal{Q} = (R_R, R_A, R_F)$                    ▷ The QoD requirements
function RECEIVE($data$)
    RETAIN($data$, MAXRETAINED)
    if $R_A$ is set then
        if $|data - lastSentData| > R_A$ then
            SEND($data$)
            SCHEDULER.CANCELPREVIOUSWAKER( )
            return
    if $R_F$ is set then
        if not $dataChanged \wedge$
          $data \neq lastSentData$ then
            $dataChanged \leftarrow$ True
            $nextWakeTime \leftarrow now + R_F$
            SCHEDULER.CANCELPREVIOUSWAKER( )
            SCHEDULER.WAKEMEAT($nextWakeTime$)
    if $R_R$ is set then
        if $nextWakeTime > now + R_R$ then
            $nextWakeTime \leftarrow now + R_R$
            SCHEDULER.CANCELPREVIOUSWAKER( )
            SCHEDULER.WAKEMEAT($nextWakeTime$)

function WAKE
    $data \leftarrow$ ESTIMATEFROMRETAINEDDATA( )
    SEND($data$)
    if $rate$ is set then
        $nextWakeTime \leftarrow now + rate$
        SCHEDULER.WAKEMEAT($nextWakeTime$)
```

those monitoring agents, as well as some technical considerations for agents' implementation.

A. Monitoring Agent

A monitoring agent is a software component containing a *monitoring producer* (MP), which produces events and metrics according to the context it monitors. Inputs of a monitoring agent come from one or more *monitoring adapters*, which retrieve events and metrics from the underlying resources or application monitors, and/or one or more *monitoring consumers*, which consume events from other agents.

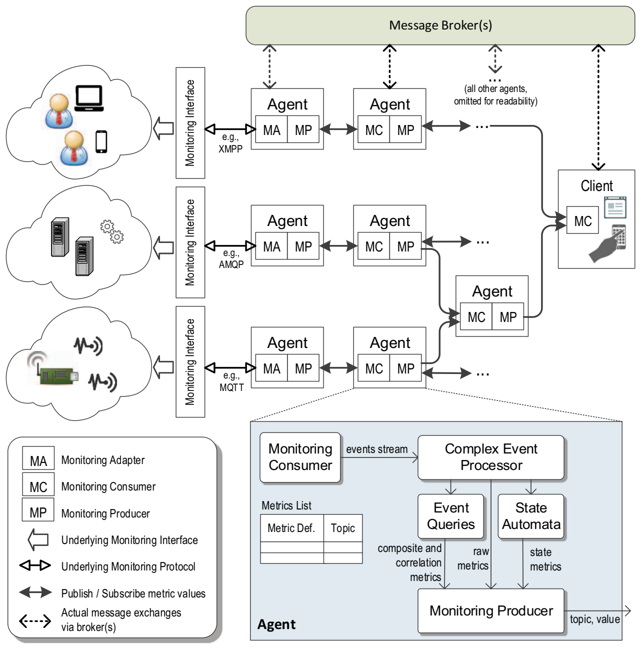


Fig. 3: Monitoring Framework

The *monitoring adapter* (MA) component of an agent, adapts events and metrics captured from a specific monitoring tool provided by the application or resource platform. An MA may retrieve metrics through an underlying protocol provided by the monitoring tool. For example, the presence events of a human-based service can be provided using XMPP. Other publish/subscribe protocols, such as AMQP and MQTT may also be used for retrieving metrics from software-based or thing-based systems. Hence, the implementation of MA is platform-specific, and is beyond the scope of this paper. Moreover, an agent may also consume metrics provided by another agent by implementing a *monitoring consumer* (MC).

Note that the proposed construct of monitoring agents is a conceptual abstraction. On the practical level, multiple agents can be implemented either on a single physical node (e.g., an agent may consume its own metrics to produce more complex metrics), or on multiple nodes. In the case where agents are distributed, they need to communicate each others. The communication protocol between MCs and MPs is discussed in Section IV-B.

A proposed implementation model of an agent is shown in the bottom-right inset of Fig. 3. Here, an agent is implemented using a complex event processor to process event streams retrieved by an MC. A straightforward raw metric can publish directly from the incoming event stream. For composite and correlation metrics, an associated event query can be utilized, e.g., aggregating an incoming event stream using an aggregate function, or combining multiple event streams using a composite expression, or normalizing and correlating multiple event streams. State automata can be employed to listen to state events from the stream and to produce state-based metrics. Each of these produced metric streams are then published by the MP under a specified topic. Proposed models for implementing QoD are discussed in the following subsection.

B. Protocol and Quality-Aware Delivery

There are currently a multitude of protocols supporting the publish/subscribe pattern. Our proposed framework focuses on the abstraction for dealing with monitoring entities involved in a CPSS. Hence, the realization of such an abstraction may use available protocols.

A typical implementation of such publish/subscribe protocols decouples publishers and subscribers by employing a message broker (or a cluster of brokers), which has the logic for routing message exchanges between publishers and subscribers. To the best of our knowledge, currently there are no protocols that have out-of-the-box support for a dynamic message exchange routing which allows one topic to be delivered to multiple subscribers having different quality requirements with respect to the data rate, accuracy, or freshness. However, the implementation of the QoD-aware data delivery may extend available subscription message format, when possible; and then use the custom exchange routing to implement the QoD-aware data delivery algorithm.

The implementation of such QoD-aware data delivery can be done on two sides, i.e on the broker side or on the client (agent) side. Implementing QoD-aware delivery on the broker side is only possible for protocol that supports custom exchange routing, for example on an AMQP-based implementation (e.g., RabbitMQ). For a protocol that does not allow a custom exchange routing, e.g., MQTT, the QoD-aware delivery implementation is only feasible on the agent side.

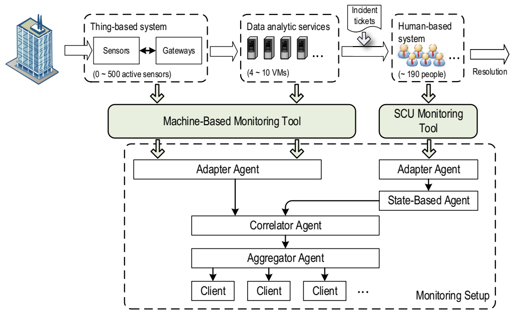


Fig. 4: Monitoring Experiments Setup

In agent-side QoD-aware delivery, the publisher must know which subscribers are listening to topics with the required QoD, so that the publisher knows exactly to whom and when messages should be sent. Hence, the agent-side QoD-aware delivery breaks one of the original goals of the publish/subscribe pattern, i.e., the decoupling of publishers and subscribers. Moreover, the broker-side QoD-aware delivery puts all the QoD processing logics on the broker, hence making the implementation of agents simpler. However, the agent-side QoD-aware delivery allows more optimized metrics publication, because it allows more granular control on when a publisher should publish a metric, instead of publishing on every produced metric values.

V. Implementation and Experiments

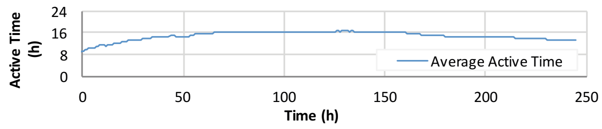
We have prototyped our proposed monitoring framework and integrated the prototype into our platform, *Runtime and Analytics for Hybrid Compute Units* (RAHYMS)[1]. This platform is open-source and implemented using Java and provides tools for simulating hybrid systems based on the GridSim toolkit [22]. We employed Esper[2] as the complex event processors, and the evaluated metrics are translated into Esper event processing language (EPL). For the QoD processing, we implemented both, broker-based, and agent-based approaches.

A. Experimental Setup

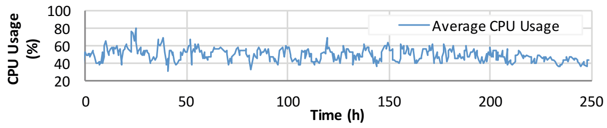
We setup our experiment based on the smart city infrastructure maintenance scenario discussed in Section II-C. To demonstrate the diversity of the underlying systems we monitor, we setup experiments employing monitoring data from a thing-based and software-based system, as well as a human-based system. We use data from a realistic M2M DaaS experiment executed using an elastic cloud service framework, ADVISE [23]. The datasets of this experiment are available online[3]. During the execution we injected events that create incidents, which should be further investigated by human-based units. Based on a real incident management system, we simulate the composition and execution of the so-called *social compute unit* (SCU), which contains a group of experts that can be composed and dissolved on-demand [24].

For evaluation purposes, we created adapters for the underlying monitoring tools to retrieve the recorded monitoring data from the aforementioned setup. We implemented generic classes of agents, namely state-based agent, correlator agent, aggregator agent, and client agent as shown in Fig. 4. Together with a messaging broker, these agents are then incorporated

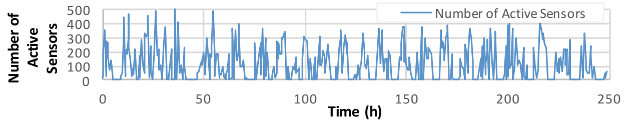
[1]https://github.com/tuwiendsg/RAHYMS
[2]http://esper.codehaus.org
[3]https://github.com/tuwiendsg/ADVISE/tree/master/data/M2MApp



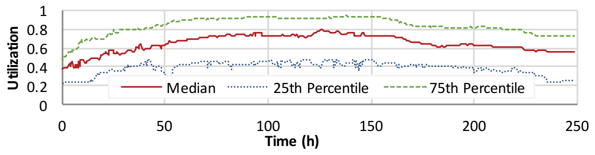
(a) Human-Based Units Utilization (average active time in the last 24 hours)



(b) Software-Based System Utilization (average CPU usages on all data collection and processing machines)



(c) Thing-Based System Utilization (the number of active sensors)



(d) Cyber-Physical-Social System Utilization

Fig. 5: Correlated Utilization Metrics

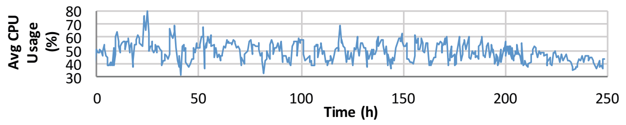
as grid entities in the GridSim framework. Interested readers may get more detail information on the implementation and experiments from the supplemented material online[4].

B. Experiments

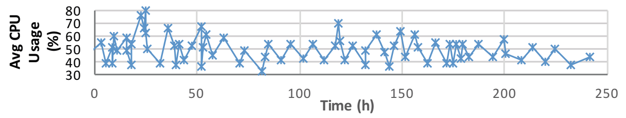
a) Complex Metrics Experiments: Traditional monitoring systems typically deal with homogenous systems, where correlating similar metrics with different semantics from different subsystems is difficult. In our first experiment, we demonstrate the capability of our framework for capturing complex metrics derived from the correlation of metrics of different subsystems. Here we use the *utilization* metrics as discussed in Section III-A3. The utilization of human-based units is derived from their active hours during the last 24 hours. The utilization of the software-based system is obtained from the CPU usages of the machines running the software-based services. On the thing-based system, we capture the snapshots of the numbers of active sensors at any particular time. These three different metrics are then correlated, i.e., normalized and combined into one metric stream, so that further unified operations becomes possible. We then applied stream data aggregation operation (median and percentiles) to obtain new aggregated utilization metrics, which represent the behavior of the overall system.

An XML definition of such correlated utilization metric is provided in the online supplemented materials. Such a metric definition is then transformed into EPL and deployed into a complex event processor. We deploy the metric processors into our prototype implementation running the aforementioned infrastructure maintenance scenario and capture the resulted metrics as shown in Fig. 5. The streams of CPU usage and

[4]http://dsg.tuwien.ac.at/prototypes/CpssMonitoring/



(a) CPU Usages without QoD



(b) CPU Usages with QoD (rate = 12h, accuracy = 10.0)

Fig. 6: Quality of Data (QoD) Experiments

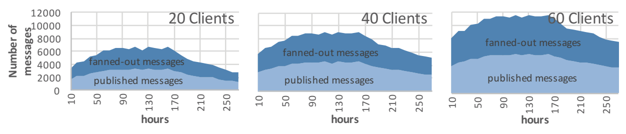
active sensors metrics are fluctuated much rapidly as shown in Fig. 5b and Fig. 5c, while the active time of human-based units are more steady (Fig. 5a). We remove the data captured from the first 24 hours to avoid the effect of incomplete initial collection of human-based units activities. The outcome of the correlated utilization metrics shown in Fig. 5d.

b) QoD Experiments: In CPSS monitoring, different monitoring clients may require different data qualities. In this experiment, we would like to show the benefits of QoD-aware data delivery provided by our framework, especially for the monitoring clients with respect to the intrusiveness of the data. We deploy two monitoring clients that subscribe for CPU usage metrics. The first client subscribes without QoD requirements, while the second one emulates a human-based client, who wants only to receive updates on every 12 hours, while still requiring data accuracy of 10 points.

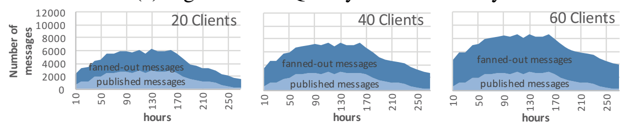
Here we use again a similar setup as in the first experiment, and apply the algorithm for QoD-aware data delivery shown in Algorithm 1 on the message broker. The estimation of the QoD-aware data is using moving average to calculate the data value on a particular point. As can be seen in Fig. 6, the data received by the second client is much more sparse then the first one, as it requests to receive data on every 12 hours basis. However, on the events where the metric fluctuates very rapidly (i.e., more than the requested 10 points before the 12 hours duration dues), the clients receives more data.

c) Comparing Implementations of QoD-aware Data Delivery: As discussed in Section III-B3, QoD-aware data delivery can be implemented either on the broker-side or on the agent-side. In this experiment, we want to compare both approaches, and study the costs and benefits, especially from the perspective of monitoring providers. Here we experiment using similar setup as in Experiment 1, and apply the QoD-aware data delivery algorithm on either the broker or the agents and evaluate the results based on the number of messages, which represent the monitoring overhead for the overall system. The messages are counted and classified in two classes, the published messages (i.e., messages sent out by agents to the broker), and fan-out messages (i.e., message sent out by the broker to consumers).

First, we run the experiments using varying number of clients, i.e., 20, 40, and 60 clients, each with varying QoD requirements. As shown in Fig. 8, the broker-based quality-aware delivery is more efficient compared to the agent-based counterpart with respect to the number of total messages. This is due to the fact that the number of published messages on the broker-based quality-aware delivery is constant regardless the number of clients; while on the agent-based quality-aware delivery, the published messages are addressed to each clients



(a) Agent-based Quality-aware Delivery



(b) Broker-based Quality-aware Delivery

Fig. 7: Number of Messages in Quality-Aware Delivery

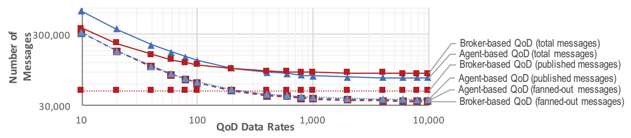


Fig. 8: Number of Messages in Varying Data Rates

with particular QoD requirements.

However, the agent-based quality-aware delivery can be more efficient than the broker-based one. We setup again the experiments with 10 clients. We run several set of experiments with this clients, each set with different data rates requirements as shown in Fig. 8. Here we can see that the agent-based quality-aware delivery is more efficient in low data rate requirements, because the number of its published messages becomes lower than the number of published messages in the broker-based counterparts. The cross points of these two approaches represent the data rates that are roughly equal to the mean original data rate (i.e., the data rate of messages sent out by agents if there is no QoD requirements).

VI. Conclusions and Future Work

In this paper we present our approach for monitoring Cyber-Physical-Social Systems (CPSSs), which is necessary for optimizing smart services in complex settings such as smart cities. We tackle challenges to deal with heterogenous events and metrics emitted by those diverse subsystems. Moreover, we use Quality of Data (QoD) to enable more efficient monitoring of CPSSs according to the consumer's requirements.

We present our CPSS simulation tool, and implement a prototype of our monitoring approach. We run monitoring experiments using monitoring data derived from real world scenarios. Our experiments demonstrate that our framework is useful to model and measure complex metrics from a running CPSS. Furthermore, we show benefits for both monitoring clients and providers in applying QoD-aware data delivery on CPSS monitoring.

Our work presented in this paper is part of our ongoing research on dependable hybrid human-machine computing. Future works include modeling and measuring various dependability metrics such as availability, performance, and quality of results in the context of CPSS.

References

[1] Mina Sakamura, Tomotaka Ito, Hideyuki Tokuda, Takuro Yonezawa, and Jin Nakazawa. Minaqn: web-based participatory sensing platform for citizen-centric urban development. In *UbiComp'15*. ACM, 2015.

[2] Anhai Doan, Raghu Ramakrishnan, and Alon Y Halevy. Crowdsourcing systems on the world-wide web. *Communications of the ACM*, 54(4):86–96, 2011.

[3] Fei-Yue Wang. The emergence of intelligent enterprises: From CPS to CPSS. *Intelligent Systems, IEEE*, 25(4):85–88, 2010.

[4] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami. An information framework for creating a smart city through internet of things. *Internet of Things Journal, IEEE*, 1(2):112–121, 2014.

[5] Muhammad ZC Candra, Rostyslav Zabolotnyi, Hong-Linh Truong, and Schahram Dustdar. Virtualizing software and human for elastic hybrid services. In *Advanced Web Services*, pages 431–453. Springer, 2014.

[6] Nagios. Nagios - the industry standard in it infrastructure monitoring. Website, 2015. http://www.nagios.org/.

[7] J. Lesbegueries, A. Ben Hamida, N. Salatgé, S. Zribi, and J.-P. Lorré. Multilevel event-based monitoring framework for the petals enterprise service bus: industry article. In *DEBS'12*. ACM, 2012.

[8] S. Frischbier, E. Turan, M. Gesmann, A. Margara, D. Eyers, P. Eugster, P. Pietzuch, and A. Buchmann. Effective runtime monitoring of distributed event-based enterprise systems with asia. In *SOCA'14*. IEEE, 2014.

[9] Jeffrey A Burke, Deborah Estrin, Mark Hansen, Andrew Parker, Nithya Ramanathan, Sasank Reddy, and Mani B Srivastava. Participatory sensing. *Center for Embedded Network Sensing*, 2006.

[10] Ashish Agrawal, Mike Amend, Manoj Das, Mark Ford, Chris Keller, Matthias Kloppmann, Dieter König, Frank Leymann, et al. WS-BPEL extension for people (BPEL4People). *V1. 0*, 2007.

[11] Daniel W Barowy, Charlie Curtsinger, Emery D Berger, and Andrew McGregor. Automan: A platform for integrating human-based and digital computation. *ACM SIGPLAN Notices*, 47(10):639–654, 2012.

[12] Amazon. Amazon mechanical turk. Website, 2015. http://www.mturk.com/.

[13] Patrik Spiess, H Vogt, and H Jutting. Integrating sensor networks with business processes. In *Real-World Sensor Networks Workshop at ACM MobiSys*, 2006.

[14] Giuseppe Aceto, Alessio Botta, Walter De Donato, and Antonio Pescapè. Cloud monitoring: A survey. *Computer Networks*, 57(9):2093–2115, 2013.

[15] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H.R. Motahari-Nezhad, E. Bertino, and S. Dustdar. Quality control in crowdsourcing systems: Issues and directions. *IEEE Internet Computing*, 17(2):76–81, 2013.

[16] IBM. Monitoring and administering human tasks with websphere business monitor. Website, 2009. http://www.ibm.com/developerworks/websphere/library/techarticles/0904_xing/0904_xing.html.

[17] Felix Freiling, Irene Eusgeld, and Ralf Reussner. Dependability metrics. *Lecture Notes in Computer Science. Springer-Verlag, Berlin, Germany*, 2008.

[18] Karthik Lakshmanan, Dionisio De Niz, Ragunathan Rajkumar, and Gines Moreno. Resource allocation in distributed mixed-criticality cyber-physical systems. In *ICDCS'10*, pages 169–178. IEEE, 2010.

[19] Carlo Batini, Cinzia Cappiello, Chiara Francalanci, and Andrea Maurino. Methodologies for data quality assessment and improvement. *ACM Computing Surveys (CSUR)*, 41(3):16, 2009.

[20] Alexander Keller and Heiko Ludwig. The wsla framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[21] A. Mos, C. Pedrinaci, G. Alvaro Rey, J. Manuel Gomez, D. Liu, G. Vaudaux-Ruth, and S. Quaireau. Multi-level monitoring and analysis of web-scale service based applications. In *ICSOC/ServiceWave'09*. Springer, 2010.

[22] Rajkumar Buyya and Manzur Murshed. Gridsim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and computation: practice and experience*, 14(13-15):1175–1220, 2002.

[23] G. Copil, D. Trihinas, H-L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. Dikaiakos. Advise–a framework for evaluating cloud service elasticity behavior. In *Service-Oriented Computing*. Springer, 2014.

[24] Bikram Sengupta, Anshu Jain, Kamal Bhattacharya, Hong-Linh Truong, and Schahram Dustdar. Who do you call? problem resolution through social compute units. In *ICSOC*, pages 48–62. Springer, 2012.