Edge-of-Cloud Fast-Data Consolidation for the Internet of Things

Gilles Privat, Laurent Lemke Orange Labs, Grenoble, France [firstname.lastname@orange.com] Pascale Borscia, Marc Capdevielle Orange Labs, Toulouse, France [firstname.lastname@orange.com]

Abstract—Control-oriented IoT applications that require time-sensitive information about the entities they control need to operate in their proximity, from the edge domain of the network, rather than from a remote cloud. We describe how different types of data filtering and consolidation may get integrated in a generic IoT infrastructure in order to support this. We describe the processing and data models that are required for consolidating and locally maintaining, if need be, the corresponding information, combining stateless event stream processing with the use of discrete-state-based models. We describe how the corresponding distribution of resources maps to both edge and cloud platforms and evaluate the tradeoffs that come with this distribution.

Keywords—Internet of Things; Complex Event Processing ; Fog Computing

I. INTRODUCTION

The availability of sensors and actuators connected through open networks opens up a much-touted cornucopia of new Internet of Things (IoT) application opportunities, yet it does also require new infrastructures and platforms to collect, process and maintain these data in support of the corresponding applications.

The most obvious challenge facing these infrastructures results from the sheer volume of such plentiful IoT sensors, capturing all kinds of data in various physical modalities: it becomes essential to filter these data locally before passing them on to cloud processing in order to save network and storage resources, by avoiding to clog WANs and remote clouds with unchanging, irrelevant or insignificant data that is tantamount to noise.

Using data locally, back to back with these early stages of processing, is also required when taking immediate actions on these data. Most mainstream IoT applications rely exclusively on upward data collection from sensors, with actions, if any, being effected non-real-time by human operators. Automatic control and, more generally, cyber-physical systems integrating it, still have a long way to go in their journey from custom-designed one-of-a-kind solutions to automatically configurable applications on top of generic platforms [1]. These are the applications we intend to address, neglected as they have been from mainstream IoT and ambient intelligence research so far. Some of these control-oriented applications cannot rely exclusively on filtering "along the flow" a stream of incoming

events: they need to know in real time in which state, in a system-theoretic sense (see section V), the entities to be controlled are at any given time in order to control them, because the action to be performed will depend on this state. There is a rich body of know-how and models to draw upon, coming from the embedded/reactive systems literature. IoT applications need to adapt this know-how in view of the strong incentives that now push them to operate on top of shared infrastructures, in stark contrast to the way closed and dedicated embedded systems would traditionally work.

Crucially, the local vs. remote or edge vs. cloud tradeoff for IoT data processing is not meant to be addressed on a perapplication basis: it should be part of an overall implementation tradeoff and a required flexibility for a solution that will integrate this IoT data processing in a complete mediation/brokerage infrastructure, offering all attending configuration and administration facilities. This is part of what the FIWARE infrastructure¹, an open source middleware platform, affords, and our work has taken place in this framework.

Applications using such infrastructures are entitled to get access to data at different levels of aggregation/abstraction along their successive stages of processing, with homogeneous standard interfaces. Control-oriented applications that need to operate with bounded latencies may require to be co-hosted on edge platforms, close to the field, rather than on a remote backend cloud.

We describe this architecture first functionally in part III. Section IV describes the classical event stream processing that may correspond to the first stage of edge data filtering. Section V gives a brief rationale for the use of state models and how the data consolidation supported by these models would become part of an overall IoT infrastructure. The overall architecture taking into account the implementation platforms and corresponding tradeoffs are described in section VI.

II. RELATED WORK

A. Context middleware

Context management originates from a traditional view of telecom applications that rested on a clear-cut separation of

Work partially funded under the IoT chapter of the EU FIWARE project (www.fiware.org).

https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/FIWARE_ Architecture

content and context [2], such as e.g. for regular audio interpersonal communication the user's location or activity being obviously context, whereas her utterances where just as obviously content. In this strict context = content view, telecom content used to be just carried like a sealed postal envelope from one point to another and did not undergo any kind of processing other than regular coding for compression or network adaptation, whereas context could undergo several stages of deep processing to extract meaningful data. This distinction is bound to vanish in IoT applications², where context becomes content and content becomes context in turn [2]. In this perspective, the vast body of work on multilayered context management infrastructures [3] inherited from the era of ambient intelligence research should carry over to IoT infrastructures, with deep processing of sensor data being performed on sensor input, irrespective of whether it operates on content (primary) or context (ancillary) inputs.

B. From Big-data to Fast-data, or the other way round?

The recent swell of big-data/analytics activity has tended to bias IoT applications away from both the embedded/real-time culture and the requirements of their physical grounding. On the positive side, big-data leads to pool resources from different applications and environments, and thus to open up the siloes that enclose present-day vertically integrated M2M/IoT applications, but this data pooling occurs in the latest stages of data acquisition, very far from real-time requirements and from where the data could be used for control with predictable timing.

Ironically, "fast-data" appeared a few years ago [4] as a rather shallow companion buzzword denoting the need for bigdata to become more "real-time", in a weak sense which just meant processing data "in-flight", along the flow, as streams rather than as batches, so as to be able to take decisions more quickly on the basis of this data. This may have been most visibly advertised for such big data applications as market analysis or financial services that were far from real-time (in the hard, industrial sense) to start with.

Of course applications that are, by their very nature, real-time in a rigorous sense, such as automatic control, have always been "fast-data" applications long before this phrase appeared, as soon as it was understood that control was information. As they evolve to become networked IoT applications operating on top of shared infrastructures, these "fast-data" applications will in turn become more "big-data" in the sense of sharing their data and pooling it with other data sources, making it available for both their own instantaneous real-time use and the potential use of other applications.

C. Multi-sensor Data Fusion

This is once more the case of a very mature and extensive body of knowledge [5], mostly developed in the confines of military research, or more benignly by roboticists, which has to be rediscovered in order to be put to a new use, in the very

² It is also becoming less of an ironclad separation for (non-IoT) interpersonal applications, especially with OTT operators routinely analyzing and exploiting for targeted advertising the content of their users' e-mails or web searches, if not (yet) the content of their interpersonal audio or video communication.

different context of shared networked sensors of IoT. Traditional sensor fusion problems were addressed, just like automatic control, on a case by case basis with dedicated and fixed sensor configurations, possibly taking inspiration from the human perceptual system (which routinely performs some very clever tricks of multimodal sensor fusion of its own between e.g. its auditory and visual subsystems). We explain in section IV how Complex Event Processing (which was so far mostly used in very different fields like the analysis of financial data or customer data), may be used as a more general-purpose tool to approach this problem for multi-sensor configurations that will be highly variable and opportunistic in the IoT context. This does mostly amount to a coarse version of decision-based event fusion, and remains far from the computational sophistication of numeric or feature-based sensor fusion.

D. Edge/fog computing

"Edge of cloud" (a.k.a. "fog") computing [6] extends clouds to embedded platforms on the outer edges of the network, corresponding in the IoT to platforms that are closer, not to the end-user, but to the *field*, more precisely the target physical system to be controlled, the "plant", as the automatic-control community would call it. Edge computing brings together close-coupling to the field with much-vaunted benefits of cloud-based data processing and storage, enabling flexible and virtualizable platform support for real-time applications of the IoT.

III. PRINCIPLES OF AN OPEN & SHARED IOT INFRASTRUCTURE

A. Moving from an Application-centric to an Environmentcentric view

Most mainstream IoT/M2M applications are still conceived as the outcome of a top-down and narrowly functional software design paradigm, not as a system or even as part of a system, Such a neatly circumscribed view may be fine for the design of regular stand-alone software applications, but it falls far short of accounting for the complexity of interactions, interdependencies and physical grounding that condition the behavior of Internet of Things applications.

These applications should be envisioned within a multilevel and multi-scale "system of systems" view that encompasses as subsystems the relevant physical entities of the environment in which the application operates. Traditional automatic control specialists used to design their target control systems like this, in association with a model of the plant to be controlled. Powerful as such an approach may be, it cannot realistically be used as the basis for the design and development of most IoT applications, except for one-of-a-kind environments that may warrant such fully customized design such as e.g. factories, airplanes or manufactured products. By contrast, regular IoT applications operate in open environments that not only differ markedly from one instance to another, but do also evolve over time. Most IoT applications in these environments cannot afford custom design for both economical and practical reasons. We have presented in an earlier publication [1] the invariants of such classical IoT environments as homes, buildings or cities that may at the same time justify a platform-based design approach and provide the skeleton upon which this platform can be built:

- Even though these environments differ from one instance to another are different, they are structured by a few broad organizational features that are invariant from one instance to another: all cities are structured by streets, crossings, blocks, all buildings by floors, rooms, corridors, etc.
- They comprise basic physical entities that are also similar from one instance of environment to another: just as homes, cities have their own set of common "appliances" (lampposts, bus-stops, garbage containers, vehicles, etc.),

Using these invariants as a basis, an environment-centric infrastructure may support the design and operation of all IoT applications operating in the same environment, moving away from a situation where these applications rely on vertically integrated solutions that have so far failed to take advantage of even sharing between them the data that originates from this environment.

B. Relevant Data Models in IoT infrastructures

Data mediation in IoT infrastructures should revolve around abstractions of physical entities (subsystems) that can serve as common denominators between all applications operating in the same environment. These should be entities that have a physical location and an extension in the target city environment, seen as both the most invariant features of these environments from one instance to another and the most relevant level at which data can be shared between hitherto verticalized applications in these environments. They can be either material "things", like pieces of furniture or equipment like lampposts or street signs, or relevant subsets of space like streets or parking spaces. They can also be moving entities like vehicles, animals or even humans, the latter being observed as being passively part of the environment rather than active parties, as by definition of IoT and in contrast to non-IoT applications. Borderline as it may be, the example of humans serves to emphasize the role of devices as mere interchangeable intermediaries, bringing connectivity and network presence to entities. On a completely different tack, in environments like cities, factories or buildings where legacy equipment is prevalent, the distinction between such legacy entities (e.g. pieces of street furniture, home appliances, industrial machine-tools, etc.) and network devices or the latest breed of state of the art "connected devices" need not be emphasized. Decentering the IoT from devices is consistent with a broader acceptation [7] that has also been adopted by many leading IoT projects and platforms such as IoT-A and FIWARE, among others.

Similar multilevel and multiscale abstractions have been defined by context management infrastructures [3] as "context entities" a definition which has been adopted by the OMA Next Generation Service Interface (NGSI³) standard in its NGSI-9 and NGSI-10 parts relative to context management [8]. The NGSI-10 context data model has been generalized following

their adoption by the FIWARE project, where the reference to "context" entities should be interpreted in the historical...context outlined above. This data model precisely revolves around entities that are defined by a set of attributes, such as shown in the example in section IV. A proposed evolution of this standard would allow for semantic referencing in the types of these entities and for typing of generic attributes, such as date-time (based on ISO 8601) and location (based on WGS84).

C. Functional Architecture of an IoT platform

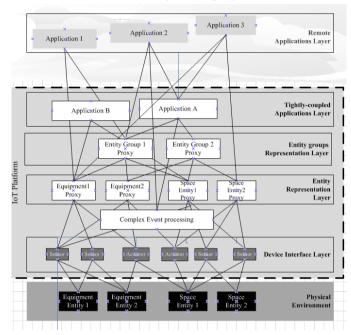


Fig.1 Functional architecture of an IoT mediation Infrastructure

An IoT infrastructure mediates data between the target physical environment and applications, both upwards by fusing /aggregating/abstracting sensor data, and downwards by passing on commands to actuators, through the different layers described in figure 1. To play its proper role as a shared mediation infrastructure, it should rely on abstractions of relevant entities of this environment in the sense described before, making up what we may call an Entity Abstraction Layer (EAL). This may comprise additional levels of abstractions when these entities get regrouped as "virtual entities" according to functional or physical criteria. Beneath this, the Device Abstraction Layer (DAL) provides a uniform interface to networked devices (sensors & actuators) that serve as physical intermediaries to the entities, regardless of protocols and technologies. DAL and EAL each have their own separate interface for a full decoupling of the two.

1) Device Iall its planterface Layer

This layer provides an interface to networked devices that is independent of the protocols used to connect them and of the technology details of their actual implementation, as proposed by various standardization bodies. The mapping between devices and their interfaces is one to one at this level. In the FIWARE infrastructure, the interface to devices uses the same NGSI-10 data model as for "context entities", even though

 $^{^3}$ http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/ngsi-v1-0

context entities are supposed to be higher-level abstractions. This may appear confusing, but this uniformity of interfaces along the abstraction stages is nonetheless useful.

2) Data fusion and aggregation/Sensor Event Processing

This layer takes in raw data from devices and extracts information that may be either input to the entity abstraction layer or used directly by applications. This is done by numeric data fusion/aggregation, or more complex and rule-based event processing that may be either numeric or symbolic. The process takes in data from both a time window of raw data (as regular filtering in the proper digital signal processing sense would do) and a "spatial" window encompassing the outputs of several devices that pertain to the same part of the environment or physical entity. This processing is normally stateless, which means the result of processing incoming data on a given time window is not dependent on the results of processing previous time windows.

The events being output from this layer are represented as attributes of relevant entities that are associated to the corresponding input devices. This makes for a mapping between devices and entities that is both many to one, with a plurality of sensor data outputs being regrouped as attributes of an entity, but also one to many, each sensor output being typically fed into many different entities. The entities used by this layer may be mapped to the entity abstraction layer, if they are generic and permanent enough to warrant this.

3) Entity abstraction layer

The entity abstraction layer provides a one to one mapping between a set of physical entities, chosen as invariant common denominators of the environment, and a set of software representatives (proxies) that maintain their states according to generic system-theoretic discrete-state models that capture the key temporal properties of these entities as they may be used by applications for monitoring but also for control. These entity proxies are meant to serve as a common denominator interface to various actual instances of physical entities sharing a common primary function. These instances may be different from one another apart from sharing this main function, like e.g. different makes or commercial brands of the same category of home appliance as a washing machine. Irrespective of these differences, the interface provided to higher layers of the infrastructure and to applications relies on the same generic entity model.

4) Entity groups abstraction layer

Entity groups are meant to represent a collective behavior or properties of a plurality entity instances that are represented individually in the entity abstraction layer. Thus they involve the composition of several individual entity models. Contrary to proxy instances which are mapped to physical entities in a one-to-one way, entity groups are mapped to corresponding physical entities in a many-to-one or one-to-many way. The entities being grouped may be instances of the same category of entities represented by the same individual model (like e.g. a car park model made up from the composition of models of individual parking places). They may also be groupings according to extrinsic properties such as spatial position. In this case the entities being grouped may be all those that belong to the same subset of space in the current environment like a

room, floor or street, or that are under the same ownership or management responsibility. These heterogeneous entities will not be represented by a common model in the sense of a full-fledged state model but they may still have common attributes that warrant representing them through a common "context entity".

5) Tightly-coupled applications Layer

This layer is not *functionally* part of the IoT infrastructure proper and is mentioned here for reference. Applications are merely the users of the infrastructure through standard interfaces, and the infrastructure should not by definition include any purely application-specific feature if it is meant to be shared. Remaining at the generic level that makes sense for an infrastructure, we may still divide applications in two broad categories depending on whether they are "reactive" or not. Reactive is not synonymous with real-time in the sense where a video streaming application could be considered real-time, it may be defined here in a sense derived from [9] by contrast to both "transformative" and "interactive", corresponding to applications whose behavior is strictly constrained by their coupling to physical systems and which cannot thus be content "best-effort", requiring strict determinacy with boundedness of platform and network response times. They are, to phrase it otherwise, applications that are "defined by their effect on the physical world"[10], and for which time is both an input, an output and a strict correctness criterion, not an ancillary OoS factor These applications need to be co-hosted with the infrastructure, by contrast to "transformative" applications like regular batch-mode big implementation issue that we address in section VI.

IV. COMPLEX EVENT PROCESSING FOR EDGE SENSOR DATA FILTERING

A. Principle

Complex Event Processing (CEP) operates on a partially ordered set of asynchronous atomic events, from which it detects supposedly meaningful complex events. In our application of CEP, input events originate mostly from sensors in the target environment, but may also be complemented by remote online sources for more global events or data.

Raw sensor outputs may correspond directly to asynchronous events (such as a contact detection event from a magnetic contact sensor), in which case they may be fed directly as input to the CEP. Many common sensors produce as output, however, time-continuous analog measurements (such as temperature or luminosity) rather than events. These signals may be transformed into events by single-sensor processing such as thresholding the signal or its derivative. This simple mono-sensor event detection may be performed on the sensor device itself, which makes sense if the sensor operates on a limited energy budget: it would thus transmit sparse events corresponding to supposedly significant changes rather than a continuous flow of possibly unchanging data for transmission to the gateway/edge platform hosting the CEP⁴. A more powerful alternative solution would be to perform multi-sensor

⁴ Local processing on the sensor device does also expend energy, but this is usually much less than the energy required for transmission from a wireless device

data fusion on the joint numeric signals from sensors in different modalities. This is too sensor-specific to be implemented as a generic enabler in the FIWARE platform, but it could be performed directly on a multi-sensor device. The kind of fusion performed by complex event processing on sensor data from different sources is obviously less powerful because it does not apply the same kind of sophisticated processing on the temporally fine-grained data of raw signals. CEP could not match, say, a moving target being tracked separately by a radio sensor and an infrared sensor, as data fusion would do. But CEP is much more generic and much easier to program without specific knowledge about the physical characteristics of the sensor data sources being used, which is why it can play a significant role as a generic enabler on an IoT platform.

CEP operates "along the flow" but normally over a sliding *time window* (temporal scope), which can be either overlapping (as for classical Finite Impulse Response or Infinite Impulse Response filtering of a synchronous digital signal) or non-overlapping, incurring a corresponding delay in both cases. A similar notion of *space window* can be defined for our IoT-oriented application of CEP, corresponding to a spatial scope over which sensor events are analyzed. This scope can be specified in an intensional⁵ rather than extensional⁶ way if position attributes are available for all data sources.

This can be done upon the requirements of a particular application, or as a general-purpose filtering meant to winnow down the streams of raw sensor data to meaningful information, without throwing away any information that might be useful to applications that operate in this environment. This filtering has to take in sufficient breadth on both a time window of sensor data and a spatial window of multiple collocated sensors that may in some way be correlated.

B. Rule Specification

Complex event processing has roots in active databases, which explains why the languages used to specify the rules for programming CEP engines are derived from database query languages. Our CEP enabler is based on the Esper⁷ CEP engine, which uses a Domain Specific Language, EPL (Event Processing Language) for programming of rules. A typical statement looks like the following (brackets indicate optional features):

[INSERT into insert_into_def] SELECT select_list FROM stream_def [AS name] [, stream_def [AS name]] [,...] [WHERE search_conditions] [GROUP BY grouping_expression_list] [HAVING grouping_search_conditions] [OUTPUT output_specification] [ORDER BY order_by_expression_list]

C. Functional Integration in FIWARE Architecture

The CEP engine works by processing incoming events and generating output events, both mapped to updates of Context Entities in the NGSI data model used by FIWARE

When the CEP engine receives an update to a Context Entity using the NGSI 10/updateContext request, it will fire the corresponding events and process the related EPL statements.

If one or more outgoing events are fired by the CEP engine, the corresponding /updateContext requests will be called to notify Context Brokers of the updates to the mapped Context Entities

D. Simple control example with CEP

This extremely basic example uses CEP for controlling shutters from the temperature of a room. We assume temperature sensors are attached to the room entity through corresponding attributes as follows⁸:

The shutters are entities in their own right, associated with the room entity and controlled by setting the appropriate attribute

The EPL rules to trigger the shutter commands based on the previous model could be:

```
INSERT INTO Shutter
SELECT R.r.shutter as id, 'closed' as status
FROM pattern [ every r=Room(temperature > 26.0) ->
(timer:interval(10 sec) and not Room(temperature <
26.0 and id=r.id))] as R unidirectional
LEFT OUTER JOIN
Shutter.std:groupwin(id).std:lastevent() as S
ON R.r.shutter = S.id
WHERE S is null OR S.status = 'opened'</pre>
```

This uses the shutter attribute of the Room as the id of the Shutter. The status is set to closed or opened for Shutter if the pattern is matched. The patterns trigger when the temperature keeps above 26° or under 24° for 10 uninterrupted seconds. The left outer join on the Shutter window allows to only trigger the command if the current status is different. The unidirectional instruction restricts to evaluate the join only when the pattern is matched (not when the Shutter window is updated).

Events detected are not complex in any way, and the control effected is also the simplest possible. We show in the following section this "along the flow" type of event processing and instantaneous control should be complemented by a stronger consolidation of data through state models of entities whenever the action to be effected to attain a desired

⁵ by way of characteristic properties

⁶ by explicitly listing all instances

⁷ http://www.espertech.com/esper/

⁸ We use a JSON serialization of the NGSI data model; more details on http://fiware-cepheus.readthedocs.org/en/latest/examples/2_CloseShutters/index.html

state does actually depend on the previous state of these entities.

V. ENTITY STATE INFORMATION CONSOLIDATION

A. Rationale

A control application need not be much more complex that the previous example for an approach based on instantaneous sensor event detection and stateless setting of actuator values to fail. Let us assume the triggering condition depends not only on the temperature of the room over a given time window, but also on whether the room is empty or occupied. Let us assume further that the only means we have to infer this occupancy state is through a sensor that detects people getting in or out of the room. Obviously we cannot trigger the control directly from sensor events, the "people going in/out" events can be interpreted as relevant state transitions only in relationship to the previous state of the room. Of course anyone could program this in any programming language with a simple counter, but the point we want to make is that this key information should not be just a hidden variable inside some little programming kludge: this entity state is the key abstraction for consolidating information about the room entity, of interest to all applications that may wish to monitor or control it, and it should be exposed as such, as the linchpin of a model of the room upon which upward and downward data mediation will rest.

Staying with the previous example, we could also try to account along the same lines for some physical constraints of a shutter: a shutter is not just a binary variable that can be set instantaneously, it is a physical system that may respond in different ways, depending also on its current state. If we assume it is a roller shutter, we might represent it as having 5 internal states rather than just open or closed, the additional states being "blocked", "moving up", "moving down". Control of the shutter occurs through a "move-up" or "move-down" action sent to its actuator, but the final result and timing of this action will also depend on its current state: if the shutter is blocked, it will not move at all, and if it is currently moving up, it might for some reason have to go all the way up before starting to move down. The abstraction exposed to applications might still be a binary state as in the previous example, the other 3 being hidden states, but the structure and timing of all transitions between the 5 states should still be accounted for in the underlying model. Considering that the result of an action is not instantaneous, applications might wish to get notified when the action is actually completed, or if it failed, which should also be part of the necessary data mediation and consolidation.

B. Principle

In the proposed approach, all self-contained physical entities of the target environment are represented as subsystems by minimal discrete-state models, drawing upon the knowhow of embedded systems to capture their essential timing and reactivity properties. These models represent generic categories to which all kinds of physical entities of the target environment (pieces of equipment or relevant subsets of space) may get approximately matched, capturing a domain knowledge that can be shared by all applications operating in these

environments. We will not focus here on how this identification of such models from entities of the environment, could be made (at least partially) automatic, something which has been the subject of our previous research

State in a system-theoretic summarizes the history of the system, i.e. the behavior of the system up to time t>t0 depends only on its state at time t0 and inputs from t0 to t. In practice we always use a simple discrete abstraction of the actual state of the physical system.

These entities expose northbound interfaces affording shared monitoring and control of corresponding entities from both tightly-coupled and loosely-coupled applications, and the arbitration of potential conflicts between these. The loose coupling interface is fully compliant with the REST architectural style: states of the instantiated models are the resources exposed through this interface to be either read or, if applicable, updated, while transitions provide links to locally attainable states, dispensing with a global functional description of the target entity.

C. Entity models

The entity models we propose to use are derived from Argos automata [11] within the broader family of synchronous languages and formalisms.

These automata have finite sets of explicit states and transitions, with a single initial state. The transitions are labeled by: a triggering condition (also called the input part) and an effect (also called the output part), separated by a slash sign. The input part may be a Boolean formula (in the sequel, j denotes the disjunction); the output part is a set.

Each automaton should be deterministic and reactive, ensuring that, in any state, for any possible input, the reaction of the automaton is uniquely defined; the reaction might be a self-loop with no outputs, meaning that the effect is not observable, in which case the transition can be omitted. Some of the states are timed states, meaning that, once entered, a time-out transition will be automatically triggered after some predefined amount of time; there can be one or several ordinary transitions sourced in a timed state. When defining a language on top of this formal model, it is usual to allow additional variables (typically counters) to be tested and set on transitions. As presented in the followings section, the formalism allows the use of several "parallel" automata instead of a single one, if this is more convenient for modeling purposes.

D. Mapping to REST interface and from stateful entity Models

From an entity model we generate automatically a corresponding REST resource by exploiting the fact that our automata have a particular structure. Our mapping is based on the following rules: (i) an automaton A will be mapped to one resource Ra; (ii) the states \$1; \$2; ::: of A will be mapped to sub-resources of Ra, denoted by Ra=\$1;Ra=\$2; :::; (iii) the GET method applied on Ra returns a link to the sub-resource Ra=\$ corresponding to the current state \$ of A, and three sets of sub-resources describing what actions are possible on A when in state \$; (iv) the POST method can then be used on any

of the sub-resources mentioned in the sets returned by GET, to activate a transition. (v) the outputs of the automaton (the acknowledgements) are returned by a POST call, as part of its response code.

The mapping of the state representation to NGSI data model could be done by assigning corresponding values to a state attribute of the entity. Weak typing of such attributes with strings, in version 1 of the NGSI standard makes this solution inconvenient. It would be much preferable to make use of an extension of the standard, currently under consideration, that would make it possible to represent state transitions as described above, as links from retrieved resources to available resources, and also structured attributes types based on a JSON serialization of this data model.

VI. IMPLEMENTATION ARCHITECTURE OF FIWARE IOT INFRASTRUCTURE

The gap between standalone embedded systems and networked IoT systems should ultimately be bridged, yet it is by now widening rather than closing... On one side are contrivances that are custom designed as informationally closed and physically enclosed systems, with every performance requirement guaranteed by the hardwired design choices being baked into a self-contained one-of-a-kind apparatus. On the other side are IoT systems that are, by definition, distributed, and are moving slowly towards being more open and less vertically integrated. Yet this welcome opening occurs in tandem with a systematic move towards allcloud operations that, beneficial as they may be for big-data types of applications, comes with a performance penalty that is unacceptable for reactive applications as defined previously. These applications require an infrastructure that can provide performance guarantees on a par with traditional embedded systems, while keeping the benefits of openness, genericity, reusability, data sharing and economies of scale that come bundled with open platform-based solutions.

It is the distributed nature of this infrastructure, encompassing close-to-the-field- hardware platforms as well as regular cloud platforms, which should make it possible to support both locally hosted tightly-coupled reactive applications with strong latency constraints, as well as remote loosely-coupled applications. It will also, crucially, provide the benefit mentioned as a requirement at the outset, i.e. the possibility to filter raw data before it swamps networks and storage sites.

A. Backend enablers

Public cloud platforms called FIWARE labs host working instances of FIWARE available for experimentation. FIWARE Labs are hosted on multiple nodes across Europe.

The architecture of the IoT chapter of FIWARE uses a specific registry enabler called IoT Discovery⁹, under which entities in the sense defined above, comprising non-connected things, space entities but also connected devices, may be registered and discovered. The interface used for this is NGSI-9, a part of the NGSI standard complementary to the NGSI-10

part used for entity information representation. This "backend" enabler is typically hosted on a remote FIWARE node. Another backend enabler called IoT Broker¹⁰ has a functionality somewhat similar to the proposed entity and entity group abstraction layers, without the state models and CEP data fusion, and implements them on the cloud without real-time properties. Backend device management and protocol abstraction are handled by another enabler called IDAS¹¹

B. Candidate Edge domain platforms and proposed implementations

Target platforms of this edge domain cover a spectrum from Raspberry Pi boards to high-end servers. Home gateways have been at the forefront of the evolution of such platforms, initiating the opening of hitherto closed boxes, turning them into open platforms that will, ultimately, become the networkedge outposts of cloud platforms. Security and isolation between applications have been first concerns addressed [12], but time-response should be next if the new breed of open gateways are to support time-sensitive applications.

The evolution of "points of presence" [13] supporting network function virtualization could provide a more high-end, less widely distributed and hopefully more secure alternative for the hosting of such platforms close to the field and the physical systems to be monitored and controlled. Ensuring bounded latency and strictly deterministic response times in this case will also require operator-grade last-mile "capillary" networks, rather than the kind of best effort Low Power Wide Area networks being deployed by now on unlicensed bands. Just as for computing platforms, many network technologies may still take time into account only as a QoS rather than a functional requirement [10]

The reference implementation called Cepheus of the Complex Event Processing part of our edge enabler called "IoT Data Edge Consolidation" has been ported on a Raspberry Pi platform. The time performance of this implementation compared to backend implementations of similar functionalities also available in FIWARE is currently being evaluated.

VII. CONCLUSION AND PERSPECTIVE

The "edge" implementation of IoT data mediation that we have described is but a first stage in the direction of a truly distributed cloud, supporting the constraints of time-sensitive reactive applications while keeping all the benefits of data pooling, virtualization and elasticity of a regular cloud implementation. Communication within such a distributed cloud could also be handled through an overlay network that would make it easier to enforce latency and jitter requirements. The next stages of work will require a comparative evaluation of the execution environments such as OSGI which have been used since quite some time for gateway platforms, against virtual appliances or containerization solutions which are currently being used as a lightweight alternative to full-fledged

⁹ http://catalogue.fiware.org/enablers/iot-discovery

¹⁰ http://catalogue.fiware.org/enablers/iot-broker

¹¹ http://catalogue.fiware.org/enablers/backend-device-management-idas

 $^{^{12}\} http://catalogue.fiware.org/enablers/iot-data-edge-consolidation-ge-cepheus$

virtualization, and could make it possible to come closer to the transparent distribution of a full IoT infrastructure.

ACKNOWLEDGMENTS

Part of the work on which this article is based has been carried out within the IoT Chapter (WorkPackage1.4) of the EU FIWARE project, with key contributions from all the following people on the overall architecture and especially from Carlos Ralli-Ucendo from Telefonica I&D for the IDAS Device Management Generic Enabler, Dr. Tobias Jacobs, Stefan Gessler and Dr. Ernö Kovacs from NECLab Europe for the IoTBroker Generic Enabler, Tarek Elsaleh and Maria Bermudez from the University of Surrey for the IoTDiscovery (registry) Generic Enabler. The development of the Cepheus (called at this time Espr4Fastdata) Complex Event Processing enabler was initiated in the first phase of the FIWARE project by Laurent Artusio, under the leadership of Thierry Nagellen for the IoT chapter. We are most grateful to Stephane Vialle for supporting the participation of his team to the project and to Christophe Azemar for his contributions.

The ideas on identification of generic system-theoretic discrete-state and discrete-time models for physical entities build upon the PhD research of Zheng Hu and Mengxuan Zhao and have evolved from there by drawing upon the invaluable embedded-systems experience of Pr. Florence Maraninchi (Grenoble-IPNP/Verimag) and the insights of Pr. Didier Donsez (Université Grenoble Alpes/LIG) on distributed IoT middleware.

Thanks to Dr. François-Gaël Ottogalli for his review of the paper and his insights on fog computing and to anonymous reviewers for their constructive remarks.

REFERENCES

- [1] G. Privat, M. Zhao, and L. Lemke, "Towards a Shared Software Infrastructure for Smart Homes, Smart Buildings and Smart Cities", International Workshop on Emerging Trends in the Engineering of Cyber-Physical Systems, Berlin, April 14, 2014
- [2] G. Privat, N. Streitz, "Ambient Intelligence", Chapter 60 of Universal access Handbook, Constantine Stephanidis, Editor, CRC Press, 2009 http://www.crcpress.com/product/isbn/9780805862805
- [3] Ramparany, F., Poortinga, R., Stikic, M., Schmalenstroer, J., & Prante, T. (2007). An open context information management infrastructure-the IST-Amigo project. *Intelligent Environments* 2007
- [4] H. Rizvi "Fast Data gets a Jump on Big Data", Forbes, March 2013 http://www.forbes.com/sites/oracle/2013/03/01/fast-data-gets-a-jump-on-big-data/
- [5] Hall, D. L., & Llinas, J. (1997). An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1), 6-23.
- [6] F Bonomi, R Milito, J Zhu and S Addepalli, "Fog computing and its role in the internet of things", ACM MCC workshop on Mobile cloud computing., 2012. p. 13-16
- [7] G. Privat, "Extending the Internet of Things", Communications & Strategies, Digiworld Economic Journal, vol 87, 2012, pp101-119
- [8] Bauer, M., Kovacs, E., Schülke, A., Ito, N., Criminisi, C., Goix, L. W., & Valla, M. (2010, October). The context API in the OMA next generation service interface. In *Intelligence in Next Generation Networks (ICIN)*, 2010 14th International Conference on (pp. 1-5).
- [9] Halbwachs, N. Synchronous programming of reactive systems. Springer Science & Business Media, 2013.
- [10] Lee, E. A. (2009). Computing needs time. *Communications of the ACM*, 52(5), 70-79.
- [11] F. Maraninchi and Y. Rémond, "Argos: an automaton-based synchronous language," *Computer Languages*, no. 27, pp. 61–92, 2001.
- [12] K. Attouchi, Gael Thomas, André Bottaro, Gilles Muller. Memory Monitoring in a Multi-tenant OSGi Execution Environment. CBSE '14 -17th international ACM Sigsoft symposium on Component-based software engineering, Jun 2014, Marcq-en-Baroeul, France.
- [13] Gosselin, S., Mamouni, T., Bertin, P., Torrijos, J., Breuer, D., Weis, E., & Point, J. C. (2013, July). Converged fixed and mobile broadband networks based on next generation point of presence. In *Future Network* and Mobile Summit (FutureNetworkSummit), 2013 (pp. 1-9). IEEE