Bringing Context to the Internet of Things

Dr. Emmanuel Frécon

This paper is dedicated to my newly become wife

Abstract—**The context manager is aimed at being the hub of the house, a place where all sensors report (directly or indirectly) their data, sometimes in aggregated form, but also where all applications will search for information relevant to them, i.e. sensor values, location or information about their surroundings. The context is instantiated from a dynamic model to fit the needs of a variety of scenarios and settings. The manager provides an easy-to-use Web API and integrates external cloud services relevant for applications running in the house.**

***Index Terms*—IoT, Tcl, REST, JSON, Web, Integration, Web Services, Sensors, Actuators, Middleware, Energy, Smart Homes**

I. Introduction

THE Internet of Things promises a near future where domestic and work environments, but also cities and factories, are augmented with sensors and actuators that all are Internet entities. The deployment of IPv6 is key to this evolution by enabling each sensor or actuator to be accessed from any Internet enabled application or user, thus from almost anywhere. The Internet of Things is often seen as the catalyst of more intelligent environments, where applications will use Things to perform actions and sense on our behalf, with little human intervention.

As the number of connected Things will grow, making sense of what is accessible and can be done and how they relate to one another will be harder and harder. For taking good and qualified decisions, applications will not only need to know how to access the sensors and actuators, but also where these are located, the people in their vicinity, their immediate neighbouring Things, etc. The context manager is a modular Tcl[1] web service that attempts to provide this contextual information to applications, i.e. the extra logical layer empowering applications with the overlaying of dynamic sensor data on top of locational data of more static nature. While the context manager primarily targets home environments, it can also be used in other environments. The context manager relies on simple PubSub[2] and pull mechanisms to account for the low resources available on sensors. It also offers a streaming interface based on WebSockets for push and pull of sensor and actuator data.

Emmanuel Frécon is with the Interactive Collaborative Environments Laboratory, Swedish Institute of Computer Science, Box 1263, SE-16429 Kista, Sweden, e-mail: emmanuel@sics.se.

II. Related Work

There starts to exist a number of cloud-based services that target the IoT (Internet of Things) and provide APIs to store and later retrieve data that has been sent for storage into the cloud. Probably the most well-known of these services is COSM[1] (formerly known as pachube). But there are a number of other services such as sen.se[2], nimbits[3] or thingspeak[4] to mention a few. Common to all those services is a web-based API that is easily integrated directly from sensor platforms, providing tiny (connected) sensors or gateways off-site storage. The same API can be used to retrieve data from

[1]Cosm is available at https://www.cosm.com/ and is open for new account registration.

[2]Sen.se is available at http://open.sen.se/ and is open for beta testing by the way of invitations only.

[3]Nimbits is available at http://www.nimbits.com/ and is open for new account registration. Nimbits also touts private cloud solutions by allowing the integration of the nimbits solution within existing architectures.

[4]Thingspeak is available at https://www.thingspeak.com/ and is open for new account registration

the web services, thus opening up for data-mining activities if ever necessary. Common to those APIs is the use of REST[3] and JSON[4] for retrieving and posting data from and to the cloud. This is to ease integration from low-power consumer-oriented hardware platforms such as arduino[5] or gadgeteer[6].

In addition to providing an ”infinite” data storage for sensors and actuators, the power of these services lies in the community of people around them and the ability to integrate values from several sources to reason in improved and more sensible ways. In short, these services provide a whole ecosystem of devices, applications and people, by being able to pinpoint sensors using location services and providing ways to get notified whenever their value change. From the point of view of a house and household, sen.se seeks to take a step further by allowing users to build user interfaces to control their houses. Sen.se provides ways for its users to visually create applications by connecting sensors to web-side logic boxes and finally present the resulting “computation” through its dashboard, a web-based UI combining output of values with input of commands to be utterly received by actuators.

However, these services fail to provide more information about the context within which all these sensors and actuators are being placed, especially when it comes to smaller scale installations such as a house or a building. In order to be able to make energy-smart decisions, leading to smart actuation of the devices that are accessible to them, applications need to know about inhabitants, relative locations of sensors, external conditions, etc. So far, the merging of the Semantic Web[7] with sensor networks, also known as the Sensor Web or the Sensor Internet [8][9][10][11] has focused on the creation of specifications for different functionalities related to the management of sensor-based data (observations, measurements, sensor network descriptions, transducers, data streaming, etc.), and for the different types of services that may handle these data sources (planning, alert, observation and measurement collection and management, etc.). The cost of providing network abstraction and ontologies often comes with increased complexity. So while these middlewares effectively provides ways to reason about devices and actuators at a high level, they seldom solve the problem of providing the general context while still lowering the threshold for regular users.

III. Goals

The context manager is aimed at being the nav of the house, i.e. the place where all relevant sensors report (directly or indirectly) their data, sometimes in aggregated form, but also where all applications will dig for information that is relevant to them, i.e. both values from some sensors, but also their location or information about their surroundings. Being such a nav, the context manager is designed to be placed and hosted in a home gateway, i.e. a “number crunching” appliance that provides computing power and intelligence at a lower price in a central place[5]. In this context, houses are taken in their larger forms and can be entire buildings if necessary, and the design should open up for federations of context managers to adapt to the needs and privacy concerns of both building owners and flat owners (or inhabitants).

The main goal of the context manager is to provide dynamic ways to model the context, e.g. a house and all its online devices, be them sensors or actuators. The dynamism of the context is essential at different levels: first it is important to be able to host new devices as they are installed in the house, secondly it is important to be able to model the context in various ways because all houses are far from being the same and because there might be cultural differences between location that have an impact on the context itself. Consequently, the context manager takes an object-oriented approach, where the possible content of the context, i.e. the objects themselves is driven and controlled by a simple schema, i.e. a model of what objects can be made available in the context, but also a model

[5]The current implementation of the context manager has been verified to be fully functional on the open source BeagleBoard xM, an ARM A8 development board.

of their relations. The use of schemas could introduce complexity to the conceptual approach, so the context manager features a simplified schema with few rules and in a human-readable format. Several schemas can be aggregated, allowing for experts to provide base schemas, perhaps somewhat more complex in their form while still providing power to the end users and the inhabitants, so as to adapt to the specific needs of a household, a building or a custom-made online sensor.

The context manager seeks to provide an open API that follows the current trends within Web-based services and development. Web asynchronous communication is slowly moving from SOAP[12] and XML[13] standards into REST and JSON for a number of reasons. One of the advantages of these new standards are their ease of reading, i.e. core communication can be tested directly in the web browser, and the results from a query are easily read in a textual format that is much more compact than XML. It is out of the scope of this document to advocate for either one or the other standard, but since the context manager aims at providing an easy interface to application programmers, REST/JSON are more suitable to the task. Apart from allowing programmers to test queries against an existing instance, both the format of the queries and of the result are in general less cumbersome to parse, thus easily integrated into existing code and onto low-power platforms such as mobile phones or even sensor platforms.

IV. Design

A. Schema and Model

In order to cope with different sorts of environments and to account for the cultural differences between housings in various regions of the world, the context manager is based on a dynamic schema that directs the content of the objects that will be instantiated to describe a house or any other environment. The schema can include remote (web) schemas, thus providing ways for experts and/or interested users to collaborate, but also providing for the inclusion of new classes of objects that will support newly created sensors. In order to easily be accessible to technically inclined users, the schema supports few paradigms: single inheritance, a few base types (`Boolean`, `Integer`, `Float`, `String`, `Timestamp`[14] and arrays) and constraints. Constraints describe rules to which field values should comply to, providing minimum and maximum bounds or constraining only a few possible values. Constraints offer a way to model physical units and laws: for example, temperature can be expressed in Celsius and is always greater than -274.15. The syntax for the schema minimises idioms and is designed to be human-readable with lesser effort.

Objects modeling the context are instantiated from the schema, and sensors and/or external services will update the fields of these objects as new values are measured, made available or acquired. Initial instantiation will provide decent default values for all fields and subsequent updates will always be checked against the possible constraints that direct the content of one or several fields. The context manager provides a number of techniques for remote services to be notified when objects and their contained fields are modified as time passes.

B. Data Flow and Storage

The context manager reads its schema (and included schemas) during initialisation, subsequently reading a file describing the initial context. Typically, the initial context will be composed of more or less transient information such as the rooms composing a house together with their interconnections, but also an initial instantiation of the objects that will be involved in the dynamic representation of devices, sensors, actuators and inhabitants, together with their spatial relations. Modification of fields in objects, and queries for the inter-relationships is supported by a REST/JSON inspired API.

The API supports (basic) authentication and HTTPS[15] encryption if necessary, because of their widespread use and their ability to scale down to the few resources available on sensors. As times goes by, all values set are automatically mirrored to a noSQL database (cluster) implemented on top of REDIS[16]. The API supports access to historical data. It also enables setting values in the future, thus supporting prediction. Whenever the scheduled time is reached, a value will automatically be set to the one that had been set in the future. The API also permits setting values in the past for the automated storage of historical data. So-called triggers implement a PubSub mechanism, allowing remote Web services, applications and sensors to be notified whenever value(s) in objects change upon given conditions. Finally, WebSockets[17] can be kept opened against particular objects, offering both a way to stream updates from the object as time passes, but also to update its fields whenever needed.

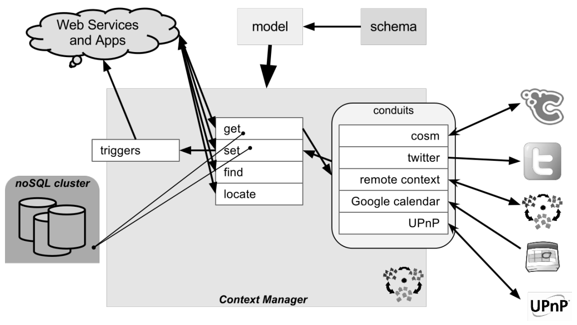

Figure 1. The API of the context manager provides REST/JSON entry points both to query the state of the context, but also to modify it. In addition, it supports external known and generic Web Services, while being able to predict and automatically store historical data through a connected noSQL database.

C. *Extensibility through Conduits*

The context manager is extensible through the concept of “conduits”. Conduits are logical entities connected to external web services that will direct data into or from the context depending on a number of conditions. Typically, conduits will perform some transformation on the data to or from the external web service, while also retaining data that is specific to the remote service, e.g. login credentials, authorisation details, session information, etc. At present, there are conduits for Twitter, for import and export of data to the COSM cloud service, to remote context managers, to nearby UPnP objects and services and for the control of objects’ values according to Google calendar bookings. Conduits are loaded as a set of plugins during the initialisation phase, they access the context manager through its modular internal API.

V. IMPLEMENTATION

A. *Functionality*

The context manager roughly provides the following set of functionality:

- It takes a schema and a model to provide a logical context of a building. This context can be accessed and modified using REST/JSON calls for maximised flexibility and integration. This means that most operations to and from the context can actually be made (tested?) from the comfort of any Web browser[6].
- The context manager provides a number of ground operations to:
  - Get the content of whole or part of the context, including the values of the fields of the instantiated objects and including values from the past, whenever they are accessible.
  - Modify values of objects that already are instantiated, which will be an operation that is often used when the value of a sensor changes.
  - Provides means to search for objects by the content of their field, the name of their class, etc.
  - Provides means to understand arrays as a technique to organise (part of) the model in a hierarchy and to find specific objects within such a hierarchy.
  - Provides means to trigger external web services whenever (part of) an object has changed, i.e. to mediate the content of the object to remote Web Services. Triggers offer enough flexibility so as to be able to:

[6]There are several JSON formatting extensions for most of the Web browsers. Such an extension will be necessary since the context manager minimises output by removing all unnecessary indentation or line breaks.

    * Restrict which field of the object are under watch and how to mediate their value to the remote service, i.e. as part of the URL, in the body of the posted data, etc.
    * Specify in details the headers, the MIME type and the method of the HTTP request (GET, POST, PUT, DELETE).
    * Control the maximum frequency of this mediation to avoid flooding the network.
    * Mediate only under certain conditions, expressed as a mathematical expression involving any of the fields of the object (based on the Tcl `expr` command).
    * Control if mediation should happen every time the value is updated, or only if it has changed since last time (the default).
  - Provides means to stream the flow of changes to remote clients via WebSockets, expressed as JSON representations of the object. This interface provides approximately the same level of control as the triggers described above[7]. WebSockets have two advantages:
    * They easily pass through firewalls and multi-level NAT hierarchies, thus making sure that even clients at the edges of the network can be notified of changes.
    * Once the connection has been established, packets containing object data are kept to a minimal (with almost no additional overhead or verbose header), which makes them suitable for transmission across WSN (wireless sensor networks).
  - Automatically saves versions of objects to a database for later retrieval. Given the unstructured nature of the context, noSQL databases are a perfect match, especially since they form the base of a number of data-mining techniques.
- The context manager offers a pluggable architecture through the concept of "conduits", i.e. logical entities connected to external web services that will direct data to or from the model depending on a number of conditions. Typically, conduits will perform some transformation on the data to or from the external web service, while also retaining data that is specific to the remote service, e.g. login credentials, authorisation details, session information, etc. There are a number of conduits already available:
  - The COSM conduit is able to pull and push data from the COSM Cloud service. The conduit supports both the access of non-public feeds via an API key and to public feeds, polling their content at the necessary frequency whenever needed. Data can be transformed on the way to and from the feeds, matching *feed* names against *fields* names in the context manager, but also performing any mathematical operation supported by the `expr` command at copy time.
  - The remote context conduit is able to pull and push data from remote context managers, using triggers at the remote managers to get notified of changes. The conduit can be forced to poll for data instead to ease firewall and NAT traversal. As for the COSM conduit, data can be transformed on the way to and from the remote context. Being able to incorporate (parts of) remote context into a local context opens up for the creation of federations of context, and the ability to (re)use the sensors of your neighbours when taking decisions.
  - The local context conduit is a simplification of the remote conduit that only acts between local objects. It allows for the transfer (and transformation) of fields

[7]All control that is only relevant to how the HTTP request to the external Web service should be made are left aside since this is not relevant in the case of WebSockets, i.e. in a framework that keeps the connection opened at all time.

values between different objects of the context, whenever some conditions are met.

- The Google calendar conduit binds the events of a given calendar to a `Boolean` that will turn `on` when there is a booking in the calendar, and `off` when there is no booking. Combined to local conduits and actuation (see section VII-B2), this can be used to specify when some devices should be turned on or off.
- The UPnP[18] conduit is able to pull and push data from remote UPnP services. The conduit is built on top of an SSDP discovery mechanism, thus being able to bind objects of the context manager to a service that has a given (discovered) name, or to a service that is at a given known location. While the conduit has been designed to bridge the context manager to objects within the LinkSmart middleware[11], it makes a number of assumptions to be able to be used in more generic cases. Similarly to the other conduits, the UPnP conduit is able to push and pull data to and from the known state variable of a UPnP server. For this to work in a generic way, the conduit assumes that the service has methods which name contains the name of the state variable and that contain the keywords "get" or "set" to get or set the content of the variable.

B. Security Mechanisms

There are two intertwined security mechanisms that will control the access to the context manager. First of all, the context manager is able to run on top of HTTPS[15] thus providing encryption of both requests and their results, so as to avoid eavesdropping from external parties. HTTPS was chosen because it is a well-established protocol that is widely supported across languages and platforms. The context manager supports both self-signed and authorised certificates.

Secondly, all web accesses can be controlled by a user name and password that will be mediated to the context manager using Basic Authentication[19]. Control should occur at the (virtual) directory level so as to provide for finer grained access restrictions if necessary. The goal is to refrain some users from, for example, setting the values of some objects of the context. Again, basic authentication was chosen because it is widely supported across languages and platforms. Basic Authentication sends the password from the client to the server unencrypted, however it should be used in conjunction with HTTPS.

There might be cases where HTTPS encryption is too heavy for the client platform in terms of computing resources, for example if sensors send directly their data to the context and/or need to reason about other sensors in their vicinity to take decisions. For those cases, the context manager is able to provide regular HTTP access. This HTTP access should be secured by a set of firewalling rules that will prevent access to the context manager from any remote client except the ones that need to access the manager for the reasons detailed above. Since these cases are most likely to occur within home networks and since most current home installations and Internet accesses are based on NAT techniques, the security risks introduced by unencrypted access in those cases are deemed to be low. In those cases, wires or proximity ensures physical security. This security relies however on proper configuration of the Internet access and the different firewalls involved.

C. Startup and Initialisation

On startup, the context manager will perform the following operations in sequence:

1) The context manager will start a web server with the proper credentials (see V-B) and proper encryption settings. Alternatively, the context manager can be embedded in an existing server framework if more suitable.
2) The web server will expose the schema and model that will define the context of the building or the house that the manager is controlling and modeling.
3) It will read the schema (see VI-A) that will describe what classes of objects are allowed

to appear in the context. This includes possible access to remote schemas that might be included from the main schema. Reading of the main schema might be through accessing the internal web service if necessary[8].

4) It will then read the model (see VI-B) that describes the particular building that it is modeling and controlling. All constraints implied by the schema that has just been read will be applied as the model is being read.
5) All objects instantiated as part of the model are bound to the noSQL engine so that further write operations will automatically lead to new versions of the object being stored and so that later get operations will be able to get older data, whenever possible.
6) It will initialise all conduits that are accessible to this context manager. Conduits are conceptually separated from the remaining of the code and are plugins communicating with the remaining of the context manager through a tiny and well-defined (internal) API.
7) It will read an initial "pairing" state (see VI-C) that is used to initialise a number of conduits and to bind a number of objects to remote services. Pairing is explained later and mostly a helper functionality that aims at reinitialising the context manager every time that it starts and reaching a similar functioning state.

VI. File Interfaces

Instead of providing an entire specification of the file formats that are understood by the context manager, this section focuses on providing real-life (shortened) examples. These examples are annotated and explained, bringing further insights to the internal of the context manager and all the facilities that it offers.

[8]Actually, reading the main schema via the web is encouraged since this will enforce UUIDs that remain constant over time and are bound to the specific installation. Preferably, a hostname will be involved in the main URL to bind the instantiated objects, classes and their UUIDs to a specific and logical place.

A. Schema

As highlighted before, the context manager provides techniques to specify the schema that will be used to describe the context itself. A key requirement to the provision of this schema is that it should be easily approachable not only by IT specialists, but also by less-knowledgeable people. To this end the schema brings in object-orientation concepts but simplifies them to their outermost. For example, it provides simple inheritance and mixes both object field specifications and inheritance[9]. The schema does not provide concepts such as private variables or similar, once again for the sake of simplification.

Below is a cut-down example of a schema, providing a flavour of how a schema looks and feels like. Roughly, this example schema divides the space into a number of possible floors and rooms within a building, and enables each part of the space to carry a number of devices (inhabitants are left aside on purpose). The example sports a single type of device, namely a thermometer, which demonstrates the (definition and) use of constraints to provide for a richer expression of units and properties of the physical world. The constraint defines temperature (in Celsius) as a floating point value that always is above the 0K.

```
Space {
  name String
  contains Space[]
  devices Device[]
  Outside {
  }
  Building {
     address Address
     pos Coordinate
  }
  Apartment {
     number Integer
  }
  Floor {
     above Floor
     below Floor
  }
  Room {
```

[9]While mixing class hierarchy and description in the same flow might surprise, this solution was chosen for the sake of simplicity. It has the advantage of presenting all data relevant to a given schema at a glance.

```
        Kitchen {
        }
        Bedroom {
        }
        Office {
        }
        Bathroom {
        }
    }
  }
  Address {
    street String
    streetNumber Integer
    areaCode Integer
    city String
    country String
  }
  Coordinate {
    latitude Float
    longitude Float
  }
  Temperature:Float {
    intervals {[-273.15,[}
    unit "celsius"
  }
  Device {
    name String
    SensorDevice {
        Weather {
            Thermometer {

              value Temperature
            }
        }
    }
  }
```

To simplify the approach by non-technical experts, no forward declaration of classes or constraints is necessary. All new "types" that are discovered will be understood as (empty) classes as a start and converted when their real definition occurs. While this has the drawback of more complicated parsing and the possibility of duplicates or of unknown state — what to do when a class with a given name is then specified as a constraint under the same name — these problems are considered minor compared to the necessity to forward declare classes or constraints before being able to use them.

B. Model

The schema only specifies and constraints the types of the objects that should be placed in a model. While the schema is essential to the context manager since it provides guidelines to what can be instantiated within the model, achieving a conceptual model of a home and all its online devices is the ultimate goal of the context manager. To this end, the context manager provides a file format that is easily approachable, allowing people to quickly model their own house. At later stages, and depending on the success of the approach, graphical tools would certainly provide help in specifying the final model, perhaps based on existing drawings (blueprints or CAD).

Below is an extract of a model, based on the example schema above. The purpose of this example is to set the scene and provide a flavour for how model files could look like. Complete models tend to be more extensive, so the example below is not complete.

```
Outside pHataren1 {
  name "PositivHataren1"
  contains {myHouse}
  devices {
     outsideTemp
  }
}
Address aSoderman10 {
  street "August Södermansväg"
  streetNumber 10
  areaCode 12938
  city "Hägersten"
  country "Sweden"
}
# Approximate center of our lot.
Coordinate myPosition {
  latitude 59.299428
  longitude 17.970209
}
# The house contains three floors,
# which will contain the rooms.
Building myHouse {
  name "House Frecon-Waller"
  address aSoderman10
  pos myPosition
  contains {
     ground cellar top
  }
}
```

```
# The different floors in the house,
# here only one for the sake of
# concision.
Floor ground {
  name "Ground Floor"
  contains {
     hall kitchen diningRoom
     livingRoom bath vilma
  }
  above cellar
  below top
}
#######
# Devices
Thermometer outsideTemp {
  name "Temp. sensor outside"
}
```

The model uses the schema to control the content of objects that are created within the model. Every instance of a class is referenced using an identifier. Using techniques similar to those used for the schema, objects can be referenced before they are actually used, but the model provides enough feedback whenever the data that is specified does not correspond to the schema that controls what can be specified.

In the resulting model, both instantiated objects within the model and classes are identified by a UUID[20]. The UUID is of type 3 or 5 and built using a concatenation of the URL to the model (or to the schema), the class name and (when relevant) the reference to the object. This ensures that, even upon restarts, objects and classes will keep their UUIDs as long as the file structure, content and location has not changed.

C. Pairing

In order to be able to restart from a similar state at all times, the context manager is able to read from a pairing configuration file once the schema and the model have been read. The purpose of this file is to establish all the necessary conduit connections to well-known services. Pairing is made at the conduit level, thus at the REST/JSON level. In other words, when initialising the pairing, the context manager behaves as if it was an external client to itself. This is to be able to support new conduits in the future and to fail nicely if some conduit initialisation did not succeed properly.

Below is an annotated example file showing how pairing can be initialised at start, the syntax provides some visual markup to highlight the source and destination objects and uses a number of heuristics to detect which conduit to use for data migration. An integer is understood as a COSM feed, a UUID as an object from the local context manager, a URL ending with a UUID as an object in a remote context manager, a URN starting with `gcal:` as a Google calendar and a URN starting with `UPnP:` as a UPnP service. The "arrow" of the markup can specify and/or force polling frequency and indentation is used to further specify how the value of fields are carried to the remote entity.

```
# Map my heat pump to the COSM feed with
# identifier 53880. Non-matching
# fields/datastream names will be ignored.
# API key is picked up from the configuration
# of the context manager.
55851044-b290-56a5-3c88-d64ffbfa75e9 -> 53880
# Another COSM mapping, making sure the COSM
# datastream "inside" is mapped to 4 times the
# value of the field "value" in the context
# object,
20ecdbe4-8459-5636-6146-71c618badc71 -> 53882
  %inside% = 4.0*%value%
# Reverse COSM mapping, datastream called "2"
# in feed 55180 at COSM is brought the field
# "value" in the context object.
55180 --> 929494fb-84e1-50cb-beea-c04aecda088a
  %value% = %2%
# Pick up the weather station of somebody else,
# do some field names / datastream mappings and
# force polling to occur every 180 seconds.
45036 -180-> 684c4e19-c4ed-5861-f127-59109a41bb56

  %temperature% = %OutsideTemprature%
  %pressure% = %ABSPressure%
  %humidity% = %OutsideHumidity%
  %rain% = %Rain%
  %windDirection% = %WindDir%
  %windSpeed% = %WindSpeed%
# Send status of context object to the UPnP
# service named "Dev"
5d9a66e5-9738-598c-d0b0-e707eb0e2a36 -> UPnP:Dev
```

VII. Applications and Examples

This work has been carried out within the framework of a European project looking into energy optimisation. The project uses traditional home automation in order to attain energy savings while still offering the same level of comfort. The project also seeks to offer “soft” actuation mechanisms, i.e. providing enough (summarised) information about some of the decisions taken by devices in the home to let inhabitants take the final necessary steps. Ambient displays are used to carry out this type of information in a form that is aesthetically acceptable. A number of prototype pilot houses have been equipped with sensors and actuators of various forms in order to gather data for future data-mining activities, but also to experiment with how smart actuation can turn into energy savings or reduction of CO_2 emissions. This section describes one of these prototype installations, located in the outskirts of Stockholm. There are several other installations, featuring a slightly different feature set of software and hardware so as to adapt to the particularities of these households: type of heating, electricity meter, etc.

A. Heating and (Inner) Climate

1) Heat Pump Analysis (and Control): Live status from a heat pump (IVT Greenline HT+) is picked up via its service serial interface using software from a small Swedish company called Husdata[10]. This is connected to a PC running Windows sitting on top of the pipe system in the direct vicinity of the heat pump. The default settings within StatLink, the software provided by Husdata as part of their offering have been slightly modify to increase the number of sensors being read and to regularly dump sensor data to a particular location on the PC, a location that is served by a tiny web server[11]. Raw dump data is (remotely) polled by a Tcl script at regular intervals and pushed to the context manager after naming transformations. Within the context manager, a conduit forwards acquired data to two COSM feeds[12].

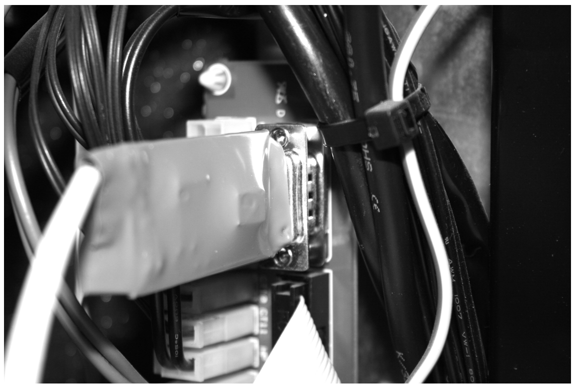
Figure 2. Husdata provides a hardware module to connect to the service serial interface of the heat pump, together with Windows software for analysing the decisions taken by the pump over time.

Figure 3. A Windows PC collects data from the heat pump, it is placed on top of a number of copper pipes in direct connection to the pump. The module with an antenna is the root node of the WSN network that collects temperature and main electricity data (see sections VII-A2 and VII-B1).

The current solution only provides gathering of heat pump sensor data. This has been immensely valuable since we can now perform long-term analysis of the behaviour of the heat pump using data-mining techniques, so as to be able to detect with

[10]Husdata http://www.husdata.se/ offers a number of hardware modules to connect a computer to a range of heat pump commonly found in Sweden, from several manufacturers.

[11]The current installation relies on mongoose, available at https://github.com/valenok/mongoose.

[12]Converted pump data is pushed to https://cosm.com/feeds/53880, temperature is pushed to one of the datastreams of https://cosm.com/feeds/53882, additionally raw data, as taken directly from the husdata software is pushed to https://cosm.com/feeds/52002. This is being used mainly for debugging purposes and for detecting possible failures in the context manager and in the surveillance PC network connection.

this given household will need warm water. It opens up for predicting when it will use warm water in the future, so as to shutdown warm water production during peak hours and start again at off-hours, before warm water is needed again. However, taking these last steps implies being able to control the internal logic of the heat pump, using the serial protocol described at http://rago600.sourceforge.net/.

2) Inner Temperature: A TinyNode[21] board running Contiki[22] and carrying a temperature sensor is hidden behind a photo frame. Measurements are sent along a mesh network at regular intervals, captured via the pump computer and sent on via UDP to a Tcl script. The script automatically pushes data into an object of the context manager, and further to COSM[13] via a conduit.

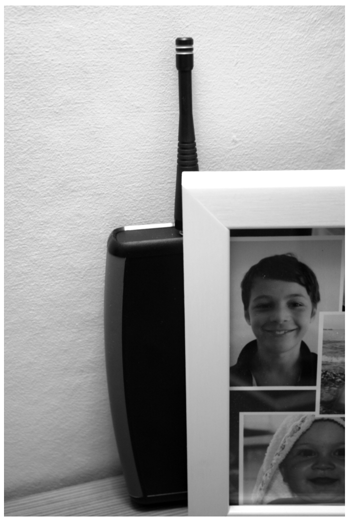

Figure 4. The TinyNode measuring temperature hides itself behind a photo frame in the living room, so as to break the aesthetics as little as possible. It has been slightly pushed aside for the sake of documentation and picture taking.

This particular heat pump installation only contains an outside temperature sensor, with which all decisions are taken when it comes to heating. The pump is able to host an inner temperature sensor (cabled) to take better decisions about when and when not to generate heat. Combined with the planned implementation of serial connection and control of the pump, the provision of a wireless inner sensor could provide for better inner climate without the wiring that is required by regular installations.

[13]Temperature is pushed to one of the feeds https://cosm.com/feeds/53882 that already is used to publish the outside temperature acquired via the heat pump, though to another datastream.

3) Weather: A specially written Tcl script can be used to update (in the future) an object of the context manager to reflect the weather forecast for a given location. The script uses the REST/JSON API from the Weather Underground[14] to access an hourly forecast for the coming 10 days. As updates are made in the future for those specific times, they are automatically stored in the noSQL cluster and set back as the "current" value as time passes. The script can be run once in a while or continuously. It will thus keep updating the object with an up-to-date weather forecast, allowing other applications using the context manager to reason about the current and future weather situation. For example, an application that would control heating could make the decision to accept temporary temperature drops if the outside temperature is only going to decrease for a few hours/days. The object that the script sends its data share the same model as a weather station, thus implementing a virtual private weather station in combination with the script.

B. Electricity and Energy Consumption

1) Total Measurement: The past decade has seen the progressive replacement of all electricity meters in Sweden in favours of so-called smart meters. These meters are able to report the hourly electricity consumption to the utility company as time passes, so as to bed for refined billing and better dimensioning of the grid. Pulses from the electricity meter are captured by another TinyNode sensor running Contiki, manufactured by CRL Sweden. Data is pushed out of the sensor network to the same Tcl

[14]Documentation for the API is available at http://www.wunderground.com/weather/api/. There are numerous other services offering the same type of data, Weather Underground was chosen because of its ability to chunk several questions into one request, but also because it is also uses ideas from the Internet of Things: forecast are improved using the data from private weather stations, whenever possible.

bridge as in section VII-A2. The bridge forwards to another object of the context manager, and thus automatically to COSM[15]. This publishes an history of the instantaneous power used by the household over time. As electricity is one of these hidden cost that is seldom understood, an ambient display is at the planning stage; a display that will both visualise how much electricity has been used so far, but also provides feedback to the instantaneous variations, thus to power, required by new devices being switched on (or off).

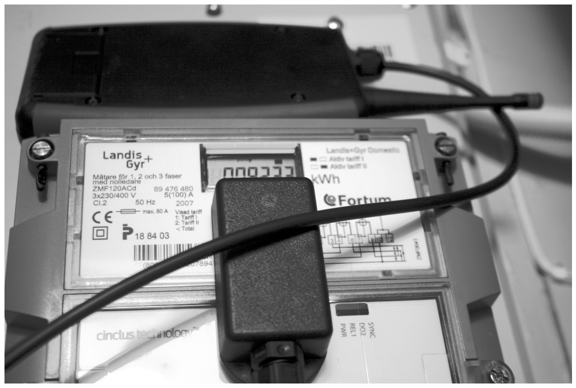

Figure 5. All electricity meters installed in Sweden host a LED (IR or visible) that flashes a number of time for each number of Watts used. The pulse metering node, sitting on top of the meter itself, continuously count these pulses and reports the total count for the latest period via the WSN network.

2) Measurement and Control at the Device Level: PlugWise are smart plugs manufactured by a Dutch company. They form a ZigBee mesh radio network, allowing access and control from a computer to which a specific USB key is connected. They offer three key features.

1) They host a relay, meaning that they are able to turn on or off all the devices connected to the plug and this from a distance. The state of the physical relay can be queried at any time.
2) They measure the instantaneous power being used by all devices connected to the plug, a value that can be requested from a distance.
3) They keep an hourly log of the electricity consumption related to the plug. As this log is kept in memory in the plug itself, historical data for the plug can be accessed from a distance at later time if necessary. The log rotates with time but keeps a few days worth of hourly data.

A Tcl script couples one or several objects from the context manager to as many smart plugs as there are objects. At the core of the Tcl bridge is a wrapper library around the command line interface of one[23] of the open source libraries created to access the PlugWise hardware and network. The bridging script connects to object representations of the plugs in the context manager using the WebSocket API[16] and pushes all information, including relay state and historical data, gathered from the plug. The state of the relay is represented by a `Boolean` and turning the field on and off in the context manager will be propagated to the physical plug, allowing to turn on and off connected electrical devices.

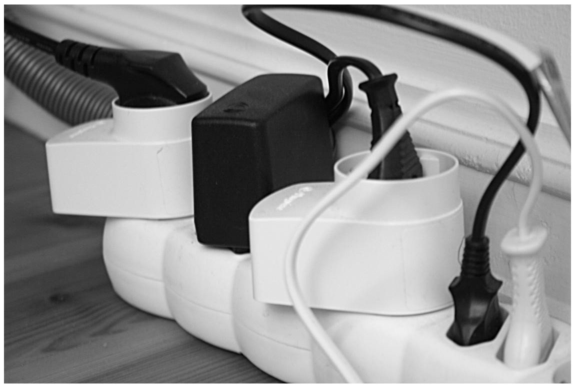

Figure 6. The form factor of the smart plugs from PlugWise (white plugs on the picture) make them easy to install across the house in order to measure the consumption of particular devices, but also to automatically turn on and off (sets of) devices based on heuristic such as the time of the day, the day of the week, or more advanced schemes in response to Demand/Response requirements from the grid.

[15]The COSM feed https://cosm.com/feeds/60040 is updated at two minutes interval with the current power consumption of the whole house

[16]In order to be able to resist to network equipment that restrict the use of WebSockets, the plugwise bridge is also able to poll at regular intervals for the desired state of the physical relay.

3) Spot Prices: The Swedish electricity market has been deregulated for a number of years and prices vary on an hourly basis, dividing Sweden in four different geographical regions. Nord Pool Spot runs the power market in Sweden and offers day-ahead prices to its customers. A Tcl script continuously acquire the prices[17] for all regions and updates one or several objects of the context to reflect the current price at that location.

C. Ambient Interfaces

An off-the-shelf multi-coloured lamp is put under the control of a REST-based server written in Tcl. Controlling of the lamp is via the IR from Dangerous Prototypes[18]. At present and for time reason, the solution only works on Windows, on top of WinLIRC. The lamp can take a wide number of colours and the REST interface accepts any RGB codes, approximating to the closest available colour on the lamp.

Figure 7. The design of the lamp makes it an acceptable display for home "events" in a number of cases.

A second Tcl REST server offers a Web interface to "tune" the lamp to various data sources present in the context manager. The user interface is kept to a bare minimum, but is easily accessible from both computers and mobile devices, which is the expected future scenario. Using colours, the lamp can visualise the live status of the heat pump (from green when not working to purple when using external heat), the temperature inside or outside, the price of the electricity on Spot, etc.

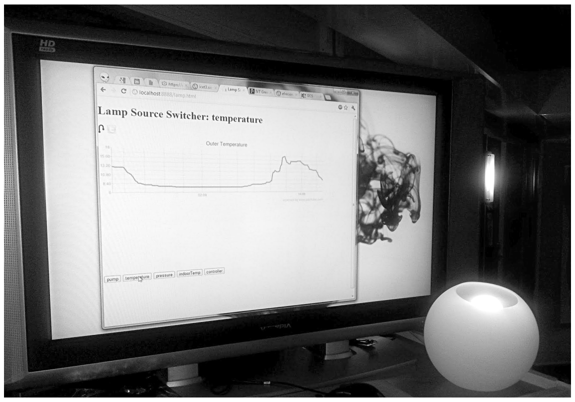

Figure 8. In the figure above, the lamp is tuned to the outside temperature and the user interface is shown on nearby TV for demo purposes. The UI uses the COSM connection to display relevant historical graphs.

VIII. Conclusions and Future Work

This paper has presented the context manager, a central hub designed both to provide contextual data to IoT applications and a storage for historical, present and future sensor and actuator data. The context manager is designed to lower the learning curve, letting less technically inclined people model and reason about their smart homes. Concepts such as pairing, conduits and triggers lean themselves easily to be controlled and specified via user interfaces rather than via the file formats that have been summarised in this document. As such, these concepts already contain parts of the logic that would control the flow of data between different objects, combined to both actuation and visualisation through external services. Possible extensions would consist in looking into visual programming efforts such as App Inventor[24] and ways to incorporate some of these ideas into the IoT domain. Already, colleagues have started to work on the

[17]Prices are scraped from http://www.nordpoolspot.com/ for historical reasons.

[18]The IR toy is a set of open source hardware and software available at http://dangerousprototypes.com/docs/USB_IR_Toy_v2 to record and (re)play the IR codes of most infrared-based remote controls.

building blocks of an “appification” of the home, i.e. the installations of “apps” that can control parts of your homes to attain some energy savings, while providing a (mobile) user interface to input settings and refine controlling. For this “appification” to take place, applications will need to be able to reason about the context in order to adapt to the specificities of as many homes as possible. They cannot rely on users or hard-coded objects, instead the location and search facilities of the context manager will be key to reasoning about the context and answers questions such as “give me all the lamp sources in that room” or “Have all inhabitants left home now?”. In its current implementation, the context manager is starting to be able to provide an answer to this type of questions.

Appendix A
Integrating WSN Sensors and Actuators

Apart from the TinyNode deployment that has been described in section VII, additional Tcl scripts have been written in order to interface with IPv6-based WSN, one of the areas where the OS Contiki[22] is widely used. In meshing WSN, it is essential to restrict the size and number of packets to a strict minimum in order to keep power requirements low. The current implementation of these scripts relies on the HTTP capabilities of the motes. HTTP leads to sizable headers and the necessity to keep the TCP state across the network. Future directions will look into UDP[19] and WebSockets.

Scripts bridging motes to the context manager will receive or poll for mote data and push this data as updates to the field of one or several objects in the context manager. The simplest script will regularly poll for data at given motes with a given frequency. However, the more complex script is inspired by techniques initiated by CoAP[26]. It combines a UDP and HTTP servers, in order to both support regular HTTP POST and GET operations, but also to entertain WebSockets connections. On startup, the script contacts all relevant motes and subscribes itself (the proper root/details to the servers that it implements), together with a frequency for reception of data. Consequently, motes will, whenever needed push data to the script, which will forward it further to the appropriate objects of the context manager, depending on its configuration.

[19]Problems with the current UDP implementations in Tcl 8.6 (and in combination with IPv6) have unfortunately put part of the development on hold.

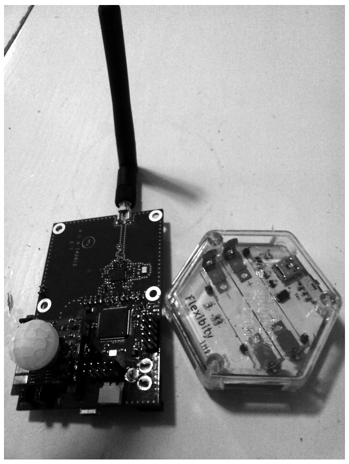
Figure 9. Two of the supported motes. To the left is a mote from Tyndall[25] that sports a stackable and pluggable interface for various sorts of sensors (temperature and humidity on the picture). To the right is a commercial mote from Flexibility (see http://www.flexibity.com/) implementing a thermometer, hygrometer and barometer.

Acknowledgments

Most of the work has been sponsored by the European ARTEMIS project me^{3}gas, with valuable input from M. Westbergh (CRL Sweden), S. Duquennoy (SICS), J. Eriksson (SICS), P. Kool (CNet), P. Hansson(SICS) and L. Moore (Tyndall). Most of the code has been opened source at the following project location: http://code.google.com/p/efr-tools/

References

[1] B. Welch, K. Jones, and J. Hobbs, *Practical Programming in Tcl and Tk.* Prentice Hall, 20 June 2003.

[2] K. P. Birman and T. A. Joseph, “Exploiting virtual synchrony in distributed systems,” in *ACM Symposium on Operating Systems Principles (SOSP’87)*, pp. 123–138, 1987.

[3] R. Fielding, *Architectural Styles and the Design of Network-based Software Architectures*, ch. 5, pp. 76–106. 2000.

[4] D. Crockford, “The application/json Media Type for JavaScript Object Notation (JSON).” RFC 4627 (Informational), July 2006.

[5] M. Banzi, *Getting Started with Arduino*. Make:Books, O’Reilly Media, Inc., Aug. 2011.

[6] S. Monk, *Getting Started with .NET Gadgeteer*. Make:Books, O’Reilly Media Inc., 4 May 2012.

[7] T. Berners-Lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Scientific American*, 17 May 2001.

[8] K. Aberer, M. Hauswirth, and A. Salehi, “A Middleware For Fast And Flexible Sensor Network Deployment,” in *Proceedings of VLDB’06*, pp. 1199–1202, 2006.

[9] P. B. Gibbons, B. Karp, Y. Ke, S. Nath, and S. Seshan, “IrisNet: An Architecture for a Worldwide Sensor Web,” *IEEE Pervasive Computing*, vol. 2, pp. 22–33, Oct-Dec 2003.

[10] D. Halvik, G. Schimak, R. Denzer, and B. Stevenot, “Introduction to SANY (Sensors Anywhere) Integrated Project,” in *Proceedings of ENVIRONINFO*, Sept. 2006.

[11] P. Kostelnik, M. Sarnovsk, and K. Furdik, “The Semantic Middleware for Networked Embedded Systems Applied in the Internet of Things and Services Domain,” *Scalable Computing: Practice and Experience*, vol. 3, no. 12, pp. 307–315, 2011.

[12] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. Frystyk Nielsen, A. Karmarkar, and Y. Lafon, “SOAP Version 1.2 Part 1: Messaging Framework (Second Edition).” W3C Recommendation, 27 Apr. 2007. http://www.w3.org/TR/soap12/.

[13] T. Bray, J. Paoli, C. Sperberg-McQueen, E. Maler, and F. Yergeau, “Extensible Markup Language (XML) 1.0 (Fifth Edition).” W3C Recommendation, 26 Nov. 2008. http://www.w3.org/TR/xml/.

[14] G. Klyne and C. Newman, “Date and Time on the Internet: Timestamps.” RFC 3339 (Proposed Standard), July 2002.

[15] E. Rescorla, “HTTP Over TLS.” RFC 2818 (Informational), May 2000. Updated by RFC 5785.

[16] S. Sanfilippo and P. Noordhuis, “Redis,” 2012. http://redis.io/.

[17] I. Fette and A. Melnikov, “The WebSocket Protocol.” RFC 6455 (Proposed Standard), Dec. 2011.

[18] A. Presser, L. Farrell, D. Kemp, W. Lupton, S. Tsuruyama, S. Albright, A. Donoho, J. Ritchie, B. Roe, M. Walker, T. Nixon, C. Evans, H. Rawas, T. Freeman, J. Park, C. Chan, F. Reynolds, J. Costa-Requena, Y. Ye, T. McGee, G. Knapen, M. Bodlaender, J. Guidi, L. Heerink, J. Gildred, A. Messer, Y. Kim, M. Wischy, A. Fiddian-Green, B. Fairman, J. Tourzan, and J. Fuller, “UPnP Device Architecture 1.1,” tech. rep., UPnP Forum, 15 Oct. 2008.

[19] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart, “HTTP Authentication: Basic and Digest Access Authentication.” RFC 2617 (Draft Standard), June 1999.

[20] P. Leach, M. Mealling, and R. Salz, “A Universally Unique IDentifier (UUID) URN Namespace.” RFC 4122 (Proposed Standard), July 2005.

[21] H. Dubois-Ferrière, L. Fabre, R. Meier, and P. Metrailler, “Tinynode: a comprehensive platform for wireless sensor network applications,” in *Proceedings of the 5th international conference on Information processing in sensor networks*, IPSN ’06, (New York, NY, USA), pp. 358–365, ACM, 2006.

[22] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki - a lightweight and flexible operating system for tiny networked sensors,” in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, pp. 455–462, 20 Dec. 2004.

[23] S. Petai, “python-plugwise.” Bitbucket Project, 20 Mar. 2011. https://bitbucket.org/hadara/python-plugwise/wiki/Home.

[24] D. Wolber, H. Abelson, E. Spertus, and L. Looney, *App Inventor*. O’Reilly Series, O’Reilly Media, Inc., 15 Apr. 2011.

[25] A. Lynch, K. Aherne, P. Angove, J. Barton, Harte S., D. Diamond, and F. Regan, “The Tyndall Mote. Enabling Wireless Research and Practical Sensor Application Development.,” in *Adjunct Proceedings, Advances in Pervasive Computing*, pp. 21–26, May 2006.

[26] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, “Constrained Application Protocol (CoAP),” Internet Draft draft-ietf-core-coap-12, IETF, 1 Oct. 2012.

Dr. Emmanuel Frécon is a senior researcher at the Swedish Institute of Computer Science (SICS). He received his Ph.D. from the IT university of Gothenburg in 2004. He has (co-)authored a number of articles in books, refereed conferences and journals, as well as edited a book in the field of computer science. Across the years his research interests have slowly shifted from collaborative virtual environments to ubiquitous computing, not forgetting ambient displays and novel interaction techniques. He strongly believes in the feedback loop between technology and users.

Dr. Emmanuel Frécon is also an entrepreneur and has co-founded two companies. He is on leave from his second company, JoiceCare, where he worked as a system architect and CTO. JoiceCare sells products for the elderly market: a SIP-based video telephone and a video-based supervision system. He also believes that industry and research have a lot to bring to one another and intends to alternate workplaces as opportunities present themselves.