Smart Plugs: a Low Cost Solution for Programmable Control of Domestic Loads

Giovanni Galioto*, Natale Galioto*, Costantino Giaconia*, Laura Giarré*, Giovanni Neglia+*, Ilenia Tinnirello*
*Department of Electrical Engineering, Universitá di Palermo, Italy
email: *name.surname*@unipa.it
+Maestro project, INRIA, Sophia Antipolis, France
email: *name.surname*@inria.fr

***Abstract*—Balancing energy demand and production is becoming a more and more challenging task for energy utilities. This is due to a number of different reasons among which the larger penetration of renewable energies which are more difficult to predict and the meagre availability of financial resources to upgrade the existing power grid. While the traditional solution is to dynamically adapt energy production to follow the time-varying demand, a new trend is to drive the demand itself by means of Direct Load Control (DLC).**

In this paper we consider a scenario where DLC functionalities are deployed at a large set of small deferrable energy loads, like appliances of residential users. The required additional intelligence and communication capabilities may be introduced through *smart plugs*, without the need to replace older "dumb" appliances. Smart plugs are inserted between the appliances plugs and the power sockets and directly connected to the Internet. An open software architecture allows to abstract the hardware sensors and actuators integrated in the plug and to easily program different load control applications.

I. Introduction

Load control in modern power grids is becoming more and more important for maintaining a balance between energy supply and demand. Traditionally, the demand was much more variable and less controllable than supply, so that the energy balance was achieved by adapting dynamically generation levels to match the consumption. The increasing penetration of renewable energies has radically changed the scenario, due to their lower predictability. The possibility to control power demand is then becoming more appealing for several actors, such as the energy utilities (that can better plan the production) and the end customers (that can actively participate to the energy market). Two main approaches are envisioned: demand-response and Direct Load Control (DLC). The first expression refers to end users changing their normal consumption pattern in response to a dynamic price signal. DLC denotes the possibility for the electric utility (or some third-party entities) to directly control remote electric loads.

However, despite the many proposals in the literature for demand response approaches [1] or direct load control [2], [3], the control of residential users' energy demand (which significantly affect the overall energy load variability [4]) is still limited to pilot projects [5], [6] with little penetration perspective in the near future. The reason is that the implementation of these mechanisms requires investments (for updating user appliances and communication infrastructures) which are not clearly justified for the end users.

978-1-4799-0959-9/14/$31.00

Recently, in [7],[8] it has been proposed a realistic deployment and low-cost path for large scale direct control of inelastic home appliances whose activation can be deferred. The idea is to exploit 1) the Internet connections of customers for transporting the admission control requests and 2) some simple actuators to be placed on the electric plugs for connecting or disconnecting non-smart appliances. Admission control of electric loads is performed deterministically [7] by a central server processing all the activation requests originated into a given control area, or probabilistically [8] by a local controller programmed by the central server. These previous studies have been focused on the theoretical analysis of the admission control policies for guaranteeing a maximum activation delay under a given limit on the power consumed by a group of users, by assuming that plug control devices are available.

In this paper we focus on the design of the devices to be inserted between the appliances' plug and the power socket. These devices are usually called *smart plugs* and are already produced by some vendors [9] with proprietary applications (mainly for monitoring purposes). Our idea is designing a smart plug able to be easily connected to the Internet according to the M2M communication paradigm and to run *customized applications*, including the interruption and reactivation of the current flow as a response to a *control protocol*. The application can be defined and installed by the end users, by the energy utility itself or by some other entities like an energy aggregator [10], [11], [12]. It generally requires to interact with a remote server controlling a number of aggregated households or direclty with other households. The connection to the Internet can be provided by an home gateway (a PC, an ADLS box or even a smartphone), while the communication between the smart plug and the gateway is based on local communication technology, such as WiFi.

After a brief introduction on the envisioned control architecture and mechanisms (section II), we describe the smart plug hardware and software architecture (section III) and the abstractions used for defining simple plug programs (section IV). Some final remarks are drawn in section V.

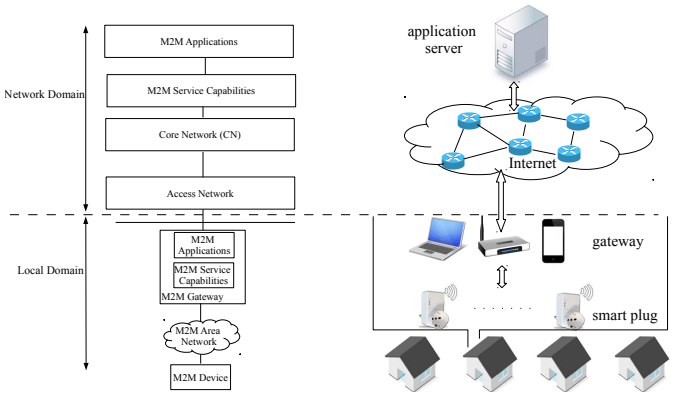

Fig. 1. Load control deployment in the local and network domain and mapping to the M2M high level architecture.

II. Domestic Load Control

DLC mechanisms are currently employed by large commercial and industrial customers, whose power demand has a significant impact on the aggregated demand of a power zone. The control is often performed by interacting with human operators and by exploiting the traditional phone network as a signaling network. In order to perform DLC on residential users, it is necessary to interact with a much higher number of customers, being the single power demand negligible in comparison with the industrial one. Moreover, residential users cannot be directly involved in the control decisions apart from the specification of a control profile, such as the selection of the appliances that can be deferred or the maximum tolerable delay. To easily reach a large scale number of users and support autonomous control mechanisms, as proposed in [7], we assume that the signaling messages are transported over the Internet and that machine-to-machine (M2M) applications run over the electric plugs to control the appliance activation.

A. Network Architecture

Figure 1 shows the envisioned architecture for implementing domestic load control: in each household, called local domain, control applications runs on a number of smart plugs connected to the Internet by means of a gateway; at the network domain, the admission control server (or equivalently an overlay peer-to-peer network) exchanges messages with different groups of households by means of Internet. The architecture is based on the reasonable assumption that an Internet connection is already active in almost all households, thus avoiding to deploy a dedicated data network. The home gateway can be a normal PC, an ADSL box or smart phone with a 3G data connection. The smart plugs, described in the next section, are embedded smart devices equipped with a local network technology, a power meter devised to detect (and/or to measure) the energy required by the connected appliance, and a processor able to run control and monitoring applications. The figure also highlights the mapping to the M2M high level architecture [13], which allows to simplify the development of the control application by using some management functionalities (such as service discovery, authentication, association, etc.) already available as M2M service capabilities [14].

In the network domain, the client-side application (running on the home gateways and on the plugs) is responsible of joining the application server (or the overlay network) and controlling its grid segment. The server-side application implements an admission control logic which depends on the aggregation level of end customers. By tracking the current state of the controlled power grid, the server sends the control commands to the clients in order to provide deterministic or probabilistic guarantees that the total energy demand of a group of customers does not overcome a given (time-varying) threshold. To this purpose, the server continuously interact with the clients for collecting activation requests of new appliances, as well as notifications of successful activation and deactivation of the admitted appliances. Since all these messages are delivered through the Internet, no delay guarantees can be assured between an activation request and response. This makes the architecture suitable for admission control logics pursuing an energy peak shaving or price control, while it is less reliable for reacting to emergency grid conditions.

In the local domain, the home gateway interacts with the smart plugs for collecting the activation requests, the activation/deactivation notifications, and other (more complex) data, such as the energy consumption of the plugs, when available. According to the programmed admission control logic, the activation requests are immediately forwarded to the application server [7] or processed locally for probabilistic decisions programmed by the server [8]. The messages exchanged between the home gateway and the smart plugs are typically sent through a wireless local area network working on unlicensed ISM bands (e.g. ZigBee and WiFi), where transmission reliability can vary significantly according to the experienced interference conditions.

B. Shaping of Aggregated Power Demand

Load control works by shaping the aggregated power demand of a number of users as conceptually shown in Figure 2, where the red curve represents the expected power demand and the blue curve is the actual demand under admission control. Note that load control does not change the total energy demand (the area below the curve) on a long time interval, but it can shape the instantaneous power demand by postponing the operation time of some appliances, for example with the goal to satisfy a power cap P_g. The power demand exceeding P_g is then shifted to a subsequent time interval. The power cap may be originated by power transmission constraints, by production constraints or by too high production costs the utility would incur to generate energy during the peak time.

Shaping the aggregated power profile can bring significant savings to the energy utility. Hence, customers installing a load control application may be motivated to accept the deferral of their appliances by some economic incentives. If load control managed by the energy utility itself, then these incentives may be in the form of some reduction of the users' electricity

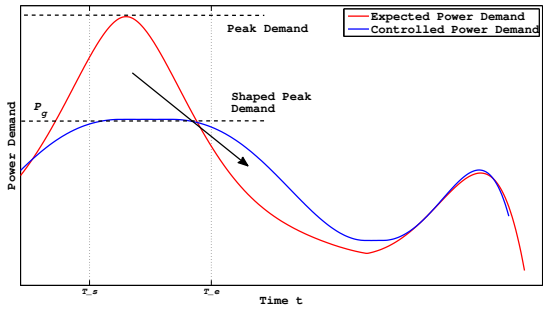

Fig. 2. Effects of load control on the expected power demand.

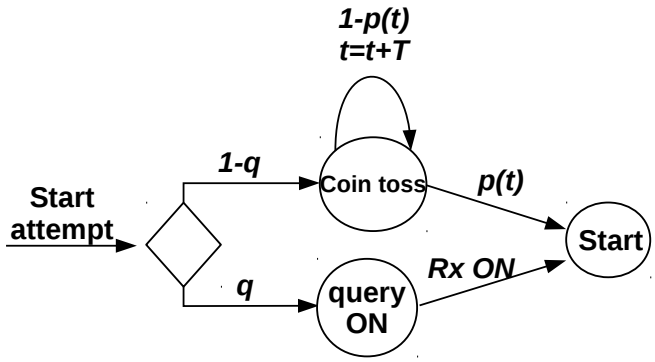

Fig. 3. Deterministic, probabilistic and mixed load control.

bill. Although we are not analyzing the most effective revenue sharing mechanism (e.g. uniform sharing, random extraction of selected users, etc.), we assume that customers perceive a reward proportional to the deferral time of their load requests (i.e. proportional to their discomfort) and to the energy consumed by the deferred loads after their admission. Such a solution disincetivizes customers from removing the smart plugs or implementing other misbehaviors.

C. Load Control Applications

Load control applications can be implemented by considering deterministic or probabilistic guarantees on the aggregate power demand or by mixing the two previous approaches.

Deterministic Approach [7]. It is assumed that the supplier communicates with an adequate advance to the aggregation server its demand expressed by a cap on the maximum absorbed power to be enforced during a specific time interval. The aggregation server is able to keep the total demand state of the controlled households and enforce the supplier's demand by disconnecting a subset of the plugs (postponing the load of the corresponding appliances). The plug control is performed only upon a novel activation request, thus avoiding to disconnect already on appliances. The continuous monitoring allows the aggregation server to reconnect some plugs if the instantaneous power demand is below the cap during the controlled period. The approach allows to achieve the maximum efficiency under the required power constraint, at the expense of high communication costs (being all the activation requests forwarded and processed by the central server).

Probabilistic Approach [8]. It is assumed that the supplier demand is expressed by specifying that the instantaneous power demand can exceed a given bound with a probability smaller that ϵ. To satisfy the demand, the aggregation server characterizes stochastically the power demand of the households and periodically broadcasts an activation probability function for each appliance in different hours of the day. Control decisions are open-loop, because they are taken locally without further interacting with the server. This allows to avoid disclosing private information of the users (about the time instants in which they desire to active the appliances), but it achieves an average lower utilization of the resources because it cannot rely on the exact knowledge of the aggregated power demand.

Mixed Approach. In case of probabilistic supplier demands (equal to the ones considered in the previous case), an alternative solution is combining the deterministic and probabilistic approaches, by forwarding the appliance activation request to the aggregation server with a given probability q, and by locally processing the request with probability $1 - q$. The parameter q can be tuned in order to achieve the wished trade-off among resources' usage, control overhead and privacy leakage.

Figure 3 depicts the work-flow of the mixed approach as a combination of the deterministic approach (based on queries to the aggregation server) and probabilistic approach (based on random local decisions). Although all the previous schemes have been described according to a centralized architecture, i.e. by assuming that a central aggregation server is available for processing the requests, monitoring the power demand or broadcasting the probabilistic activation profiles of the loads, it is also possible to consider distributed implementations, in which the information and decisions about the power demand of a number of households to be aggregated are managed by an overlay peer-to-peer network.

III. Smart Plug Design

Most of commercial smart plugs have a very limited programmability of sensing and controlling functionalities and can interact only with proprietary applications provided by the vendors. Our design has been inspired by these limitations for achieving ease of programming on an *open software platform*, with advanced communication and processing capabilities, and supporting heterogeneous hardware sensors and actuators, with a cost comparable with current commercial solutions (about 50$). To accomplish these goals, we chose to work on an embedded system, running an operating system with a M2M stack directly on the plug, and to provide some basic primitives that can be opportunistically exploited by third parties for defining different control applications.

A. Hardware Architecture

The envisioned hardware architecture includes four main sub-systems:

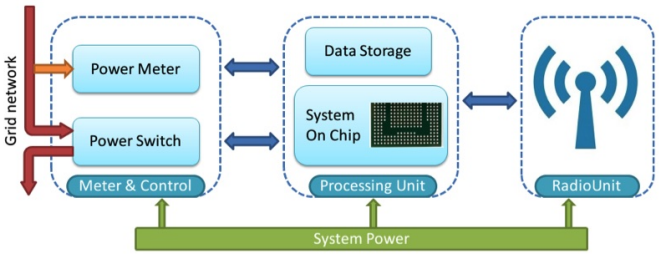

Fig. 4. Hardware components of the Smart Plug.

- **Processing Unit.** It represents the core of the device, responsible of orchestrating the interactions between sensors, actuators, communication modules and programs. We chose to use a System on Module component by Acme Systems, called ARIA G25, which provides an ARM926EJ-S processor at 400 MHz, 128 or 256 MB of memory, USB ports, and several I/O interfaces such as I2C, SPI, GPIO, etc. An SD card is required as a non-volatile memory. The choice has been motivated by the very compact size of the module and by its excellent cost/performance tradeoff.
- **Network Unit.** It is responsible of providing the physical connectivity to the network infrastructure for interacting with the remote controller and the other network nodes involved in the programmed control application. In our design, the system is based on a WiFi interface supporting connectivity to a local gateway. The specific component used in the prototype is the OEM WiFi USB module based on the Ralink chipset RTS5370N IEEE 802.11b/g/n compatible.
- **Power Meter.** It allows to sample the energy consumption of the appliances connected to the plug according to the accuracy and time granularity programmed by the application. The system design allows to easily integrate meters based on different technologies and communication interfaces. In the prototype we used the ADE7753 power meter by Analog Devices, which includes two 16-bits sigma-delta converters, a thermal sensor, a frequency meter, and an evaluator of RMS voltage and current values. The communication interface is SPI.
- **Power Switch.** It is the actuator responsible of turning the appliances on and off. The system is implemented by means of a power switch controlled by the application.

Figure 4 summarizes the hardware elements of the smart plug. The figure does not explicitly shows the power supply module, whose design does not require special attention (being the plug connected to the power wires).

B. Software Architecture

As most of M2M applications, we assume that smart plug applications are targeted to monitoring and control: they collect the data provided by the sensors, filter and aggregate the data in order to detect specific events, and react to the events by driving the actuators. We chose the *Mihini* framework as

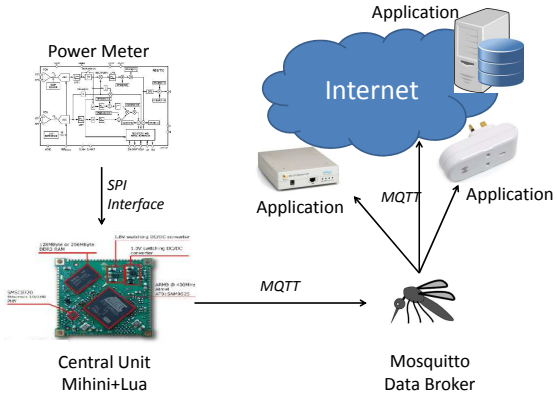

Fig. 5. Communication model between data producers and data consumers in smart plug applications.

an open source project providing these functionalities and the programming language *Lua* for defining some reference applications. More into details, the main features of the software components used in the project are:

- **M2M Libraries.** Mihini allows to easily distinguish between general functions to be integrated in any M2M application (*send data, receive commands*) and specific functions which depend on a particular application (*process data, react to events*). The first set of functionalities is managed by an autonomous Mihini agent, while the second set is specified by the programmer in the application *container*. The Mihini agent is responsible of the application time life, configuration, data structures, and communication protocols. The container provides the libraries for I/O operations based on most industrial protocols, data storage, assignment of task priorities and scheduling policies, definitions of event reactions functions.
- **Lua Language.** Lua is a very simple language, specifically defined for M2M applications on embedded systems, with a very powerful expressiveness which results in compact and efficient code. It has also been shown that the language is very robust to programming errors, especially in the management of complex data and conflicts due to parallel threads. It is also possible to easily interface Lua programs with C routines.

C. Communication Protocols

Figure 5 shows the main hardware and software components used in the smart plug design and the relevant communication protocols. The figure distinguishes between the *data production* sub-system (i.e. the power meter) from the *data consumption* sub-system (i.e. the application), that may be run by the home gateway, a remote server or the smart plug itself according to the specific application. While the communication between the sensor and the processing unit is conceptually very simple (e.g. a serial interface managed by vendor-specific

protocol), the communication between the unit and all the control applications may require to interact with a potentially large number of nodes, such as all the domestic plugs or all the plugs of a given neighborhood.

The solution adopted in our design for allowing an asynchronous communication between multiple sensors and distributed applications is using a *broker* for decoupling the data production and data consumption systems. The solution is adopted in several other M2M applications: data generated in a given context (called *topic*) are published (i.e. made available) to all the applications that declared to be interested to the context (i.e. subscribed a given topic). The broker tracks all the subscriptions and publication records and forwards the data received by each sensor to all the relevant subscribers.

The implementation details of our prototype are summarized in what follows.

- **Mosquitto Broker.** The Mosquitto broker is an open source broker for M2M applications, that can be run by the home gateway in case of deterministic (centralized) load control or by the plug in case of probabilistic load control.
- **MQTT Protocol.** MQTT has been explicitly designed for networks of simple sensors and actuators. The protocol is message oriented, with a fixed header, variable fields, and a minimum size of two bytes only. Bidirectional communications are established between the sensors and the server, the server and the subscribers, by specifying quality of service parameters for data delivery. The protocol has been shown to significantly reduce the network traffic and latencies in comparison with other protocols under a push/push data communication model.

IV. Developing Smart Plug Applications

The smart plug design allows to easily develop monitoring and control applications for domestic load control. The overall implementation of load control policies requires the design of a distributed application which involves, as described in section II, different network nodes (plugs of multiple users, home gateways, servers). An important component is the application running on the plug itself, that can be written in Lua by also exploiting some simple functionalities developed within our project. In order to show the effectiveness of Lua abstractions for defining compact and robust code, we discuss some simple programming examples.

Power Sampling. Being the primitive *readPower* available for measuring the active power consumed by the appliance connected to the plug, different sampling schemes can be programmed by simply specifying the scheduling policies of consecutive readings. The scheduling can be periodic or event-based. For example, the following code implements a periodic reading performed every second:

```
while true do
  local power = readPower()
  mqttClient:publish(MQTT_DATA_PATH.."Power", power)
  sched.wait(1)
end
```

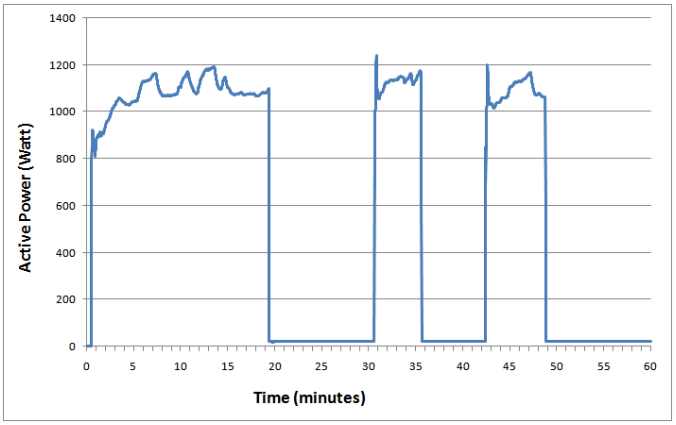

Fig. 6. An example of power demand trace of an air conditioner monitored with the above Lua code.

where the function *mqttClient:publish* is natively provided by the Mihini libraries for publishing the reading result to the broker by using the MQTT protocol. The periodic sampling application can be used for characterizing the time-varying power demand of different electric appliances. An example of resulting power trace is shown in figure 6, where the active power of an air conditioner placed in our lab has been monitored for 60 minutes. The figure clearly shows that the air-conditioner has some activity cycles that can be exploited for introduce some flexibility in the power demand (for example, by slightly delaying the starting of an activation cycle).

Local Power Control. The previous example can be easily extended for implementing a simple plug control based on the instantaneous power demand. The programmer could specify the maximum demand power admissible from a given plug and the reactivation policy in case the plug is switched off as follows:

```
while true do
  local power = readPower()
  mqttClient:publish(MQTT_DATA_PATH.."Power", power)
  if power > 250 then
    switchOff()     -- Switch off the appliance
    sched.wait(10) -- Wait 10 seconds
    switchOn()      -- Switch on the appliance
  end
  sched.wait(1)
end
```

where *switchOn* e *switchOff* are the primitives responsible of power switching control, and the reactivation policy is based on a fixed delay of 10 seconds. Figure 7 shows a log example of previous application.

Message Parsing. MQTT messages can be easily extended

```
16:36:25: Meter Read - Active Power : 101W - Publish to MQTT Power
16:36:26: Meter Read - Active Power : 101W - Publish to MQTT Power
16:36:27: Meter Read - Active Power : 301W - Publish to MQTT Power
16:36:27: Threshold : 250W - Active Power : 301W
16:36:27: Turn OFF for 10 seconds
16:36:37: Turn ON
16:36:40: Meter Read - Active Power : 305W - Publish to MQTT Power
16:36:41: Meter Read - Active Power : 305W - Publish to MQTT Power
16:36:42: Meter Read - Active Power : 305W - Publish to MQTT Power
```

Fig. 7. An example of application log in case of plug switch off.

with customized fields in order to support the signaling messages required by the distributed load control application. For example, power samples could be notified to a remote controller or control commands could be received from the remote controller. The following example shows how to program a message parser and a reaction mechanism to a *Switch* message sent by the broker.

```
-- Utility function to parse the message
local function Split(path,sep)
    local t = { }
    for w in string.gfind(path, "[^"..sep.."]+") do
        table.insert(t, w)
    end
    return t
end

-- Reaction to MQTT messages
local function process_mqtt(topic, value)
  local data = Split(topic, "/")[3]
  if data == "Switch" then
    if value == "0" then
      switchOff()
    else
      switchOn()
    end
  end
end
```

where the plug disactivation (or activation) is performed according to the message parameters.

Admission Control. Admission control of appliances can be programmed by exploiting the *publish/subscribe* communication model. When a power consumption different from zero is revealed by the meter, an admission request can be sent to the broker. The broker can forward the request to the remote controller or can probabilistically decide if admitting or not the appliance. In case the appliance is not admitted, the request can be rescheduled after a fixed time interval. This mechanism can be programmed as follows:

```
local secondsToKeepApplianceSwitchedOff = 0

-- Reaction to MQTT messages
local function process_mqtt(topic, value)
  local data = Split(topic, "/")[3]
  if data == "LoadAdmissionControlCheck" then
    secondsToKeepApplianceSwitchedOff = tonumber(value)
  end
end

while true do
  local power = readPower()
  mqttClient:publish(MQTT_DATA_PATH.."Power", power)
  if power > 10 then
    mqttClient:publish(MQTT_DATA_PATH.."
        LoadAdmissionControlCheck", power) -- Publish an
        admission control check request
  end
  if secondsToKeepApplianceSwitchedOff == 0
    switchOn()
  else
    switchOff()
    secondsToKeepApplianceSwitchedOff =
        secondsToKeepApplianceSwitchedOff - 1
  end
  sched.wait(1)
end
```

where the admission control is performed as soon as the power demand is higher than 10W.

V. Final Remarks

The role of direct load control in modern power grids has been shown to be beneficial for several applications. However, in the case of small individual energy loads, these benefits can be appreciable only if a large number of users are involved in the control process.

The main contribution of this paper is proposing a solution for implementing load control mechanisms without using novel smart appliances, thus allowing a prompt user penetration. We present the design of a smart plug equipped with a local network technology, a power sensor devised to detect (and/or to measure) the energy required by the connected appliance, and a microprocessor able to run simple applications. The envisioned architecture is M2M compliant (thus avoiding any user interaction for expressing the energy demand ahead of time) and completely open to third party programmers. Security aspects dealing with the identification of authorized programs to be run by the plug are currently under investigation.

References

[1] V. Balijepalli, V. Pradhan, S. Khaparde, and R. M. Shereef, "Review of demand response under smart grid paradigm," in *Innovative Smart Grid Technologies - India (ISGT India), 2011 IEEE PES*, Dec 2011, pp. 236–243.

[2] Y. Y. Hsu and C. C. Su, "Dispatch of direct load control using dynamic programming," *IEEE Transactions on Power Systems*, vol. 6, no. 3, pp. 1056–1061, 1991.

[3] G. B. Sheble and K. H. Ng, "Direct load control-a profit-based load management using linear programming," *IEEE Transactions on Power Systems*, vol. 13, no. 2, pp. 668–694, 1998.

[4] "Buildings energy data book." Tech. Rep. [Online]. Available: http://buildingsdatabook.eren.doe.gov/default.asp

[5] D. J. Hammerstrom, "Pacific northwest gridwise testbed demonstration projects part ii. grid friendly appliance project," Pacific Northwest, Tech. Rep. PNNL 17079, 2007.

[6] "Fpl on call saving program," Tech. Rep. [Online]. Available: www.fpl.com/residential/energy_saving/programs/oncall.shtml

[7] G. Di Bella, L. Giarré, M. Ippolito, A. Jean-Marie, G. Neglia, and I. Tinnirello, "Modeling Energy Demand Aggregator for Residential Users," in *IEEE CDC*, 2013, pp. 6280–6285.

[8] G. Neglia, G. Di Bella, L. Giarré, and I. Tinnirello, "Unidirectional Probabilistic Direct Control for Deferrable Loads," in *IEEE INFOCOM Workshop on Communications and Control for Smart Energy Systems (CCSES)*, 2014.

[9] "Smartplug," https://www.alertme.com/products/smartplug-1622.html.

[10] O. Abdalla, M. Bahgat, A. Serag, and M. El-Sharkawi, "Dynamic load modelling and aggregation in power system simulation studies," in *Power System Conference, 2008. MEPCON 2008. 12th International Middle-East*, March 2008, pp. 270 –276.

[11] A. Abdisalaam, I. Lampropoulos, J. Frunt, G. Verbong, and W. Kling, "Assessing the economic benefits of flexible residential load participation in the dutch day-ahead auction and balancing market," in *European Energy Market (EEM), 2012 9th International Conference on the*, 2012, pp. 1–8.

[12] "Energy-pool," March 2013, http://www.energy-pool.eu.

[13] "ETSI TS 102 690 V0.1.2 (2010-01), Machine-to-Machine Communications (M2M): Functional Architecture."

[14] "M2m development tools: Koneki, url: http://www.eclipse.org/koneki/."