A Dynamically Reconfigurable Architecture for Smart Grids

David Villa, Cleto Martín, Félix Jesús Villanueva, Francisco Moya and Juan Carlos López, *Member*, IEEE

Abstract — *This paper introduces a proposal for a smart grid management platform built on the distributed programming paradigm. Besides it is based on an information model and event architecture. The information model defines a set of clear interfaces to manage every "grid node". We built an actual implementation of a low-cost embeddable control/meter device that may even be attached to individual appliances. These devices behave as conventional autonomous remote distributed objects and provide full support for the information model and also for integration with the communication middleware. The resulting event architecture provides great flexibility to manage information flows from services and applications*[1].

Index Terms — smart grids, middleware, information model.

I. INTRODUCTION

Today there are many commercial devices that allow controlling any electric appliance, even in a remote way. Recently, concepts like *smart grid* or *Advanced Metering Infrastructure* (AMI) emerge for electrical power management and measurement purposes. However, related standards do not exist, nor programming interfaces, so they cannot be widely used yet.

In this paper, we propose a system for controlling and measuring the electrical power consumption for *large scale* and *heterogeneous* infrastructures. Intended to be used in smart grids, this proposal offers the following valuable features [1]:

- *Adaptability*: it makes it possible to build *Embedded Metering Devices* (EMDs) to manage arbitrary level facilities, from individual appliances to whole buildings or larger.
- *Scalability*: it scales easily with the number of nodes or equipments to be managed thanks to the middleware common services: event service, replication, indirect binding, etc.
- *Availability*: because of its distributed nature, this proposal allows that some network segments become temporarily isolated due to some failure and they may still continue to work in an autonomous way.
- *Hierarchical*: different types of infrastructures (house, building, residential area, etc.) may be integrated in a transparent way to build federated domains.

Furthermore, this proposal also provides other advanced features that are desirable in smart grids:

- *Small size*: in many appliances, the EMD may be very small and simple, as small as to be installed in an electrical junction box or even inside the bulb lamp base.
- *Low cost*: the EMD is cheap enough so that its cost is negligible in relation to the installation and maintenance of the controlled charge.
- *Low consumption*: obviously, the EMD power consumption must be insignificant in relation to the controlled charge.
- *Flexible access*: the platform is able to employ several communication systems, from telephone lines to standard Internet connections, or any kind of wireless network with the appropriate gateway.
- *Access transparency*: transparent and homogeneous remote access is allowed even with technologically different communication and computational resources (both hardware and software).
- *Conventional* tools: it allows using well-known and well tested protocols and software tools. That avoids additional costs and efforts derived from developer training.
- *Reconfiguration*: it lets the system customer/ administrator to easily change the logical configuration of the smart grids (the domain under its control). A switch would act over one or tens of power lines with a simple runtime reconfiguration.

This paper is organized as follows: Section II describes the current related work of smart grid architectures and management systems. Section III introduces our proposal and their main components: the OO (Object Oriented) communication middleware concept is briefly discussed and its terminology is presented at section IV, the design of EMDs and its classification is described in section V and the section VI shows the proposed information model. When every node of the smart grid is deployed, system

[1] This work is supported by ERDF and the Regional Gov. of Castilla-La Mancha under grants PAI08-0234-8083 (RGrid). Also, it has been supported by Spanish Ministry of Science and Innovation under grants CEN-20091048 (Energos) and TEC2008-06553 (DAMA).

All authors are with the Department of Information Technologies and Systems, University of Castilla-La Mancha, Spain. School of Computer Science, Paseo de la Universidad, nº 4. 13071, Ciudad Real (Spain) (e-mail: {david.villa, cleto.martin, felixjesus.villanueva, francisco.moya, juancarlos.lopez}@uclm.es).

administrator needs to configure and maintain the grid. Section VII describes which services may be useful for these tasks. An example of dynamically reconfigurable infrastructure is explained in section VIII. Section IX introduces one of our prototypes, where results about footprint are provided and, finally, we draw some conclusions and future work in section X.

II. Related Work

Recently, smart grids are becoming an interesting research field due to their potential and impact on industry, economy and society. Many works address the need of a dynamic behavior and configuration of smart grids [2]-[4]. This is a requirement to implement the following functionalities in a *smart* grid:

- *Smart reactions to faulty conditions*: applications may monitor different measurement parameters to detect and predict fault occurrence in any part of the grid. Thus, Smart Grids should provide an adaptive control for power supply and consumption in case of system malfunction or when certain parts of the grid are down. In this situation, the grid may be reconfigured to isolate the faulty zone by using an architecture that provides dynamic control.
- *Dynamic load balancing*: usually, different parts of the grid are not fed with the same energy at the same time. Different consumption profiles or time zones contribute to unbalance the energy grid. In short, smart grids should adapt energy production to current consumption demands of users. Reconfiguration makes possible a smarter and automated redistribution of the load through the grid.
- *Flexible configuration*: a reconfigurable architecture provides a way to customize the infrastructure to the end users needs without additional hardware installation. For instance, users might manage and monitor every home appliance (including those that may be installed in the future) with the same "control panel".

  In a similar way, but on a larger scale, electrical power companies and energy providers may use this flexibility for managing and monitoring substations, energy generators and so on.

A recent proposal is FREEDM System [5], a distributed hardware/software infrastructure, focused on MicroGrids [6] and renewable energies, for controlling and storing electric power. This system uses a collection of smart services, so called Distributed Grid Intelligence (DGI). Algorithms, like smart load balancing [7], are implemented at DGI and provide functionality and knowledge to users and high-level applications. FREEDM is a good solution if all of the hardware nodes and network protocols are homogeneous. Our proposal is based on smart grids that are built on different hardware platforms and different network protocols and, at the same time, the entire system should be seen in a homogeneous way. In fact, our architecture might integrate a FREEDM-based smart grid into the system.

Other research works have been carried out in relation with the Advanced Metering Infrastructure (AMI) concept. These platforms are focused on measurement, monitoring and accounting processes for smart grids. For instance, ZAMI [8] is a ZigBee based AMI for monitoring electric system in buildings. The AMI is separated logically from the rest of the system, so it is not integrated. A service perspective is able to provide access transparency and the integrative view of the system [9]. In this case, Web Services are proposed to be used as application-level network protocol to build AMIs. This provides hardware and protocol abstractions but the need of a XML parser implies non-trivial hardware resources. Instead, in our proposal, a basic meter able to provide voltage, intensity and energy consumption data has a footprint about 340 bytes on a single 8-bit micro-controller.

Most of the current literature about smart grid is focused on measurement tasks [10]. Recently, multi-agent based proposals have appeared, as in [11]. However, they do not provide data about hardware/software requirements for an actual installation and deployment.

We have not found any former approach dealing with reconfigurable architectures for smart grids with heterogeneous components at different levels that allows applications, power supplier and end-users to reach fine grained control and measurement functionality.

III. A Reconfigurable Platform for Power Management

Our target is a generic platform suitable to develop advanced electric power management services and applications for any environment and provide core mechanisms and support for them. With this in mind, we employ object-oriented distributed communication middleware to deal with inevitable heterogeneity in smart grids and to provide a standard protocol for the whole system. That platform is called CoSGrid (*Controlling the Smart Grid*).

On top of the base of the distributed programming paradigm, we define an information model that will be shared among the grid components. To achieve that, all involved parties must understand the underlying application protocol (imposed by the chosen middleware). That is not a problem for company computers (even in substations) but we are talking of *all* of the grid components, including user home appliances, wall switches, bulbs and virtually any electric device susceptible to be connected to an electric outlet. That conceptual approach requires an actual implementation that we achieve by means of the Embedded Meter Devices.

When all of the system components can be managed as objects and they share the same information model, it is possible to establish logic relations among them. By means of event channels, and object and data aggregation it is easy to perform non-trivial services, as we show in the section VIII.

Therefore, the platform is based on the next components that are explained in detail in the next sections:

- The communication middleware.
- The Embedded Meter Device.
- An information model.
- A set of core services.

IV. COMMUNICATION MIDDLEWARE

The platform is based on standard Object Oriented Middlewares (OOM) like CORBA [12] or similar [13], [14]. Thus, the system may be seen as a set of distributed objects that share information via remote method invocations. The middleware provides a uniform, generic and fully specified application protocol. There are many general purpose middlewares supported by the industry[2].

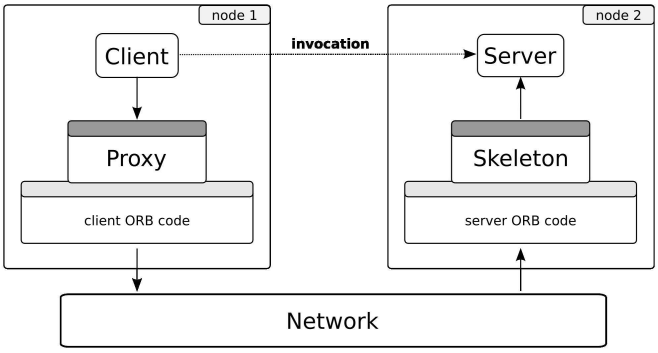

Fig. 1. Simplified invocation schema in object-oriented middleware.

Fig. 1 shows the essential behavior of this kind of middlewares. From the programmer point of view, invocation occurs as usual in the object oriented paradigm (dotted arrow). Actually, it does not happen in this way. Client invokes a method on a reference of the remote object: the proxy. Using the communication core, the invocation is coded (*marshalled*) using a specific binary protocol and transmitted to the server using the underlying network. At the server side, the invocation is re-built (*unmarshalled*) and finally it arrives to the object; the reply goes back to the client in the same manner.

The server side implements a well-defined *interface* that clients know and share. Interfaces are usually defined using an interface description language, like IDL in CORBA. Both proxy and skeleton parts are dependent of the interface and may be generated by tools, usually provided by middleware vendors.

V. EMBEDDED METER DEVICES

The EMD allows encapsulating the sensor (electrical magnitude measures) and actuator (control) to show the appliance as a distributed remote object. This is a powerful abstraction that lets the platform operate in a seamless way. All of the services deal with remote object references, without any knowledge about the underlying device nature: sensor technology, network access, computing platform or any other detail.

Every EMD must be capable to hold one or several distributed objects of the selected middleware. EMDs are *autonomous* and just need conventional networking support, like routers, bridges or gateways between technologies.

Generally, as shown in Fig. 2 the EMD is composed of: a microcontroller, a network interface, an electrical switch for turning on/off the electric load and measurement devices for monitoring such electric load.

Due to the distributed object oriented paradigm, each object in the EMD must implement a set of interfaces, that is, the contract with their clients. In this sense, a distributed object may be seen as a service: the client can access this service independently of its location (so called location transparency) and the technology in which it has been implemented. However, to provide these features different hardware requirements are needed.

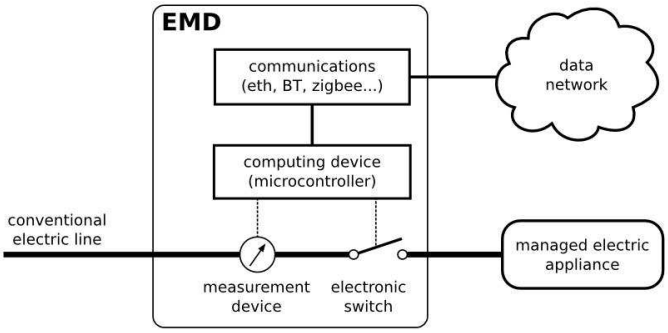

Fig. 2. Block diagram of the general EMD structure.

We provide a classification of different EMD implementations according to their characteristics, goal or performance. EMDs can be classified in three different basic types (see Fig. 3 for an example of deployment):

A. Low range EMD (L-EMD)

For simple electric loads like a light or electric outlet, L-EMD can modify, know and transmit the state (typically on/off) of the controlled devices by implementing the *Control* interface (see section VI). If the infrastructure requires fine-grain measured values, EMD-L devices may implement the *BasicMeter* interface. Keep in mind that L-EMDs are designed to be integrated into appliances and devices that consumer will use. They should be cheap and easy to deploy, so the functionality they may provide should be simple too.

To integrate them into the whole system, we need to build distributed objects into devices with a few K-words of memory and a single 8-bit microcontroller. To achieve that it is used the picoObject approach [15]. picoObjects are being used to implement the smallest standard distributed objects (hundreds of bytes) in a wide variety of embedded devices, including the cheapest microcontrollers.

In a common deployment, many L-EMD devices will be deployed. Each L-EMD needs a power line and a data

[2] Our EMD current prototypes work with the Ice middleware.

connection as Fig. 3 shows in *ground* floor. A multi L-EMD may be used to reduce cost and simplify the deployment process. The multi L-EMD behaves as a set of L-EMD that are accessible individually and remotely, but it employs only a computing and communication device. We have built a prototype of a multi L-EMD device which is described in section IX.

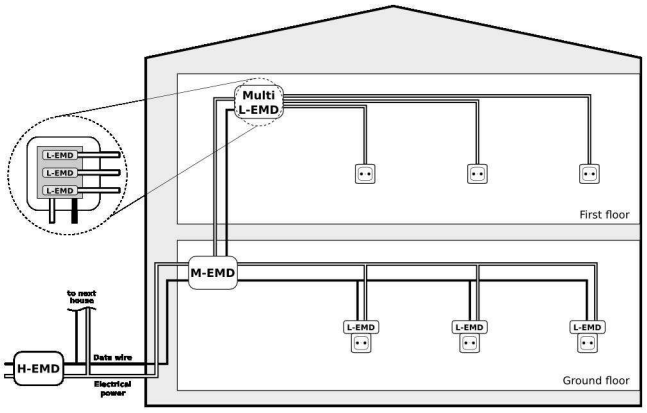

Fig. 3. Example of deployment where different types of EMD are used.

B. Medium range EMD (M-EMD)

This type of EMDs includes all functionality of the L-EMD but adds some basic properties for measurement of consumed power, voltage and electrical current. Furthermore, to deal with scalability issues, M-EMD provides aggregation mechanisms that let us read and modify any amount of devices, as if they were all a single one. This class of devices is designed for the installation in the low-voltage electrical panel.

In order to implement these properties, this class of device requires a 16-bit microcontroller, due to their needs of more memory and also more in/out ports for several sensors. M-EMDs may support, if desired, routing functions between the managed (local) area network and the global (external) system network.

C. High range EMD (H-EMD)

This class of devices requires a more powerful embedded device because they may store logged data (collected remotely or locally) about measurements and power statistics, voltage and current. H-EMDs are a good example of the *smart meters* supposedly provided by the electricity supplier. Due to its goal, these EMD are accessible only to companies and they may decide whether it requires remote control. The power company may need this kind of functionality at upper level (perhaps at substations) but those EMD are essentially the same of H-EMD.

VI. INFORMATION MODEL

Our platform (CoSGrid) provides a set of abstractions that may be used to model and design platform services as distributed applications. To address all of the requirements and objectives exposed before, the information model is broken down into several non-exclusive categories:

- Metering. To access electric magnitude measurements. It is functionally independent of the scale of the associated load.
- State control. To activate/deactivate an arbitrary load.
- Notification. To allow the nodes to send asynchronous updates of their measures or states.
- Node aggregation. To organize devices (their object references) in arbitrary ways: functional, geographic, importance.
- Data aggregation. To build "virtual" objects that represent composed measures for a set of nodes by means of an operator.

The next sections explain these interfaces, their possibilities, features and limitations.

A. Controlling and Measuring

The basic CoSGrid infrastructure provides interfaces for controlling and measuring tasks. Interfaces for measuring are described in Slice language as follows:

```
module CoSGrid {
   interface BasicMeter {
      short getVoltage();
      short getIntensity();
      int getPower();
   };
   interface AdvancedMeter extends BasicMeter {
      byte getHarmonic(byte n);
      int getEnergy();
   };
};
```

Depending on hardware features, sensor objects may implement the *BasicMeter* or *AdvancedMeter* interface. Using the first one, clients may collect information relative to voltage, current intensity and power from sensors. Advanced sensors which are also able to determine n-harmonic using a FFT [16] and the consumed energy, should implement the *AdvancedMeter* interface. This information is useful for provider companies to prevent side-effects due to harmonics and to predict grid malfunctions.

On the other side, CoSGrid provides the following interfaces for controlling tasks:

```
module CoSGrid {
   interface Status {
      bool isEnabled():
      bool isUsed();
   };
   interface Control {
      void setEnable(bool value);
   };
};
```

Distributed objects representing actuators should implement the *Control* interface that enables clients to change their state.

An actuator controls one electric lines, regardless of its load. Furthermore, if actuators are able to provide its state, objects may implement *Status*, so that clients would be able to know if the electric line is *in use*, that is, something is consuming electric power in that moment (the current is greater than zero); or if the line is *enabled*, that is, it is ready to provide power.

Of course, it is possible to have not-controllable objects that may implement the *Status* interface, although in that case, the *isEnabled()* method returns always *True*.

B. Event notification and logic relations

A monitoring object (that usually implements the *Status* or *Meter* interfaces, or both) may send its state to other objects by means of method invocations too. The former (the object that send updates) is called *active object* and the latter (the one that receive them) is called *observer* [17]. The observer is, of course, a remote object and therefore it is required that it implements well-defined interfaces to indicate the signature of its supported methods. Note that, when the active object sends notifications to its observer it is acting as a client.

The point here is that the observer of an active object may be changed at any time, even in a working system. To do that, the active object exposes specific interfaces that allow others to modify the reference of its observer. The observer *object reference* is persistent and is stored in the active object flash memory. That feature allows the system to be dynamically reconfigurable because these flows of information may be changed by very high abstraction level services.

This kind of object reference supports interoperability (they may be shared through the network), keeping the reference to the correct object: its *remote proxy*.

The next listing shows the CoSGrid interfaces related to *active objects* and *observers*.

```
module CoSGrid {
    interface MeterObserver {
        void setVoltage(short value);
        void setIntensity(short value);
        void setPower(int value);
        void setHarmonic(byte n, int value);
        void setEnergy(int value);
    };
    interface ActiveMeter {
        void setMeterObserver(MeterObserver* observer);
        MeterObserver* getMeterObserver();
    };
    interface StatusObserver extends Control {
        void setUsed(bool value);
    };
    interface ActiveStatus {
        void setStatusObserver(StatusObserver* observer);
        StatusObserver* getStatusObserver();
    };
};
```

Active objects are those that implement *ActiveMeter*, *ActiveStatus* or both. Trough these interfaces, clients are able to get and set/change the observer object at execution time. That is, the object which will receives the measurement or state updated value notifications of the active object. This observer object must implement a concrete observer interface depending on active object type. An *ActiveMeter* requires a *MeterObserver* object and, in the same way, an *ActiveStatus* requires a *StatusObserver* object.

Note that actuators (objects implementing *Control* interface) may be active objects too, both for metering and status purposes. You may assume that a non-controllable line or charge is always enabled, but may be being used or not.

Active objects introduce a new way to implement different kinds of relationships between objects:

- *1-to-1*: this is the relationship that has been described so far: an active object has an associated observer which receives state changes.
- *1-to-n*: if the observer is an *event channel*, other objects can be subscribed to the channel and receive state changes. Event channels may be federated and *linked* to others[3].
- *n-to-1*: several active objects may share a single observer, so that all state changes will be received by the same observer (useful for logging purposes). Note that in this case, the observer may be a channel also, and then the relation will be *n-to-n*.

Besides state change, an active object may use different ways to notify observer objects:

- *Asynchronously*. Active objects send notifications when the related magnitude changes. For the *ActiveStatus*, that occurs when an electric load is connected/disconnected (*setUsed()* method) or the node is enabled/disabled (*setEnable()* method). The latter is only applicable for actuators, as we explained above. This mechanism is of course fully asynchronous and it just requires a message.
  For the *ActiveMeter*, a notification for each change is definitely inefficient for continuous magnitudes. In that case, an additional mechanism may be implemented to specify *conditional* notifications. For each magnitude users may define a *range*, a *delta* or both. Range specifies the values that the magnitude must fit in order to trigger a state update notification. Delta indicates the minimum difference between current and last sent value to generate a new update message.
  Conditional notifications should be set up locally, that is, as part of the program code of the node.
- *Periodically*. Measured values may be updated and transmitted at certain programmed period. Typically, most high level applications do not require large amounts of data continuously. In these scenarios, periodically and time programmed requests may reduce the network load.

[3] Ice middleware provides a built-in event channel service, so-called *IceStorm* [13], that we use in our prototypes.

- *On-demand*: clients may cause state notification on demand by means of a common method. For instance, any remote object implements the *Object* interface that provides a *ping-like* method. It is used by clients to check object reachability. In our implementation, when an active object receives such invocation it transmits their current state values to the configured observer. Such methods may be found in environments based on CORBA or CORBA-like frameworks.

The most valuable feature of CoSGrid platform is the active objects. In section VIII an application example is shown using a distributed smart grid environment.

C. Object and data aggregation

Power grids are large scale systems. Suppose a smart grid with hundreds or thousands of sensors and each sensor sending its measured values to a processor data node. This scenario is not scalable. Thousands of sensors will cause a *storm* of messages through the network that would interfere with the correct operation of other applications.

For these situations, CoSGrid provides interfaces for composite objects. A composite object is inspired in the software design pattern with the same name [17]: a composite object *is-an* object that knows other objects (that in turn can be composites) and the whole set behaves as one of them (it exposes the same interface). In CosGRid, a composite object is a virtual sensor/actuator object that contains references (proxies) to remote objects that may be grouped by arbitrary criteria. Furthermore, composite objects may be used for aggregating data. For instance, a composite *BasicMeter* sensor would provide the average value of its associated objects.

The CosGrid Composite module provides the following set of interfaces (simplified code for readability):

```
module CosGrid {
    interface Component {
        Ice::StringSeq getAllFacets();
    };
    module Container {
        interface RW;
        interface R { ObjectPrxDict list(); };
        interface W {
            void link(string key, Object* value);
            void unlink(string key);
            Container::RW* create(string key);
            void destroy();
        };
        interface RW extends R,W {};
    };
    module Composite {
        const string MIN = "minimum"; // lowest value
        const string MAX = "maximum"; // highest value
        const string AVG = "average"; // sum(0..n)/n
        const string MED = "median"; // sort(0..n)[n/2]
        const string HEI = "height"; // max-min
    …
        const string ANY = "any"; // true if any true
        const string ALL = "all"; // true if all true
        interface R extends Container::R, Component {};
        interface W extends Container::RW, Component {};
        interface Factory {
            Object* create(string scalarType);
            void destroy(Object* proxy);
            Ice::StringSeq getAllowedTypes();
        };
    };
};
```

The composite object implements the *Container* interface (to hold remote object references) and the *Component* interfaces (to provide *facets* browsing). Each facet may be used to offer different aggregation operator. For instance, a composite sensor would provide both *average* and *maximum* value of the electric power consumed by using different facets with these names. Thus, *Component* interface lets the clients know which facets the object has.

Composite objects, as containers, may be *readable* (interface *R*) if clients are only able to get their references but not change them. If modification is required, composite objects should be *writable* (interface *W*). A *Factory* interface to create and destroy composite objects is also provided.

Note that composite objects may also be active, that is, when the aggregated value changes (with the same considerations explained in section VI-B) the composite send an update to its configured observer.

All these interfaces may be mixed (interface inheritance) to specialize behavior. Fig. 4 shows an example of class diagram of a CoSGrid composite object that clarifies which part of the total functionality contribute each interface.

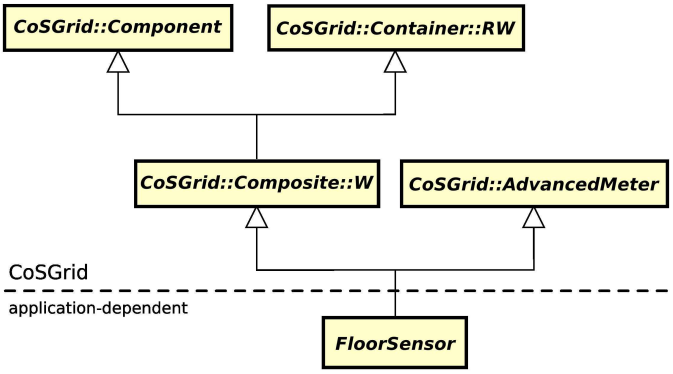

Fig. 4. Example of user interface definition for a composite object.

A *FloorSensor* would provide statistical measured values (through facets) from all sensors of an entire floor. Thus, the interface *FloorSensor* extends from *AdvancedMeter* and *Composite::W*, so that clients can retrieve values from associated sensors and reconfigure in execution time which sensor are part of the composite object.

VII. Deployment and Configuration

After the physical deployment of the EMDs has been done, the system administrator needs to identify and associate each object with the electric load it will monitor and/or control. We designed a service discovery protocol suitable to identify every node in the environment [18]. The node can periodically send

asynchronous messages to advertise itself and its features. With the advertisement information and an administration software tool, it is possible to add user or application specific information to each node, such as the location in the building, human readable description, etc. All of this information is propagated up in the hierarchy when required.

For security and privacy reasons, the access to the system follows a role-based schema. The visibility and access privileges depend on each actor role; a house owner may see and access all of the devices that control his appliances. Furthermore, middlewares usually provide security at protocol level by using SSL/TSL encrypted communications.

VIII. SCENARIO

This section describes an example scenario where high-level applications can be implemented over CoSGrid platform to provide smart services using reconfiguration and aggregation features. A schematic representation of such scenario is shown in Fig. 5.

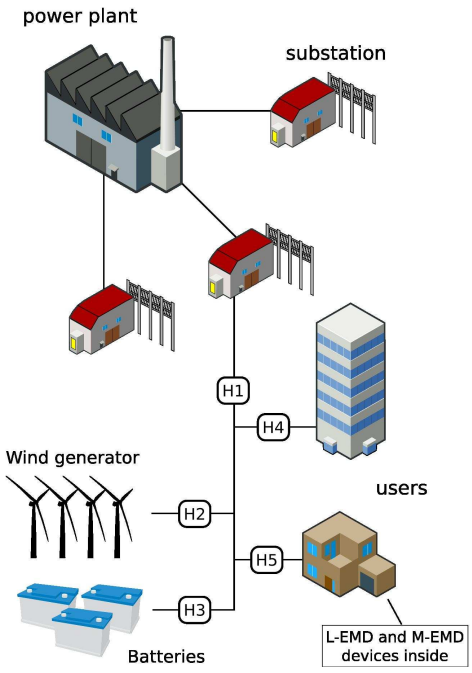

Fig. 5. Example scenario of a power grid where CosGrid devices have been installed.

User buildings may represent complex hospitals or business buildings that need generators in case of power blackout. Batteries (or other way to accumulate energy) may be charged using wind generator or main power, and the remaining energy is inserted into the grid.

All of these infrastructures are provided with L-EMD (or multi L-EMD) and M-EMD devices for electric appliance and floor electrical panels, respectively. Thus, fine grain control and measurement of consumption or generation values can be done. L-EMD devices can monitor appliances like computers, lights, electrical sockets, etc. M-EMD devices are composites of L-EMD devices and control an entire floor (or a small building). Using the composite interface, clients can get statistical values from the associated L-EMD devices. However, if it is required by the application, each L-EMD device may be individually inspected.

H-EMD devices are also represented in the figure above, and typically monitor an installation or building:

- *H1*: controls and measures the power line from substation. It is a composite object which provides statistical values from the other H-EMD nodes (there is a two level hierarchy of H-EMD).
- *H2*: controls and monitors the generated energy from the wind generator. It is also a composite of EMD devices of the generator system. Unlike the rest, it is expected that this EMD will provide negative consumption values due to energy generation.
- *H3*: this EMD just controls whether the batteries system should start providing power or they should stay disconnected.
- *H4* and *H5*: monitor and control their respective buildings and both are composite objects.

By using composite and active objects, smart agents or services (built on top of this platform) can reconfigure the logical architecture under different situations described below.

A. *Billing and measuring*

The most basic capabilities of the Smart Grid (the energy accounting and billing) may be directly obtained from the state of EMDs. In addition, the platform provides some other features along this line:

- It allows the provider to read the whole energy consumed (or generated) by each user.
- It is possible to summarize in real time the combined expenditure of all users (taken from the corresponding composite object facet) and compare it with the actual measure for the whole area. In this way it is possible to detect and localize installation or maintenance problems, sensor defects, frauds, billing errors, etc.
- It allows the user to know his expenditure immediately, and which appliances are responsible of it.

B. *Flexible and automated deployment*

New appliances may be deployed at buildings. By using the service discovery protocol (see section VII) and M-EMD and H-EMD composite objects, agents can group automatically new appliances, so that they may be monitored and controlled immediately.

By using a graphical interface, the administrator can remove, edit and add new properties and meta-information which describes location, features description and so on. This information may be used by agents to provide more valuable functionality like automated service composition [19].

C. Power blackout

Suppose H1 detects that a substation stops supplying electric power due to a certain failure. If H1 is an active status object, a conditional notification may be programmed in this case: *enable the observer (setEnable(True)) if electric current falls to zero, otherwise disable it*. If H3 is the observer of H1, when a power blackout occurs, the batteries system will be activated to supply electric power.

Agents may change observer object at execution time, so that the same active object will notify state changes to other observer. Depending on grid requirements and its state, smart agents can reconfigure active objects so that the batteries systems supply electric power to the right grid zones. A batteries system which originally was not designed as a backup of a concrete infrastructure may be used by others by simply modifying observer references.

On the other side, in case of a power blackout, certain appliances can be shutdown without major inconvenience but some others are critical. All of the devices in an operating room or staff elevators in a hospital are examples of critical appliances.

We propose a flexible architecture for agents that can manage this situation: every non-critical appliance is subscribed to the *non_critical* channel. This channel is the observer of an active object which detects the electric power failure. In such case, active object will send a *setEnable(False)* to channel, and the message will be retransmitted to subscribers, turning off all of the appliances to reduce electric power consumption.

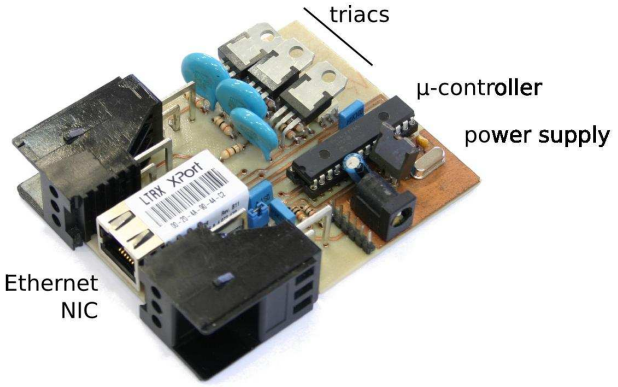

Fig. 6. Multi L-EMD prototype

The valuable contribution of this solution is that it is reconfigurable. Suppose the hospital building is restructured and where there was an operating room, now there is a seating room for the staff. Electric sockets and all of the appliances installed into it are now non-critical. Just subscribe them into the *non_critical* channel and they change their role immediately and without any operational action.

IX. PROTOTYPE

Currently, we have built a prototype using a CORBA-like middleware and the picoObject concept. Fig. 6 shows our multi L-EMD prototype with the following hardware components:

- Embedded Ethernet network interface. It includes an implementation of TCP/IP protocol.
- 8-bit microcontroller with 14.3 KB for program memory, 368 bytes of RAM size and 256 bytes for ROM memory.
- Three triacs for power control switching.

The distributed objects for the middleware that are built into this kind of devices (L-EMD and some M-EMD devices) are be implemented using the IcePick framework [20]. This framework provides a high-level programming language to specify the distributed scenario, that is, the logical distribution of distributed objects into nodes through the distributed system. Then, an automated tool builds all required picoObjects in a low-level specification language (*bytecode*). The bytecode should be executed by a given virtual machine. We have built different virtual machines for several platforms: PC (assembler, Python, C and others) and embedded operating systems [21] IcePick framework also provides tools for automated installation of both bytecode and virtual machine in the final hardware nodes.

There are three distributed objects installed in our multi L-EMD device. Each object implements the controlling, status information and active interfaces of CoSGrid, so that each object implements *Control*, *Status* and *ActiveStatus* interfaces. Currently, this prototype does not include hardware for measuring purpose, so that it does not provide sensing services. Using optimizations provided by IcePick, this prototype has the following virtual machine requirements:

- 126 bytes for ROM memory.
- 455 bytes of bytecode length.
- 21 bytes for RAM memory.
- 60 bytes for flash memory.

Thus, this prototype requires 662 bytes at virtual machine level. To get the final footprint, the resources required for the virtual machine should be added, but these values depend on the concrete implementation. For instance, a virtual machine implementation in assembly for the microcontroller platform with embedded Ethernet interface (as in the prototype) requires 2643 bytes of program memory and 65 bytes of RAM memory.

X. CONCLUSIONS

The smart grid presents many common aspects with distributed systems. Heterogeneity and scalability are key problems and may be solved by the core platform. OOM seems to be directly applicable but that is only a part of the problem. We proposed the use of distributed object paradigm, but defining a concrete information model that allows concreting communication mechanisms and the rules that services must fit to gain interoperability and take advantage of the middleware features. We also provide a very low cost

implementation of an autonomous and reliable distributed object; that makes it possible to interact with any components of the grid easily.

By means of logic relations (built on active objects and observer concepts) we provide and on-the-fly self-contained reconfigurable event architecture that greatly simplify the deploying and management of reactive services in the grid.

As future work, we are working on prototypes that integrate measurement capabilities and improved management features. From the company point of view, we are interested in services that can make automatic data aggregation and data logging to detect potential grid problems, sabotage, etc. Other goals include high abstraction level services (agent based) that analyze user behavior from their activities related to home appliances and in this way detect anomalous situations, accidents, security problems, etc.

REFERENCES

[1] R. Hassan and G. Radman, "Survey on Smart Grid," in Proceedings of the IEEE SoutheastCon 2010 (SoutheastCon). IEEE, 2010.

[2] M. McGranaghan, D. Von Dollen, P. Myrda, and E. Gunther, "Utility experience with developing a smart grid roadmap," in *Power and Energy Society General Meeting - Conversion and Delivery of Electrical Energy in the 21st Century*, IEEE, pp. 1–5, July 2008.

[3] D. P. Bernardon, V. J. Garcia, A. S. Ferreira, and L. N. Canha, "Multicriteria Distribution Network Reconfiguration Considering Subtransmission Analysis," *IEEE Transactions on Power Delivery*, vol. 25, no. 4, pp. 2684–2691, October 2010.

[4] S. Massoud Amin and B. F. Wollenberg, "Toward a smart grid: power delivery for the 21st century," *IEEE Power and Energy Magazine*, vol. 3, no. 5, pp. 34–41, September 2005.

[5] F. Meng, R. Akella, M. L. Crow, and B. McMillin, "Distributed Grid Intelligence for future microgrid with renewable sources and storage," *North American Power Symposium (NAPS)*, pp. 1–6, September 2010.

[6] S. Chowdhury, S. Chowdhury, and P. Crossley, "Microgrids and Active Distribution Networks". *Institution of Engineering and Technology*, 2009.

[7] R. Akella, F. Meng, D. Ditch, B. McMillin, and M. Crow, "Distributed Power Balancing for the FREEDM System", *Smart Grid Communications (SmartGridComm), 2010 First IEEE International Conference on*, pp. 7–12, October 2010.

[8] H. Y. Tung, K. F. Tsang, and K. L. Lam, "ZigBee sensor network for Advanced Metering Infrastructure," *in 2010 Digest of Technical Papers International Conference on Consumer Electronics*. IEEE, 2010.

[9] S. Chen, J. Lukkien, and L. Zhang, "Service-oriented Advanced Metering Infrastructure for Smart Grids," *in 2010 Asia-Pacific Power and Energy Engineering Conference*. IEEE, 2010.

[10] K. H. Han, S. W. Choi, B. C. Park, and J. J. Lee, "An implementation of a wireless sensor network-based meter reading system," *in SenSys'09: Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. New York, NY, USA: ACM, 2009.

[11] M. Pipattanasomporn, H. Feroze, and S. Rahman, "Multi-agent systems in a distributed smart grid: Design and implementation," *in 2009 IEEE/PES Power Systems Conference and Exposition*. IEEE, 2009.

[12] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, 3rd ed., 2002.

[13] M. Henning and M. Spruiell, *Distributed Programming with Ice*, Revision 3.4.1, 2010.

[14] Sun Microsystems Inc., *Java Remote Method Invocation*, 2006.

[15] F. Moya, D. Villa, F. J. Villanueva, J. Barba, F. Rincón, and J. C. López, "Embedding standard distributed object-oriented middlewares in wireless sensor networks," *Journal on Wireless Communications and Mobile Computing*, vol. 9, no. 3, 2009.

[16] S. Herman, *Alternating Current Fundamentals*, 7th ed., T. D. Learning, Ed., 2006.

[17] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, 1995.

[18] F. J. Villanueva, D. Villa, M. J. Santofimia, F. Moya, and J. C. López, "A framework for advanced home service design and management". *Consumer Electronics, Transactions on,*: IEEE Computer Society, 2009.

[19] M. J. Santofimia, F. Moya, F. J. Villanueva, D. Villa, and J. C. López, "An agent-based approach towards automatic service composition in ambient intelligence," *Artif. Intell. Rev.*, vol. 29, pp. 265–276, 2008.

[20] C. Martín, D. Villa, F. Villanueva, F. Moya, M. Santofimia, and J. López, "A development model supporting integrative object oriented middlewares for sensor and actuator networks," *in ICWN '10: Proceedings of The 2010 International Conference on Wireless Networks*. CSREA Press, 2010.

[21] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, and D. Culler, "TinyOS: An Operating System for Sensor Networks," *in Ambient Intelligence*, W. Weber, J. M. Rabaey, and E. Aarts, Eds. Berlin/Heidelberg: Springer-Verlag, ch. 7, pp. 115–148, 2005.

BIOGRAPHIES

David Villa received the Computer Eng. Diploma from the University of Castilla-La Mancha (UCLM) in 2002. Since then he works as a Teaching Assistant at the UCLM. In 2009, he obtained the PhD degree in Computer Science at the UCLM. His current research interests include heterogeneous distributed systems, distributed embedded system design and virtual network routing protocols.

Cleto Martín received the Computer Eng. Diploma from the University of Castilla-La Mancha (UCLM) in 2010. Since then he works as a Technologist and Researching Assistant at ARCO Research Lab. He is currently a Master degree student. His current research interests include heterogeneous distributed systems, virtual network and QoS–based routing protocols.

Félix Jesús Villanueva received the Computer Eng. Diploma from the University of Castilla-La Mancha (UCLM) in 2001. In 2009 he obtained the PhD degree from the UCLM, where he is now working as Teaching Assistant. His research interests include wireless sensor networks, ambient intelligence and embedded systems.

Francisco Moya (received his MS and PhD degrees in Telecommunication Engineering from the Technical University of Madrid (UPM), Spain, in 1996 and 2003 respectively. From 1999 he works as an Assistant Professor at the University of Castilla-La Mancha (UCLM). His current research interests include heterogeneous distributed systems and networks, electronic design automation, and its applications to large-scale services and system-on-chip design.

Juan Carlos López (M'76) received the MS and Ph.D. degrees in Telecommunication (Electrical) Engineering from the Technical University of Madrid in 1985 and 1989, respectively. From September 1990 to August 1992, he was a Visiting Scientist in the Department of Electrical and Computer Engineering at Carnegie Mellon University, Pittsburgh, PA (USA). His research activities center on embedded system design, distributed computing and advanced communication services. From 1989 to 1999, he has been an Associate Professor of the Department of Electrical Engineering at the Technical University of Madrid. Currently, Dr. LÃ³pez is a Professor of Computer Architecture at the University of Castilla-La Mancha where he has served as Dean of the School of Computer Science from 2000 to 2008. He is and has been member of different panels of the Spanish National Science Foundation and the Spanish Ministry of Education and Science, regarding the Information Technologies research programs. He is member of the IEEE and the ACM.