

GRENAD, a Modular and Generic Smart-Grid Framework

Sylvain Ductor, Jesus-Javier Gil-Quijano, Nicolas Stefanovitch, Pierrick Roger Mele
CEA-LIST, Département Métrologie Instrumentation et Information (DM2I),
Laboratoire d'Analyse de Données et Intelligence des Systèmes (LADIS),
Digiteo lab - bat 565 - Point courrier 192,
91191 Gif-sur-Yvette Cedex - France Bâtiment 425,
Email: {surname.name}@cea.fr

I. INTRODUCTION

Abstract—We present in this paper GRENAD, a Multi-Agent System based framework for the simulation and piloting of power-grids and particularly smart grids. Exploiting a component-based approach, it allows a flexible design of complex smart grid applications by providing a generic canvas where extensible, modular and reusable components, defined on the basis of their functionalities, can be easily combined and connected. Thanks to Multi-Agent approach, a set of such components can naturally be integrated into a coherent economical agent. GRENAD makes no assumption on the energy definition and eases the development of MAS control algorithms for smart grids. The level of details of the energy-related information is controllable. This information is computed either through internal physical models or by interfacing with external simulators. We present here our model, illustrate its features with a rich example which exhibits its genericity, and demonstrate how a coordination protocol can easily be integrated to it.

ENERGY supply is at the core economy while being also one of the major items of expenditure. Decreasing fossil energy supplies as well as rising concerns about climate made it critical to change in the near future the way energy is produced, distributed and consumed. Smart grids are highly automated power networks that possess fine grained monitoring and control capabilities, from the power plant to the domestic appliance. Easy access to information and autonomous decision making allow smart grids to save energy by quickly reacting to changes in the environment and reorganising demand, production and distribution. Smart grid are also meant to allow the massive integration of distributed renewable energy resources (DER) as well as new equipments such as combined heat and power (CHP) generation and energy storage. As such smart grids are called to play a crucial role in the coming years for the efficient control of power systems.

Hardware for production, monitoring and storage in smart grids already exist. While still not being widely deployed, the penetration of such equipments is constantly progressing and increasingly supported by legislations. However they raise new challenges. Indeed, as opposed to the traditional power grids approaches, energy flows may be intermittent and/or bidirectional. Moreover smart grids technologies also foster a radical change in the business of energy supply, by allowing the energy to be islanded, it is to say produced, sold and

consumed locally. Lastly, increased interconnection and information exchange between actors enables them to coordinate more closely, providing thus a new opportunity to reach higher efficiency and revenues.

One of the most challenging aspects of smart grids is thus at the engineering and at the logical level: the development of efficient control and coordination algorithms that are able to fully exploit their different potentialities. In order to tackle these challenges the use of a Multi-Agent Systems (MAS) approach is particularly relevant as the structure of MAS closely match the structure and behaviour of smart grids: They are constituted of a set of interconnected distributed autonomous actors, each aiming at maximising their respective goals through coordination. Also, a given smart grid is composed by interconnecting different components (production, distribution or storage) that may be used, as is, on another smart grid. It is thus relevant to adopt a modular approach for modeling the equipments in a reusable, and thus capitalizable, way. Therefore a key driver for the deployment of smart grids is the conception of a dedicated platform that ease their development by exploiting both oriented-component paradigm for equipment modelisation aspects and oriented-agent paradigm for coordination and optimisation aspects.

In the context of the Resilient FP7 project [20], we have developed such a platform, GRENAD, which stands for "Gestion des Ressources ENergétiques, Autonome et Distribuée" (Autonomous and Distributed Management of Energetic Resources). GRENAD presents two main aspects. The first is a feature rich API and Domain Specific Language (DSL) based on JADE [11] which allows to construct component-based JADE agents. As such it benefits of all the features and standard compliance of JADE and can natively be used in conjunction with other JADE-based applications. The second is a generic smart grid model implemented on top of this DSL. Thanks to these two aspects, it is possible to easily capitalize and reuse developed software across different smart grid applications.

While most MAS publications in the smart grid domain focus on the connection with a simulator or the conception of a new control algorithm, our contribution is a platform that offer enough flexibility to allow an easy interfacing with simulators and an easy implementation of control algorithms. Also, each

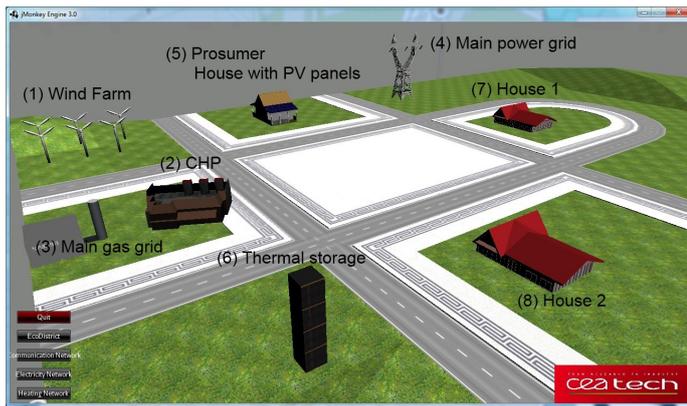


Fig. 1. A rich scenario

component defining the smart grid can natively be monitored, or even controlled, either by an agent, a GUI, or a remote optimisation algorithm.

In Section II we present a smart grid scenario exhibiting all critical features for the applications we are interested in. In Section III we present the related work and how they answer to the objectives highlighted by our scenario. In section IV we present the GRENAD agent and smart grid models and we describe how to implement the scenario with them. In Section V we show how to easily integrate a smart grid optimization algorithms in GRENAD.

II. A RICH SCENARIO

In this section we present a district energy system scenario. The richness of this scenario allows us to demonstrate the capabilities of our MAS based tool to model complex energy management systems at the district-level. This scenario exhibits most of the main characteristics that challenge today the design of smart grid energy management systems [19]:

- Interdependence of several energy flows (electricity, heat and gas)
- Distributed generation and storage capabilities
- Interaction with the main energy grids (electricity and gas)
- Local renewable energy generation capabilities
- Prosumers
- Diverse consumption profiles

In the scenario, the considered components (producers, distribution grid, consumers/prosumers and storage systems) are modelled as autonomous agents that interact with the intention of providing real time management at the district level and optimizing the energy production and distribution in the district, in conjunction with the main energy networks.

A. Business and organizational models

The behaviors of the different components are defined by their individual roles and their internal characteristics (physical, economical, quality of service, *etc.*). Those behaviors are constrained by the business and organizational models of the system as well as by the interconnection mode to the

main grid (an energy system can work either in *connected* or *islanded mode* w.r.t the main energy grids). In our scenario we considered a system connected to the main gas and power grids (represented as components 3 and 4 in Figure 1).

Among the different existing business and organizational models we can list:

- **Free markets:** the components are competitors and interact via a market mechanism [12];
- **Virtual Power Plant (VPP):** combination of production storage and consumption resources. A VPP [18] is adapted when all the resources belong to or are managed by a single actor;
- **Cooperative Virtual Power Plant (CVPP):** In a CVPP [3] different actors manage/own the different components, they coordinate in a cooperative way to use shared resources (for instance centralized storage systems, the distribution network, *etc.*) and provide global services (for instance: grant the energy balance, the security of the network and/or the quality of service).

All these different organizational models can be considered in our approach. In our scenario we model interactions between components as a free market.

B. Roles

The main roles that we support are *producer, storage, consumer, prosumer and distribution*. The different constraints and characteristics associated to those roles are described below:

1) *Producer components:* The characteristics of the generation of energy provided by producers depend on several aspects:

- **Internal physical constraints:** the nominal power (max power capabilities), start/stop conditions (time to be operational/off operation), their efficiency (it is the rate between the output generated energy and the input primary used energy), and for renewables the variability and the intermittence.
- **Controllability:** The controllability of a given generator depends on the primary energy source that it uses and on internal state variables. Most of renewable-energy sources (e.g. wind, solar) are considered non-controllable, it is, their availability and level of energy cannot be controlled. Nevertheless, some renewable based generators can adapt their internal state (i.e. orientation of wind turbines or solar trackers) in order to secure and optimally generate energy according to the primary-source real time conditions. Generators whose primary energy supply can be controlled provide different control levels, for instance some generators can only provide on/off control while others can provide intermediate levels of functioning.

In our scenario we consider two producers (the components 1 and 2 in Figure 1): a set of wind turbines for power generation and a Combined Heating and Power (CHP) generator that generates both power and heating from combustion of gas. Gas is provided by the main gas grid (component 3 in

Figure 1) and can be stored nearby the CHP. The system is also connected to the main power grid (component 4 in Figure 1). The gas and the power main grids are considered in our model as controllable producers, it is, we consider that the quantity of gas or the power drawn from the main grids can be controlled. In the real system only the gas flow can be directly controlled, the drawn power from the main grid is indirectly controlled and corresponds to balance needed power, it is, the consumed power minus the local producer power.

2) *Storage components*: The behavior of a storage component depends mainly on its nominal capacity, its rate of charge/discharge and its instantaneous state of charge (SoC). Other important variables are the efficiency of the conversion in the charge and discharge phases and the self-discharge rates. The usage of some storage systems, for instance batteries, is constrained by the charge/discharge cycling conditions that optimize their efficiency and increase their lifetime. In our scenario we consider a thermal storage component (see component 6 in Figure 1).

3) *Distribution components*: The distribution of energy generates losses due to the physical characteristics of the transportation system used (i.e. transmission capabilities, dissipation rates, etc.) and the distances of transmission of the energy. In general, at the district level, the electricity losses due to distribution are negligible while the heating and hydraulic (heating network related) losses are not. Some distribution components (as valves in heating networks and some types of transformers) allow the dispatching of input energy among different outputs. While the availability of energy over electrical networks is instantaneous (electricity is transmitted at about the speed of light), the transmission of energy over heating and cooling networks, due to the transport inertia, needs significant time (several minutes per km) to go from the injection to the consumption points. In our scenario we consider local heating and power networks: all the production and consumption components considered in the scenario are connected to at least one of these distribution networks.

4) *Consumer components*: The characteristics of the energy consumed, basically their load curves, depend mainly on the buildings physical characteristics (i.e. thermal inertia, storage capabilities), the behaviors of the inhabitants (presence/absence, used services) and the comfort constraints (e.g. ambient temperature set points). In our scenario we consider two consumers, the components 7 and 8 in Figure 1.

5) *Prosumer components*: Besides the characteristics that define the load curves of the consumer components, prosumers can partially or totally cover their needs in energy and under some conditions produce energy surplus that can be injected into the main or local energy grids. In our scenario we consider one prosumer (component number 5 in figure 1), that locally produces power thanks to the PV solar panels installed on its roof. It uses part of this energy for self-consumption and is able to inject part of this energy into the power distribution network.

C. Planing and operation

Due to the temporal inertia of most of the components (e.g. time to start or change generator command level; thermal inertia on houses and distribution networks; time to charge/discharge storage systems) and the use of non-controllable sources (that are intermittent and variable), the energy usage needs to be planned before actual generation and delivery. In our approach, we consider *planning strategies* that are combined to *real-time operation strategies*. Planning is based on estimation of the consumption loads and generation capabilities and allows to establish negotiated (i.e. agreed by all parties) generation, storage and consumption schedules. For the renewable based generators, the estimation depends on weather and generation capabilities forecasting. For end consumers, the estimation depends mainly on weather and usage forecasting, as well as on thermal inertia evaluation. Those different forecasting based estimations lead to inaccuracies at the planning phase. In order to maintain the balance between consumption and generation and minimize the use of external generated energy, we implement and operate a mechanism that allows the near real-time correction of energy schedules when deviations of generation and load are detected (monitored or forecasted). This mechanism can be seen as a capacity market where global flexibility is provided by the combined individual flexibility capabilities (e.g. deferrable load, dedicated storage capabilities, etc.)

D. Real-Time payment

Smart-Grid also aims to adapt the consumption to the energy production capacity. This is particularly critical when some producers are not controllable (e.g. weather dependant renewable energy source). By exploiting for instance a building thermal inertia, it is possible to respect a required comfort level while adapting the actual consumption.

A commitment of the consumer on how it will consume energy allows to optimise energy production and distribution during the planning. Several payment approaches [22] propose to impose penalties if such a commitment is not respected during operation phase. The actual payment of a consumer is thus computed by comparing its planned and operational consumption along with some other production-related and/or conventional parameters.

III. RELATED WORKS

Actual testing and deployment of smart grid software require platforms able to directly support the execution of control software developed following as well as the ability simulate or even pilot power systems. Among the few available commercial systems that target specifically smart grids, only a bit of them follow the MAS approach.

The available commercial software that simulate and control traditional grids [15], [7], can to some extent be used to simulate parts of a smart grid, notably for the distribution aspect. However, smart grids possess several specificities that preclude the use of such software for the full chain control. On the other hand, smart grid software have not reached

this level of maturity and are mostly experimental or at the level of academic research. Dedicated software solutions for smart grids exist independently, for different aspects presented in this scenario: storage simulation and dimensioning [10], DER integration [5], demand side management [9] and VPP [2]. While none of these commercial software uses a MAS approach, PowerMatcher [17] is an exception. It can perform simulation and piloting and uses auction protocols to interact over a market of flexibilities.

Among non commercial software, GridLab-D [4] is a notable one, it is a low level electrical simulation tool based on a MAS paradigm. It allows to assign different profiles of consumption to the devices on the network, but fails to represent economic actors and is not able to endow agents with advanced smart behaviours that react to change in their environment.

Most academic works in smart grids consider separately the development of a platform, a model and control algorithms. As such they provide very limited reusability and interoperability between the different layers. Platforms in the literature mostly act as a middleware and focus on the coupling of a MAS platform (JADE [11] being the most popular) with an electrical simulator and data exchange formats[1], [23], [21]. The designs provided in papers that present models [16], [8] fail to acknowledge either the simulation or the control aspects and often both of them, limiting their applicability. Finally, while most control algorithms are backed by experimental simulations, they are coded specifically for non smart grid environments and therefore fail to be reused and extended. We believe this shows a clear need for a generic and flexible tool for smart grid algorithms conception, testing and deployment.

In this paper we propose such a tool: a platform and a model that have been thought for reusability in smart grid applications and aims at supporting different smart grid applications at all levels. We demonstrate this by exhibiting the coupling of a state of the art distributed control algorithm. Previously mentioned works lack an abstract representation encompassing both the socio-economical actors and the physical devices, in GRENAD both are explicitly modeled.

The implementation of GRENAD is based on JADE. JADE is a widely popular generic purpose MAS execution platform providing primitives to ease the development of agents and deployment of agents [11]. This platform, by being generic, lacks support for smart grid elements needed to represent, simulate or even control a smart grid. GRENAD extends JADE and enrich its capabilities with additional communication mechanisms and smart grids specific objects and proposes a component based architecture.

IV. OUR MODULAR AGENT-BASED SMART-GRID MODEL

In this section we present and illustrate our contribution: an agent model implemented on top of JADE and a smart-grid model implemented on top of our agent model. The conception has been driven by the Design Pattern approach. For more information about Design Patterns please have a look at [6].

We first present our agent model, then we present our smart-grid model and last we describe how to formalize the scenario thanks to this model.

A. Agent Model

GRENAD is implemented as an overlay of JADE. It provides a Domain Specific Language (DSL) that exploits a component-based approach in order to describe the agents and enhance JADE with a set of native services (agent building system, ergonomic agent messaging service, generalized support of a publish/subscribe system, ...). The agent building system simplifies the use of JADE for defining agents at compile time and runtime (initialisation step). The first class to understand in the agent building system is `GrenadAgent`. A `GrenadAgent<ID extends GrenadIdentifier>` can be viewed as a pair made of an **identifier** of type `Id` and a list of **components**. It is an implementation of the *builder pattern* from [6, p. 97]: the identifiers and components hold all the information required to build a JADE `Agent` exposing all the natively proposed features of JADE as well as non natively proposed ones.

A `GrenadIdentifier` is the specification of a JADE `AID` (the JADE `Agent Identifier`): `GrenadIdentifier` allows to obtain the `AID` once the JADE agent is set up by JADE¹. A component is a stateful and active object that is executed within the environment of the hosting agent. Such an environment provides access to a set of services shared by all the components of the same agent. We distinguish three groups of components:

- **Action components** are components that produce a list of JADE `Behaviour` objects that are to be added in the to-be-constructed JADE `Agent`.
- **Runtime components** allow to modify the way a `GrenadAgent` is handled by JADE. They can modify the `GrenadAgent` or execute side effects that depend on it, at agent set up or take down. They can specify actions to realise while cloning or moving the agent. They can also catch and handle exceptions issued by the agent execution, which is a functionality not natively proposed by JADE.
- **Composite components** are collections of components (as such they follow the *composite pattern* from [6, p. 163]). They allow to manipulate a bag of components as if it was a single one. From the agent point of view, loading a `Composite` component is equivalent to loading each component of the collection it represents.

By definition a component is an independent piece of software: it has its own state (set of constants, variables and methods) and has no access to the other components of the hosting agent (except for certain Runtime Components, as it is their goal). Certain actions allow to specify their state in a separate class and thus easily share it with other components

¹JADE agent life cycle starts by building the agent, then executes the method `setup` once the agent is alive. It then executes the method `takeDown` if the agent is required to terminate, and finally kills it

of the same agent. The goal is to enhance component re-usability by separating configuration/decision aspects to action/communication ones. For example, different behaviours or protocols may require at some point to have access to the agent launch date, its neighbourhood, its preferences or abilities such as computing an optimal path. Exploiting the *Template Method Pattern* from [6, p. 325] allows to identify those methods that ought to be placed in interfaces implemented by the component state. It is then the responsibility of the agent to globally implement them in states shared by its components, thus ensuring both coherence of its actions and efficient factorisation of its code.

Let us consider, for instance, an auction about a certain type of objects that runs between an auctioneer and several bidders. In this process each bidder is characterized by the value it attributes to the objects. Except for this *decision function*, all other actions executed during this auction process are conventions specified by the auction rules. A correctly factorized implementation would thus define a component that encodes the behaviours resulting from those rules and exploits the evaluation function on the considered bidders, implemented in a separated state. Let us suppose now that one of these same bidders has latter participates in a different kind of auctions about the same type of objects. The expected behaviour for this agent is to evaluate the objects in the same way. Our "correctly factorized" approach does it naturally since the agent will only need to share the previously used state with the component executing this new process.

Please note that we strongly recommend to avoid data mutability in this approach by only sharing constants and methods. If the state has variables, it is up to the programmer to ensure, when building the agent, that the different components that exploit it do not modify them in an incoherent way.

Besides, all the components of a same agent share three services :

- Access to the unique identifier of the agent.
- Access to the agent communication service, which allows to send messages and have access to the mailbox.
- Access to the publish/subscribe service of the agent.

The *Publish/Subscribe Pattern* (also known as the *Observer Pattern*, see [6, p. 293]) is natively proposed by GRENAD, because it provides an efficient way to decouple communication, as opposed to the natural approach of message sending. In the *Publish/Subscribe Pattern*, an agent, the publisher, publishes whenever it decides any information it finds relevant. It does not know *a priori* who is interested by this information. A published information is identified by its type, a name set by the publisher, and the identifier of the publisher. A subscriber agent knows *a priori* which publication of which publisher it is interested in. It thus subscribes to it, and indicates at the same time the action it will execute upon the reception of a new publication (how it will *react*). It may stop this behaviour at any time by unsubscribing.

Simple direct message exchange and the *Publish/Subscribe Pattern* are opposed approaches to communication: in the former, the receiver of the message is decided by the sender, while

in the latter the sender of a message is decided by the receiver. Natively proposing both approaches brings a great flexibility in the way of establishing communication neighbourhoods. *Publish/Subscribe Pattern* is also well suited for the event-driven approach that characterizes reactive agents.

The approach proposed so far achieves modularity in the production of multi-agent systems by applying the *Builder Pattern* on a class that encapsulates its own strategy (*i.e.* the list of components), thus allowing to get rid of the "extends" keyword for varying the characteristics of the instantiated agent. Indeed the *Strategy Pattern*, by promoting encapsulation, is more suited than inheritance when it comes to flexibility, re-usability and modularity. Modularity comes from the fact that a component can informally be seen as a full and independent functionality. It can be assembled with other components in order to easily define a complex agent. Component independence is the first root of re-usability. The second root lies in the ability to delegate the responsibility of the configuration or decision methods to the agent rather than to the components. Hence, given a set of already implemented components, it is possible to build heterogeneous agents by varying their preferences, planning methods, *etc.* Lastly, flexibility is provided by the simultaneous native support of traditional mail-box based message sending and reactive *Publish/Subscribe Pattern*, which avoids imposing constraints upon the initial coupling of neighbours.

B. Smart-Grid Model

On the basis of the agent structure presented in section IV-A, we propose here a model that allows to describe, simulate and pilot complex smart-grids. In order to do so, it exploits physical models of atomic components as well as simulators of groups of components. It allows a fine-grained remote monitoring and control of the energy distribution given information about consumption, production and a "configuration" of each component (*i.e.* an instantiation of its actuators). Also, actuators can be remotely controlled, thus allowing the smart-grid to be controlled by a user interface (GUI, service web, ...) or an autonomous decision mechanism. Considered energy information is projected over time and includes generic multi-flow energy demand, generic offer characteristic and whether the offer can meet the demand at each point of the grid. Relevant physical model can simulate the propagation of blackouts or brownouts. Also several simulations can run simultaneously on the same smart-grid, made mutually dependent (*e.g.* planning and operation) and easily compared (*e.g.* real-time payment).

The model proposed here has been implemented in GRENAD as a collection of components (see Section IV-A). Hence it can naturally be integrated with other components and distributed over a network in order to be used in various situations. For instance, it can get its values and set its actuators from and to a physical smart-grid ; it can be used as an isolated simulator ; it can be used in conjunction with a coordination protocol ; encapsulating agents may interact

with real clients ; encapsulating agents may negotiate on stock market, ...

In this section we present the different aspects of our smart-grid model: how energy is represented by the different roles, how information about energy is stored and updated, how it is transmitted between the agents and projected over time.

1) *Energy and Roles:* We consider three roles.

- A **consumer** expresses a *demand* as an *amount* of energy on the different flows considered (e.g. heat, electricity, ...).
- A **producer** expresses an *offer*, as a function that, given a demand, returns whether it can be produced. If so, it returns also what would be the *Quality of Service* (QoS) (e.g. CO_2 impact (in $kgCO_2$), base cost (in euros), ...).
- A **distributor** distributes the energy among its neighbours. It associates with each consuming neighbour an offer and each producing neighbour a demand.

A given component has a specific representation of the energy. This representation is structured among two aspects: amount and QoS. No constraints at all are put on these objects: they are only defined w.r.t. the information required by the physical models and the simulators used in the considered application. However, some components require a manipulation function to be provided. For this reason, we consider a dedicated algebra over amounts or QoS. This algebra allows the component to manipulate energy information as a black box, thus ensuring genericity. For instance, let \mathcal{A} be an amount, $\oplus^{\mathcal{A}} : \mathcal{A} \times \mathcal{A} \mapsto \mathcal{A}$ is an additive operator over \mathcal{A} . It allows to compute the overall demand of two consumers. $\emptyset^{\mathcal{A}}$ represent a null amount, i.e. the neutral element of $\oplus^{\mathcal{A}}$. The same goes for QoS.

A $\text{Demand}\langle\mathcal{A}\rangle$ is a container of several flows of energy of type \mathcal{A} . We implements a flow as a class that extends the class \mathcal{A} . A $\text{Demand}\langle\mathcal{A}\rangle$ can be manipulated, given an aforementioned algebra over \mathcal{A} by simply applying it independently on each of its flow.

An offer $\text{Offer}\langle\mathcal{A}, \text{QoS}\rangle$ is a function from a $\text{Demand}\langle\mathcal{A}\rangle$ to an $\text{Optional}\langle\text{QoS}\rangle$. An $\text{Optional}\langle\text{QoS}\rangle$ can either be a value of type QoS , if the producer has the capacity to satisfy the demand, or be *empty* if it cannot.

2) *State, Fields and Internal Model:* The state of a smart-grid component is defined as a collection of a dedicated class of fields, $\text{SGField}\langle\mathcal{A}\rangle$. An $\text{SGField}\langle\mathcal{A}\rangle$ allows to handle asynchronicity by outputting four types of values: (1) an object of type \mathcal{A} if it has been correctly instantiated (in this case, the field is said to be *valued*), (2) `NotReady` if some information is still missing or some computation is still running on, (3) `IncoherentNeighbourhood` if an information received from some neighbour is incoherent and (4) `InvalidInternalState` if the instantiation of the state of the component is invalid w.r.t. the internal model. Note that we do not throw any exception, since the states 2, 3 and 4 can come from a delay of communication due to asynchronicity or distribution. For instance, if two neighbours are required to modify their configuration, the first that receives the request

may define an `IncoherentNeighbourhood` while it has not received the updated information of the other component.

We distinguish five types of SGField . The first, $\text{Fixed}\langle\mathcal{A}\rangle$, has a constant value and, thus, should always be valued. The second and third fields have dynamic values and exploit the Publish/Subscribe pattern: $\text{Output}\langle\mathcal{A}\rangle$ is an observable field that is observed by a $\text{Sensor}\langle\mathcal{A}\rangle$ field of a neighbour. Whenever a new value is set in an $\text{Output}\langle\mathcal{A}\rangle$, it is published. The neighbour's $\text{Sensor}\langle\mathcal{A}\rangle$, upon the reception of the publication, will trigger the occurrence of a new event within the component. This will typically result to the call of the internal model of the component, which may in turn update some outputs, thus propagating the information. The fourth field, $\text{Actuator}\langle\mathcal{A}\rangle$, is an $\text{Output}\langle\mathcal{A}\rangle$ that can be remotely controlled by the fifth field, $\text{Controller}\langle\mathcal{A}\rangle$, which is a $\text{Sensor}\langle\mathcal{A}\rangle$ that may send a request to implement a new value on the actuator it observes, thus modifying the configuration of the smart-grid.

Each class representing a smart-grid role (consumer, producer or distributor) is associated in one hand to a set of fixed, outputs, sensors and actuators (its state) and, in another hand, to an internal model. As stated above, the internal model is triggered upon the reception of a new event, either a new value observed by some sensors, or the modification of some actuators. When called, the internal model will read the value of the fixed fields, the sensors and the actuators and update the outputs. The internal model is an abstract method whose implementation is application specific and may rely on some simple Java models or complex external physical simulators.

3) *Links and Information Propagation:* In our smart-grid model, the neighbourhood relation is defined using outputs and sensors: each component associates an output to each of its neighbours. It also defines a sensor for each output that has been associated to him.

Sensors may transform the observed information. This allows to easily implement link losses or maximal capacity. Sensors may also change the type of the information. Hence, given a function from \mathcal{A} to \mathcal{B} , an $\text{Output}\langle\mathcal{A}\rangle$ can be observed by a $\text{Sensor}\langle\mathcal{B}\rangle$. This allows to easily connect heterogeneous components.

Note that each component is associated to a group. Only outputs and sensors of the same group can interact. This allows a same agent to load different instances of the same component, each being uniquely identified by its group. Thanks to this architecture, it is easy to do simultaneously the current *operation* phase and tomorrow *planning* phase on a same agent. The latter may easily compute its real-time payment by comparing the outputs of its *planning* and *operation* components.

The three roles (consumer, producer or distributor) rely on a strict neighbourhood architecture. Figure 2 presents the neighbourhood relation between the roles. Next to the role is represented the type of its output; each output is mirrored by a sensor of the neighbour. For clarity, we always consider in this article homogeneous components: they use the same energy representation, $(\mathcal{A}, \text{QoS})$.

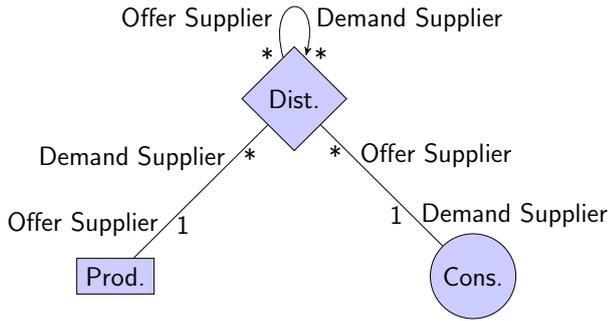


Fig. 2. Relation between roles

Consumers and producers are seen as clients of the distributing network. For this reason, they have exactly one neighbour, which is a distributor. They are responsible for defining either their demand (consumers) or their offer (producers), which is then observed by the distributor. A distributor may have several neighbours, either consumers or producers. The result of the distributed propagation of energy information is that every distributor outputs an offer to each of its neighbour consumer and a demand to each of its neighbouring producer. A distributor may also have several distributor neighbours. For each couple of distributor neighbours, one must be consuming and the other producing. The attribution of the role results from the internal models that may rely, for instance, on the value of actuators. Indeed, the distributor neighbour which consumes outputs a demand and observes an offer, and conversely for the producing one. If the neighbours disagree on their respective roles, they will output an `IncoherentState`. We refer to a distributor neighbour that has the consumer (rep. producer) role as a consumer (resp. producer) neighbour of a distributor. We refer to a distributor neighbour that is either the consumer (rep. producer) or a distributor that is consuming (resp. producing) as a consuming (resp. producing) neighbour of a distributor.

This model has been designed in order to perfectly separate the responsibility of each role. Consumer internal models are only concerned by the definition of the demand. Producer internal models are only concerned by the definition of the offer. Distributor internal models take as input the demand and the offer and are only concerned about their distribution. This abstract model is the basis of our components. Concrete classes require to implement specific internal models. Such models exploit specific actuators or outputs/sensors. Controllers allow a dynamic reconfiguration of actuators, by either a distributed or centralised algorithm, in order to optimize the energy distribution.

4) *Projection over time*: In Section IV-B2, we presented an `SGField<A>` as a container of either `A`, `NotReady`, `IncoherentNeighbourhood` or `IncoherentInternalState`. However, it is a more complex class. Indeed, it is a planning, *i.e.* a function, that, given a date, returns one of those four values or `Undefined` if no value is associated to the date. We propose several

implementations of the inner function model, which is set at construction time (see *strategy pattern*). For instance, a *constant* will return the same value for all the dates, a *scatter graph* is only defined on a discrete set of date, a *step function* is defined on a range of dates ... Many other models can be thought of, however, in order to be exploitable they have to implement the following set of methods.

Let us consider `FV<A>` (`FV` stands for Field Value), an abstract class that can only be instantiated as one of the five types of value that can be returned by an `SGField<A>`. An `SGField<A>` delegates from its inner function model²:

- `SGField` `map(Function2<Date, FV<A>, FV> f)`, a function that transforms a `SGField<A>` to a `SGField`, given a function that takes a date, a `FV<A>` and output a `FV`.
- `SGField<C>` `zipWith(SGField that, Function3<Date, FV<A>, FV, FV<C>> f)`, a function that aggregates a `SGField<A>` and a `SGField` into a `SGField<C>`, by applying a function on the couple of values associated to each date.

Both functions aim to provide required generic features, while preserving the fact that the inner model is a black box. `map` allows to modify the content of an `SGField`. For example, you can trigger a “maximum of capacity” for a planning of offers or “losses” on link transmission by mapping the appropriate function into the initial planning. `zipWith` allows to combine two `SGField` objects into one. For example, a planning of demand and a planning of offer can be combined into a planning of `Optional<QoS>`, which results, at each date, from the application of the demand to the offer. If the `f` parameter of `zipWith` is a binary operator (*i.e.* `A equals B equals C`), `zipWith` may be used to fuse a collection of `SGField<A>` into one `SGField<A>`.

Note that the `f` parameters of `zipWith` and `map` take the date as argument. This allows to implement time-dependent operations which is critical for any application that depends on the weather. For example the losses on a heat pipe may be dependent on the external temperature, which can be predicted with weather information.

Note that there exist several ways to simplify the `f` parameters of `zipWith` and `map`. First, if you do not consider a time-dependent application, you can build a function $f : Date \times A \rightarrow B$ from a function $f' : A \rightarrow B$ by simply returning the application of `A` to `f'` whatever the date is. Second, you can consider a conventional way of handling `NotReady`, `IncoherentNeighbourhood` and `InvalidInternalState` by order of criticality. If the mapped initial value (or one of the zipped value) is an `InvalidInternalState`, the resulting value is an `InvalidInternalState`. Or if it is an `IncoherentNeighbourhood` the resulting value is an `IncoherentNeighbourhood` since the internal state is valid but not w.r.t a neighbour internal state. If it is `NotReady` the resulting value is a `NotReady` since it means that there is

²this approach is inspired from the category pattern[13]

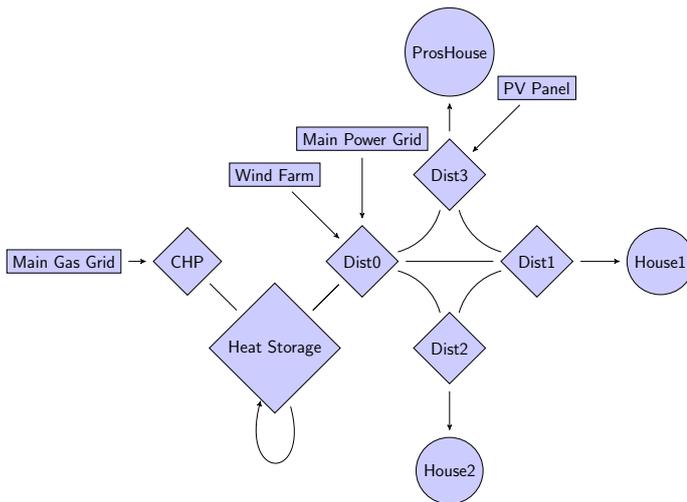


Fig. 3. Formalisation of Figure 1 scenario

a priori no error, but a computation is going on and we should wait for its completion. Hence, it is possible, using these rules, to automatically build a $\text{Function}\langle\text{FV}\langle\text{A}\rangle, \text{FV}\langle\text{B}\rangle\rangle$, from a $\text{Function}\langle\text{Optional}\langle\text{A}\rangle, \text{Optional}\langle\text{B}\rangle\rangle$ where $\text{Optional}\langle\text{A}\rangle$ is of type A if a value is associated to the considered date or of type empty if no value is associated (*i.e.* type Undefined). The third and last way of simplifying the f parameter is to consider that if the mapped value or one of the zipped value is Undefined, the result is Undefined. This allows to get rid of the Optional.

Hence, given those three simplifying rules, to manipulate an $\text{SGField}\langle\text{A}\rangle$ one can rely most of the time on $\text{SGField}\langle\text{B}\rangle \text{map}(\text{Function}\langle\text{A}, \text{B}\rangle f)$ and $\text{SGField}\langle\text{C}\rangle \text{zipWith}(\text{Function2}\langle\text{A}, \text{B}, \text{C}\rangle f)$ for time-independent application and on $\text{SGField}\langle\text{B}\rangle \text{map}(\text{Function2}\langle\text{Date}, \text{A}, \text{B}\rangle f)$ and $\text{SGField}\langle\text{C}\rangle \text{zipWith}(\text{Function3}\langle\text{Date}, \text{A}, \text{B}, \text{C}\rangle f)$ for time-dependent applications. The richness of our model allows however more fine-grained control on relevant cases.

C. Implementing the scenario

Figure 3 is the formalisation of the scenario of Figure 1 described in Section II. Figure 3 uses the same convention as Figure 2: producers are boxes, consumers circles and distributors diamonds. One can verify that Figure 3 is conform to the specification of Figure 2.

Economic actors are modelled using different components, each one characterising one functionality of the actor. Components House1 and House2 define the consumption of the elements 7 and 8 of Figure 1. They are respectively associated to Dist1 and Dist2, which define the quality of production of the energy they are supplied with. CHP and Heat Storage are distributors since they both consume and supply energy.

The links model interactions between actors; arrows are oriented from production to consumption. Both links and components are multiflow. There is a neighbourhood relation in form of a ring, the heating network, that goes along

Dist0, then Dist3 then Dist1 then Dist2 to go back to Dist0. Simultaneously, there is a tree shaped network, the electricity network, that connects the root, Dist0 to Dist1, Dist2 and Dist3. For instance, the link between Dist0 and Dist1 only transmits electricity information, the one between Dist3 and Dist1 only heating information and the one between Dist0 and Dist3 convey both. We choose to connect all the main power producers into a single distributor, Dist0, however, Dist0 may hide a complex subnetwork. Indeed, it is possible to fuse several distributors into one distributor. For instance, one could have modelled Figure 1 with only one distributor instead of Dist0, Dist1, Dist2 and Dist3 or even including Heat Storage and CHP. The same operation is feasible with consumers (or producers) connected to a same distributor. Please note that, in this case, one loses direct access to some information. For example, if one fuses every distributor into a single one, GRENAD will provide an easy access to all the information from and to the producers and consumers but not necessarily the information between the distributors. This feature allows to design the application with total control of the level of details. This is particularly useful for delegating the computation of the distribution of energy of certain local parts of the system to an external simulator.

We will now detail the implementation of the components.

1) *Consumers*: Consumers (*a.k.a* House1, House2 and ProsHouse) may be implemented in different ways. Each implementation should provide an output that defines a demand at any time and is aware of the neighbour distributor offer. Also, an enhanced consumer may implement a mechanism similar to the one described in Section V in order to adapt its consumption by coordinating, during the planning phase, with its distributor neighbour. Besides, thanks to the controller of a dedicated actuator, one can provide a total control to a remote end-user during the operation phase.

2) *Common distributor*: The distributors Dist0, Dist1, Dist2 and Dist3 have as sole function to distribute the energy coming from the producers to the consumers. The stability of a grid requires electricity to respects the Kirchoff law: the flow in must equal the flow out³. For this reason, we propose a generic distributor class, the *KirchoffStar*. This distributor is characterized by an actuator that indicates the part of the overall demand associated to each of the producing neighbours. Hence, this actuator holds a map that associates, to each producer and distributor neighbour, n , a number $p_n \in [0, 100]$. p_n is the percent of the sum of the consumer neighbour demands that the producing neighbour n has to satisfy. The actuator values, at each time point, is valid if it respects the Kirchoff law, that is to say the sum of the p_n equals 100. Note that if a distributor neighbour n is consuming, p_n will be null.

Reciprocally, we associate to each consumer the QoS that corresponds to the part of its consumption w.r.t. the overall consumption. For example, let us suppose that three consumers

³Note that as explained in Section IV-B3, losses, or any link related modification of the energy, are handled during the communication between an output and its sensor

with a demand expressed on the considered flow at the considered time point, as 3, 2 and 5 kWh. Let us suppose that there are two producers, the first assuming 30% of the consumption and the second 70%, as such, the first will assume 3 kWh and the second 7. Let us suppose that the first producer emits 4 $kgCO_2$ for producing 3 kWh and the second 6 for producing 7 kWh. Hence, the first consumer demand will be associated to a QoS of 3 $kgCO_2$, the second to a QoS of 2 $kgCO_2$ and the third to a QoS of 5 $kgCO_2$

However, our model requires the distributor to output not a QoS but an offer, that is to say, a function that associates to any demand a QoS, if this demand is satisfiable. The offer output to each consumer is computed under the hypothesis that it is the only one changing its demand. Let $prod_{dist}$ be the offer function associated of a given distributor, $dist$. A given demand a is distributed by $dist$ to each of its producing neighbor, n , according to their associated p_n . This offer is computed for the demand a as the sum of the QoS returned by each those neighbours for their associated part of the demand. Let $Producing_{dist}$ be the set of the considered producing distributor neighbours,

$$\forall a \in Demand < \mathcal{A} >, prod_{dist}(a) = \begin{cases} \text{empty} & \text{if some neighbour can't supply its part} \\ \sum_{n \in Producing_{dist}} offer_n(a \times p_n) & \text{else} \end{cases} \quad (1)$$

The offer output to a consumer $cons$, $output_{cons}$, considers $prod_{dist}$ as if all other consumers had already consumed. Let $Consuming_{dist}$ be the set of the considered consuming distributor neighbours, and, for any $c \in Consuming_{dist}$, let a_c be its demand,

$$\forall a \in Demand < \mathcal{A} >, output_{cons}(a) = prod_{dist}\left(\sum_{c \in Consuming_{dist} \setminus cons} a_c \oplus a\right) \quad (2)$$

Please note that this approach is coherent with the above mentioned examples. It also provides more information and allows to still use a simple model that only considers two types of information, demand and offer.

3) *Prosumer*: The prosumer house with PV panel needs to be separated into 3 components. One defining the offer (PV Panel), the other the demand (ProsHouse) and the third the distribution (Dist3). Please note that, in case the PV panel does not have any actuator and provides energy only to the house, it is possible to remove this component and only use a “gain” on the links, implemented thanks to the sensors (see Section IV-B3). In this case, the sensors of Dist3 and ProsHouse both transform the observed output over time by applying the weather-dependent prediction of the production of the PV Panel. If it is not the case, a separate element is required, either to be able to receive actuator modifications or to be seen by Dist 3 as a source of energy that can be distributed to the network.

4) *Storage*: The Storage unit is a distributor with two neighbours that can be decomposed as three internal components. It holds a consumer, which defines a demand, an internal state, which holds the information about the stored energy, and a producer, which defines an offer. The demand is output to one of the neighbours and the offer to the other.

The decision related to the storage is about how much energy is stored and at which time, this is why the storage demand is held by an actuator. The internal state is a private output that holds the stored energy in the form of a demand, which may be initially not null. It is updated by the storage demand, which adds energy, and the observed consuming neighbour demand, which subtracts energy. Also a special function defines losses by modifying the internal state accordingly.

The maximum capacity offer is defined by the internal state. The QoS of energy offered to the storage consuming neighbour is independent of the consuming neighbours demand. Indeed, the offered energy has already been produced at the time it is stored. Hence, the consuming neighbours are informed of this QoS even if their demands are null. For any time point t , let QoS_{stor}^t be the QoS returned by the application of the demand of the storage at time t to the offer of its producing neighbour at time t (assuming this demand is sustainable). Let $stored^t$ be a demand indicating the stored energy at time t (i.e., the internal state). For any demand a^t of the consuming neighbour, the storage can assume a^t if a^t is inferior to $stored^t$, and the offered QoS is always equals to QoS_{stor}^t .

Note that, if necessary, you can encode the fact that the storage unit can not simultaneously store and deliver by triggering an `InvalidInternalState` whenever a positive value of the demand actuator matches a positive value of the observed demand of the consuming neighbour.

5) *Main Grids*: Main gas and power grids are typically producers that are out of the scope of control of the considered smart-grids. Hence, they do not provide any actuators. If GRENAD is used as a simulator, they will be implemented with a predefined offer to output. If it is used as a piloting tool, they exploit the *interface pattern* for two purposes. Firstly, to get the information from the external system they represent and construct from it an offer, and secondly, to transmit the demand computed by GRENAD in the appropriate format.

6) *CHP*: The CHP unit offers energy both on the heat and the electricity flows. It is a distributor since it transforms energy by consuming it from the main gas grid and supplying it to Dist0. The CHP offer to Dist0 is computed from the main gas grid offer, and an internal actuator that indicates how much of the energy consumed is used to produce electricity and how much is used to produce heat. CHP demand to the main gas grid is computed back from Dist0 demand given the internal actuator value.

7) *Wind Farm*: In our scenario, the orientation of the wind farm turbines is controlled by the application. Hence, the wind farm producer provides a collection of actuators, each one defining the orientation of the turbine it is associated with, along the time. Those actuators can either be manually

controlled by a remote end-user or dynamically optimised by a coordination mechanism run by GRENAD (see for instance Section V).

In the application we are considering, the objective is typically to optimise the energy production, distribution, and consumption as a whole. The optimisation of the wind farm production is not about maximising independently their production but coordinating the production of the different producers (main grids and CHP) as well as the storage units in order to optimise the overall QoS of the energy supplied to the consumer. One may also want to include the consumer in order to match the consumption with the production capacities. GRENAD ergonomically supports the implementation of such a coordination protocol by exploiting the *state pattern* coupled with the natively proposed *publish subscribe pattern*, as described in the next section. However, the definition of such a protocol in the complex case of this rich scenario is behind the scope of this article.

V. OPTIMISATION OF ENERGY DISTRIBUTION

In order to present the genericity and flexibility of GRENAD we describe in this section the port in GRENAD of a distributed algorithm that performs an optimal distributed dispatching of electrical power over radial networks. The presented algorithm solves a distributed constraint optimisation problem and is a part of [14], the port to GRENAD introduces some modifications. First, GRENAD generic way of handling energy allows to easily apply the presented algorithm in a more general context than [14] (see Section V-B). Second, the combination of the *State Pattern*, from [6, p. 305], with the natively implemented publish/subscribe pattern allows a simple description and implementation of the distributed algorithm. In particular, no effort have to be put on the communication, one only needs to focus on how each component reacts to the observed state of its neighbourhood.

We first expose an overview of the distributed algorithm, then we describe some hypothesis on the production, and last, we details the two phases of the algorithm.

A. Overview of the Distributed Algorithm

The application cases of the algorithm are composed of consumers and producers with a static demand and offer. All of them can be directly represented using the proposed model, the nodes being modeled as distributors, which extend the `KirchoffStar` class. We also use the sensor/output algorithm, described in Section IV-B3 to implement maximum capacity of the links. The solution computed by the algorithm defines: for each producer, the demand it will have to deliver, for each consumer, the QoS resulting from its demand, and, for each distributor, the percent of the overall demand of its consuming neighbours it will require from each of its producing neighbours. At the end, if the offer can answer the demand, an optimal distribution will be implemented, otherwise all consumers will be informed that the network is overused. The use of the `KirchoffStar` class either ensures

that the demand is satisfied or detects any demand that would violate the flow conservation constraint or the capacity.

The algorithm requires the network to be tree shaped (e.g. electric network), and proceeds in two phases. In the first, the *collect phase*, information is going from the leaves to the root: each node acquires and transmits to its parent the overall consumption and production information about its subtree. In the second, the *propagation phase*, information is going from the root to the leaves: each node decides an optimal power dispatching given the one of its parent.

Implementing this algorithm in GRENAD is done in the following way: each distributor possesses an actuator named *proposal* for each of its distributor neighbours, and observes with a sensor the proposals of these respective neighbours. This communication network allows the information required by the *proposal phase* to be transmitted from point to point. Each actuator is updated by the optimisation algorithm. GRENAD allows transparently this optimisation algorithm to be either located at the level agent or in a remote controller. For the sake of clarity, in this section, we refer to the initial state of sensors and outputs of the `KirchoffStar` as *regular* sensors and outputs.

The implementation exploits the *state pattern*: a given behavior is associated to a certain instantiation of the *proposal* and *regular* sensors, output and actuator. We distinguish three states:

- 1) *Not Ready* distributor: the distributor did not already receive enough information from its neighbours to be able to act. It is therefore waiting.
- 2) *Collect Ready* distributor: all but one of the distributor neighbours has instantiated their *proposal* actuator, the only neighbour that has not instantiated its proposal is then considered as its parent and the others as its children. When a distributor is in this state, it computes and instantiates its *proposal* actuator.
- 3) *Propagation Ready* distributor: all distributor neighbours have instantiated their *proposal* actuator. The distributor has then enough information to compute an optimal distribution between itself and its neighbours. The optimal distribution for this component is implemented by instantiating its *regular* outputs. Also, it is propagated by providing instructions to its children by the mean of its *proposal* outputs.

In the next section we detail how a proposal is computed and how the optimal configuration is determined.

B. Production Characteristics

In order to compute an optimal distribution, certain hypothesis have to be made on producers. We consider a smart-grid where the energy demand is modeled as an amount \mathcal{A} and the quality of production as QoS . We consider the following algebra over \mathcal{A} and QoS , required by the computation of the following sections:

- $\oplus^{\mathcal{A}}$ is a binary operator over \mathcal{A} ,
- \oplus^{QoS} is a binary operator over QoS

- $\succ^{\mathcal{A}}$ is a preorder over \mathcal{A} .
- $\succ^{\mathcal{QoS}}$ is a preorder over \mathcal{QoS} .

In this section, a producer refers to a tuple $(p, \max P_p, \alpha_p, offer_p)$, where p is its identifier, $\max P_p$ is the maximum amount of energy it can produce, α_p is a comparable object that allows to compare producers with respect to their quality of production and $offer_p$ is their offer function. A higher α (given \succ^{α}) is equivalent to a better quality of production, i.e. :

$$\forall p, q \text{ producers}, \forall a \in \mathcal{A}, a \prec^{\mathcal{A}} \max P_p \wedge a \prec^{\mathcal{A}} \max P_q, \\ \alpha_p \succ^{\alpha} \alpha_q \iff offer_p(a) \succ^{\mathcal{QoS}} offer_q(a) \quad (3)$$

In [14] the author considers a single energy flow whose amount is expressed in kWh , and encodes α as the slope of a linear function that associates a CO_2 impact to the production of a given amount of energy.

In this setting, given a set of producers, an optimal distribution is obtained by saturating the production of producers in decreasing order of α . This is done by $dist^{opt}$, a function that considers a set of producers and an amount of energy that returns for each producer the amount of energy it must produce in the optimal distribution. \hat{dist}^{opt} is a function that exploits the result of $dist^{opt}$ and returns the part of the consumption attributed to each producer in the optimal distribution, in order to instantiate the actuator of the `KirchhoffStar`.

C. Collect Phase

A proposal is computed by a *Collect Ready Distributor* and transmitted to its parent, which will have to decide the flow of energy between them. The information provided by a *Collect Ready Distributor* in a proposal to its parents is (1) the list of producers of its subtree, composed of the producer neighbours listed in its children proposals and its own producer neighbours, and (2) the overall demand of the consumers of its subtree, computed from the overall demand provided by its children proposals and the demand of its consumer neighbours. Given this proposal, the parent has to decide, during the implementation phase, whether it considers neighbouring distributors to be producing or consuming. It also has to decide of the amount of energy demanded to producers, the remaining energy of the overall consumption being assumed by the producers of the subtree. Please note that the transmitted energy between a distributor and its parent cannot exceed the maximum capacity of their power line. Hence, proposals are modified so that the overall demanded consumption does not exceed this limit. Exceeding energy is deduced from the subtree producers offer.

The proposal of a *Collect Ready Distributor*, d , is a tuple $(Prop_d^{Offer}, Prop_d^{Demand})$ where $Prop_d^{Offer}$ is a list of producers and $Prop_d^{Demand}$ a demand. They are computed as follow: Let $Cons_d$ be the set of consumer neighbours of d , $Prod_d$, the set of its producer neighbours, and $Child_d$, the set of its distributor neighbours that have defined their proposals. To say that d is a *Collect Ready Distributor* is equivalent to say that $Child_d$ refers to all but one of its distributor

neighbours, \hat{d} , which is its parent. The overall demand of d subtree⁴ is computed as the sum of the demands of its consumer neighbours and its children proposals. The list of producers of d subtree is computed as the union of both its producer neighbours and the producers listed in its children proposals.

Let $Excess_d$ be the difference between the line capacity and the overall demand of d subtree. Two cases are possible: (1) $Excess_d$ is negative or null, in which case the parent can answer the demand without any restriction. In this case, the proposal of d is defined as the list of producers and the overall demand of its subtree; (2) $Excess_d$ is positive, in which case the answer of the parent is limited by the transmission line. The exceeding amount has then to be assumed by the subtree. In this case, d proposal demand is defined as the line transmission capacity. The exceeding power is dispatched in an optimal way using $dist^{opt}$ on d subtree list of producers. For each producer p of d subtree, let $local_p$ be the demand attributed to p by the aforementioned call of $dist^{opt}$. d proposal offer is then a modified list of producers of its subtree. First, for any p , $local_p$ is subtracted from $\max P_p$. Second, for any p , $offer_p$ is modified into $offer'_p$ by assuming that $local_p$ is produced, i.e. :

$$\forall a \in Demand < \mathcal{A} >, offer'_p(a) = \\ offer_p(a \oplus^{Amount} local_p) \quad (4)$$

D. Propagation Phase

A distributor, \hat{d} , is in the *Propagation Ready* state if all of its neighbours have either defined their proposal (i.e. they are its children) or they outputs (i.e. there is exactly one, which is its parent in the network tree or zero if it is the root of the network). Such a distributor has the responsibility to determine a globally optimal distribution to each of its neighbour by instantiating its outputs. To do so it considers the overall demand and the list of producers of its subtree. Those information are computed from the children proposals, the consumer and producer neighbours and the possible parent, which can either be seen as a producing or as a consuming neighbour.

For each producer p of its subtree, let $prod_p^{opt}$ be the demand it must assume in an optimal distribution. $prod_p^{opt}$ is computed by $dist^{opt}$ applied on the overall demand and the list of producers of \hat{d} subtree. To each child d of \hat{d} , is attributed the total amount of energy, $Prod_d^{opt}$, the child should supply in a globally optimal solution. $Prod_d^{opt}$ is computed as the sum of $prod_p^{opt}$ for any p belonging to $Prop_d^{Offer}$. Three cases are possible: (1) $Prod_d^{opt} = Prop_d^{Demand}$, which means that in an optimal solution d subtree supports exactly its demand. No energy is transmitted between d and \hat{d} . This subnetwork can be isolated; (2) $Prod_d^{opt} > Prop_d^{Demand}$, which means that in an optimal solution d subtree supports its own demand and offers some exceeding energy. In this case \hat{d} is configured so that it is demanding this exceeding

⁴the subtree of the network whom root is d

energy from d ; (3) $Prod_d^{opt} < Prop_d^{Demand}$, which means that in an optimal solution d subtree does not support its own demand. In this case, \dot{d} is offering the difference to d , \dot{d} declares itself as a producer whose maximum production is the remaining demand. Please note that in this case, even if d does not support its own demand, some of its producers may no be used in the optimal configuration. However, a distributor always consumes first all the energy offered by its parent.

VI. CONCLUSION

In this work, we have presented GRENAD, a JADE-based framework, that allows to describe, simulate and pilot smart power grids. The aim of GRENAD is to provide a great flexibility in the design and implementation of smart power grids applications, so that it can be used either as a standard basis to build and test such applications or as an interface to monitor and pilot actual grids. As such, it is compliant with the main business and organizational models: it proposes the classical roles of production, consumption, prosumption, distribution and storage. GRENAD allows time-dependant monitoring and computation; it supports both internal physical models and connection to external ones; it provides flexibility in manual, automatic or autonomous processing of the computed energy-related information; and finally it eases the integration of distributed smart grid control algorithms. To meet these objectives, GRENAD exploits a combination of Multi-Agent and component-oriented paradigms. Such an approach allows to build complex agents using simple reusable components and, thus, capitalize the development of applications.

We also have proposed a model of smart-grid that handles demand and offer and rely on the most general interpretation of consumption, production and distribution. The information about energy exchanged by the agents is complex. Indeed, it does not impose any assumption on the energy definition, which can be a combination of several flows and it is projected over time. Also, its management handles asynchronicity of computations and incoherence of states. The use of algebras and the category pattern [13] allows an efficient and ergonomic processing of this information. Also, GRENAD does not require to deal with communications, since it implements a transparent event-driven approach for information exchange, supported by the publish-subscribe pattern. Lastly, we demonstrated the ease of implementation of a generic sophisticated optimization distributed algorithm thanks to this architecture and the state pattern.

Future works include the development of more component libraries, of simulator interfaces and of dedicated optimization algorithms. Another interesting line of development

would be to improve the reliability of the platform by integrating model checking capabilities.

REFERENCES

- [1] Kyle Anderson, Jimmy Du, Amit Narayan, and Abbas El Gamal. Gridspice: A distributed simulation platform for the smart grid. In *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES)*, 2013 Workshop on, pages 1–5. IEEE, 2013.
- [2] BOSCH. Vppm. <https://www.bosch-si.com/solutions/energy/virtual-power-plant/virtual-power-plant.html>.
- [3] Georgios Chalkiadakis, Valentin Robu, Ramachandra Kota, Alex Rogers, and Nicholas R. Jennings. Cooperatives of distributed energy resources for efficient virtual power plants. In *The 10th International Conference on Autonomous Agents and Multiagent Systems – Volume 2, AAMAS '11*, pages 787–794, Richland, SC, 2011. International Foundation for Autonomous Agents and Multiagent Systems.
- [4] US DOE. Gridlab-d. <http://www.gridlabd.org/>.
- [5] HOMER Energy. Homer. <http://www.homerenergy.com/>.
- [6] Ralph Johnson Erich Gamma, Richard Helm and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
- [7] ETAP. Etap. <http://www.etap.com/>.
- [8] Bou Ghosh, Jingpeng Tang, et al. Agent-oriented designs for a self healing smart grid. In *2010 First IEEE International Conference on Smart Grid Communications*, pages 461–466, 2010.
- [9] TRILLIANT Inc. Trilliant. <http://www.trilliantinc.com>.
- [10] Electric Power Research Institute. Esvt. <http://www.epri.com/abstracts/Pages/ProductAbstract.aspx?ProductId=000000003002000312>.
- [11] Telecom Italia. Jade. <http://jade.tilab.com/>.
- [12] J. K. Kok, C. J. Warner, and I. G. Kamphuis. Powermatcher: Multiagent control in the electricity infrastructure. In *Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS '05*, pages 75–82, New York, NY, USA, 2005. ACM.
- [13] Miran Lipovaca. *Learn You a Haskell for Great Good!: A Beginner's Guide*. No Starch Press, San Francisco, CA, USA, 1st edition, 2011.
- [14] Sam Miller. *Decentralised coordination of smart distribution networks using message passing*. PhD thesis, University of Southampton, February 2014.
- [15] NEPLAN. Neplan. <http://www.neplan.ch/>.
- [16] Manisa Pipattanasomporn, Hassan Feroze, and S Rahman. Multi-agent systems in a distributed smart grid: Design and implementation. In *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES*, pages 1–8. IEEE, 2009.
- [17] Flexible Power. Powermatcher. <http://flexiblepower.github.io>.
- [18] D Pudjianto, C Ramsay, and G Strbac. *Virtual power plant and system integration of distributed energy resources*. IET RENEWABLE POWER GENERATION, 1:10–16, 2007.
- [19] Sarvapali D. Ramchurn, Perukrishnen Vytelingum, Alex Rogers, and Nicholas R. Jennings. Putting the 'smarts' into the smart grid: A grand challenge for artificial intelligence. *Commun. ACM*, 55(4):86–97, April 2012.
- [20] Resilient. Resilient project. <http://www.resilient-project.eu/>.
- [21] S Schutte, Stefan Scherfke, and M Troschel. Mosaik: A framework for modular simulation of active components in smart grids. In *Smart Grid Modeling and Simulation (SGMS), 2011 IEEE First International Workshop on*, pages 55–60. IEEE, 2011.
- [22] G. M. Team. *Electricity and gas supply market report*. Technical Report 176/11, The Office of Gas and Electricity Markets (Ofgem), December 2011.
- [23] Chia-han Yang, Gulnara Zhabelova, Chen-Wei Yang, and Valeriy Vyatkin. Cosimulation environment for event-driven distributed controls of smart grid. *Industrial Informatics, IEEE Transactions on*, 9(3):1423–1435, 2013.