Asynchronous approach to simulations in Smart Grid

Jose Evora, Jose Juan Hernandez and Mario Hernandez
University Institute of Intelligent Systems and Numerical Applications in the Engineering (SIANI)
Edificio central del Parque Científico-Tecnológico, Campus Universitario de Tafira, 35017
Las Palmas de Gran Canaria, Spain
jose.evora@siani.es, josejuanhernandez@siani.es and mhernandez@siani.es

***Abstract*—The approach that has been used to analyze power grids does not fit when demand side management is introduced. This is due to the fact that, from now on, the demand on power grids requires to be analyzed in an disaggregated manner. When the demand is represented using a bottom-up approach where every individual device is represented, a complex system approach for the simulation must be considered. This approach is usually performed under a synchronous timing which involves a unique clock for every entity. However, this paradigm does not consider the entity nature and, therefore, sticks the simulation time development to the entity that requires a higher pace. This involves that many calculations are performed even though they do not output a different result. The proposal of this paper considers this regard and allows to each entity to develop their time according to their nature pace. The case study shows that this approach achieve a significant gain in terms of execution time whereas there is a small loss on the memory usage.**

***Index Terms*—agent based model, asynchronous simulation, complex system, power grid, smart grid**

I. Introduction

The climate change and liberalization of markets are pushing the energy sector towards a new paradigm known as the smart grid. This paradigm is characterized by the introduction in the power grids of renewable energy sources (RES), new technologies such as storage mechanisms, massive integration of sensors and decision makers distributed along the grid. There is also a trend towards the introduction of a communication layer for the management and control of these technologies. The smart grid paradigm is also based on the use of the Demand Side Management (DSM) whose objectives include the minimization of the peak demand and the system operation and planning improvement [10]. The system complexity is therefore increased and new tools are needed for the analysis and design of smart grids.

Traditionally, simulators have been an essential tool for analyzing and designing power grid systems. Many simulation tools have been developed for this purpose: UWPFLOW [8], TEFTS [6], MatPower [18], VST [15], PSAT [13], InterPSS [17], AMES [1], DCOPFJ [2], Pylon [4], and OpenDSS [3]. However, these tools are limited to simulating smart grids specific issues, like a communication system integrated in a large-scale simulation. GridSim [7] was developed to deal with these problems. GridSim is a modified version of TSAT [5] (an industry-proven transient stability simulator) which addresses the electro-mechanic working mode of the power grid system. GridSim is a real-time simulator adapted to integrate sensing with a high data rate. The modeling approach of these tools manage the production and demand in an aggregated manner.

However, smart grid simulations require the representation of both demand and production in a dis-aggregated manner. Tafat is a tool able to simulate smart grids that enables a bottom-up representation which includes, not only a technical system description, but also a sociological description of people interacting with the system [9]. With this representation, it is possible to design, implement and test smart grid simulations. All these tools execute the simulation with a synchronized approach. Synchronous simulations have the advantage of simple time management as all objects of the modeled system are running in the same time instant. It forces objects to always perform calculations, in every time step. Sometimes these calculations are unnecessary due to the fact they cannot provide new results. For example, a washing machine is usually waiting for an agent to be turned on, considering this as an event. Later on, it develops some washing cycles where the power may vary along the time. Whenever the washing machine state does not change, calculations could be avoided.

In this paper, it is explained how Tafat implements a simulation approach based on an asynchronous timing to study Smart Grids. In a synchronous simulation, there is only one process that is responsible for the time management which uses a static timestep. Asynchronous simulations allow to manage events or develop the time execution with a time variable since each process is able to manage its time development. For example, in an asynchronous simulation where the power consumption of a washing machine is analyzed, calculations would be done only when the washing machine state changes. The advantage with respect to a synchronous simulation is clear since in the synchronized case, calculations are done every time step. This approach will be tested from the point of view of the performance through a use case. This use case regards the event and data interactions among elements of a power grid.

In the context of discrete event simulation the asynchronous concept has dual connotation. One of them consists in variable time-increment procedures as opposed to a "synchronous" or fixed time-increment procedures for simulation control. This connotation is related to the known concept Distributed Discrete Event Simulations (DDES) [12], [14]. For instance, Simula [16], a simulation-oriented programming language, is based on this asynchrony concept where the time-management is mainly event-based. This kind of asynchrony was already considered in Tafat through using different time steps for each mode of behavior [9]. On the other hand, the asynchrony can be understood as a non-sequential processing where simulation parts may not be executed in the proper temporal order. That is to say, later parts of the simulation may be executed before previous ones [11]. The last connotation is the one to which we subscribe in this paper. The objective is to apply the time-management to each model element allowing them to be in different time instants.

II. Tafat asynchronous simulation

In initial Tafat framework releases, the simulation of power grids was done following a synchronous timing approach. This paper examines a new approach to achieve asynchronous simulations with Tafat. This section introduces the concepts and constructions that Tafat architecture includes to model power grids. Theses constructions are focused on dependencies between objects that are massive and very relevant in a complex system simulation. In order to properly handle an asynchronous simulation, it is important to understand the dynamics of coupled objects. For the sake of clarity, a traced execution of objects interaction during an asynchronous simulation is demonstrated.

A. Tafat system modeling

modeling in Tafat is done by developing two views: an object oriented description of the scenario, and a behavioral specification of these objects. The first view is the static representation of the real world objects, where each single object is described with features (static attributes) and variables (dynamic attributes). This representation also includes the specification of object relations. The second view focuses on objects' dynamic : that is, how objects should behave, emulating the way they act in the real world (behavior). A single object can be associated with several behaviors. These associated behaviors are responsible for modifying the model object variables along the time. Object variables are encapsulated and can be only accessed and modified by their associated behaviors.

The solution of separating objects from their behaviors, makes it straightforward to change the method for calculating variables. In this way, it is possible to simulate different behavioral aspects with the same representation.

For example, a washing machine representation contains:

1) Static View
   - The description of their features such as capacity, installed power and energy labeling, and their variables such as mode (on, off), active program (temperature, cycle, timeout...), and active power.
   - The topological relation to the electrical installation in a household
2) Dynamic View
   - The specification of the washing machine-operating mode. The behavior is then associated with this model object.

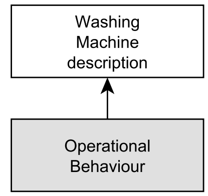

Fig. 1. The operational behavior of a Washing Machine is associated with the Washing Machine object description

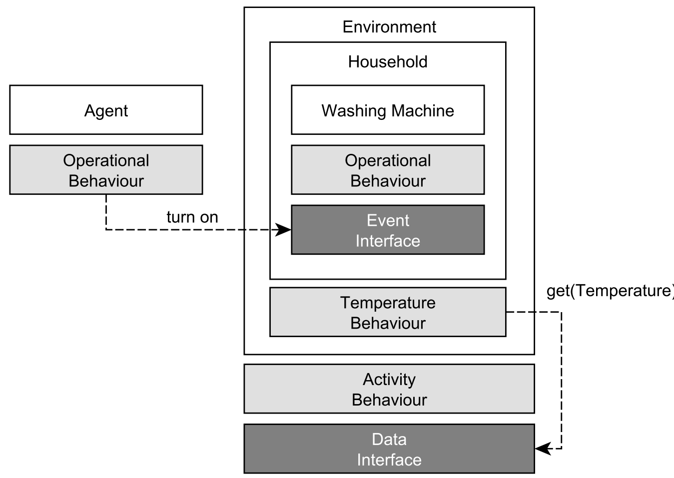

Fig. 2. Dependencies examples between objects

Normally, a behavior is coupled with other objects, both for querying their states or sending messages in order to change their states. In the Tafat model representation, defining behavior which interacts with other objects is allowed.

This representation approach consists of interfaces that should be defined in the object which could be externally accessed. In Tafat, there are two types of interfaces:

1) event interfaces that handle messages and are responsible for modifying the object internal variables as requested, and
2) data interfaces that handle queries and provide the value of requested attributes

An example of these types of interfaces is shown in the figure 2. On the one hand, the thermal behavior within a household has a data dependence with the temperature of the surrounding Outdoor. In this case, the Outdoor temperature data is requested by the associated object through the outdoor data interface. On the other hand, an agent sociological behavior wants to turn on the washing machine. Then, this sociological agent must use the washing machine event interface to achieve this task. The washing machine event interface would change the washing machine mode to ``ON". The washing machine operational behavior would calculate the proper power consumption based on this mode. Later on, when the cycles end, the operational behavior turns off the washing machine.

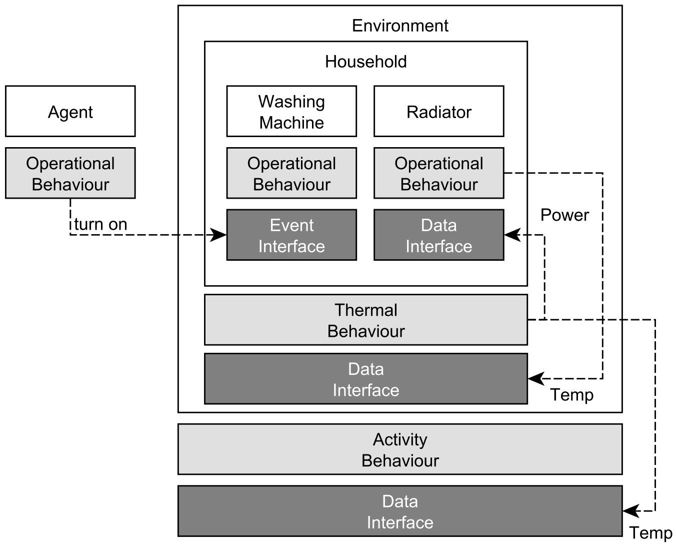

Fig. 3. Model composition. The words that start by a upper case letter are pointing out a variable requirement. Words starting by a lower case letter are related to orders. Note that *Temp* refers to temperature

B. A power grid simulation case

In order to consider the main issues that involve asynchronous simulation a simulation case is proposed to show how objects interact when working in different times (Figure: 3).

The objects within this simulation case are an Outdoor, a Household, a Washing Machine and a Radiator.

- The Outdoor is the object that represents environmental conditions, in this case, the temperature. The Outdoor temperature behavior is responsible for setting the temperature which can be loaded from an external database.
- The Household works as a container of the appliances of a household, a Washing Machine and Radiator in this case. The Household behavior is concerned with the thermal dynamics inside the household.
- The Electrical devices inside the Household are a Radiator and a Washing Machine. These devices are handled by an Agent.
- Finally, the Agent represents the people living in the Household and the associated behavior defines the actions that these people are performing. For example: a person turning on the Washing Machine.

The coupling in this model is represented by the dotted lines in the figure 3. This coupling is always defined from behaviors to interfaces. The Agent depends on the Washing Machine to change the operation mode of this device. The Radiator depends on the Household temperature, since the heat radiation is calculated based on the gap between the Radiator reference temperature and the Household temperature. The Household has two dependencies: with the Outdoor temperature and with the Radiator power, since the Household temperature is calculated by a numerical solution of a differential equation which includes these two variables. Note that, in this case, there is a cyclic dependence between the Household and the Radiator.

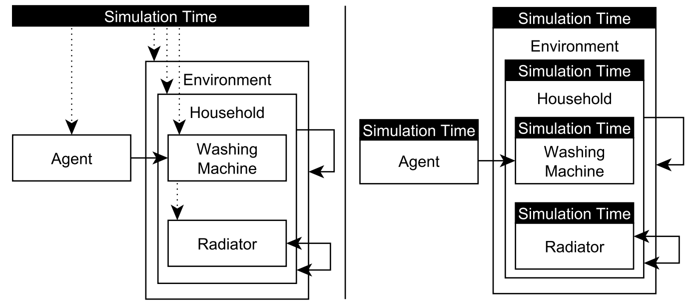

Fig. 4. Synchronous vs Asynchronous simulation

C. Asynchronous simulation dynamics

A system simulation requires time-management to ensure that temporal aspects are correctly represented and emulated. This temporal representation only exists during the simulation process and is referred to as ``Simulation Time''. Simulation Time is represented as a timestamp, a long integer where a unit corresponds to a millisecond of real time.

The time-management in a synchronous simulation is centralized while the time-management in an asynchronous simulation is distributed. That is, an asynchronous simulation involves that every object manages its time, so they could have different timestamps (Figure: 4).

In this simulation paradigm, when an object is not coupled with other objects, its Simulation Time develops without considering other object Simulation Times. In this simulation case, the Outdoor is completely independent of other objects.

However, when objects are coupled, the challenge consists of correctly reproducing temporal relationships. The identified temporal relationships are as follows:

1) Coupling with a data interface
2) Cyclic coupling with data interfaces
3) Coupling with an event interface

In the following sections these relationships are discussed.

1) Coupling with a data interface: Since an object could access a variable of an external object which may be in a different time instant, every object must keep the different states that have been calculated during the simulation execution. So, when a variable is modified, a state snapshot is created in order to keep the object state in this time instant.

If an object is querying for a variable value in a time instant t_i, there are two cases: the object Simulation Time is delayed or ahead with respect to the external object Simulation Time. In the first case, the external object is able to provide the value by retrieving the last snapshot previous to this time instant (t_i). I n the second case, the dependent object must wait until the external object reaches this time instant (t_i).

In the figure 5, the first case is shown. The Household Simulation Time is t_i and the Outdoor Simulation Time is t_j. Whenever t_i is lesser or equal than t_j, the requested data can be delivered since the data has already been calculated and stored.

However, when the Household Simulation Time (t_i) is greater than the Outdoor Simulation Time (t_j), the Household behavior is blocked (Figure: 6) until t_j is greater or equal than

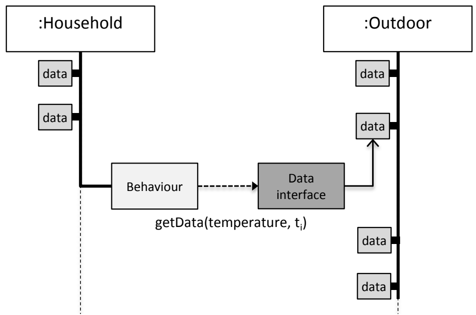

Fig. 5. Household requires the external variable temperature from the Outdoor. Note that the time is vertically represented

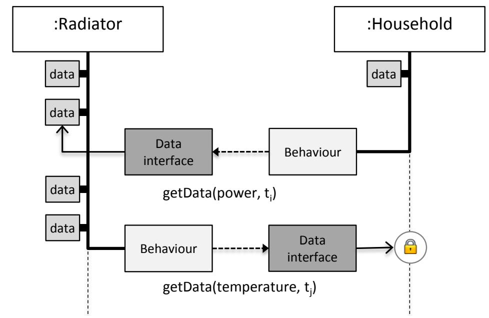

Fig. 8. Radiator and Household cyclic dependence resolution

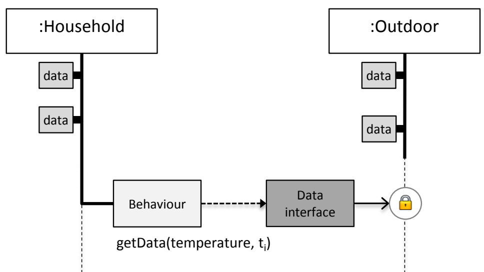

Fig. 6. The Household behavior request is blocked since the Outdoor Simulation Time is delayed with respect to the Household one

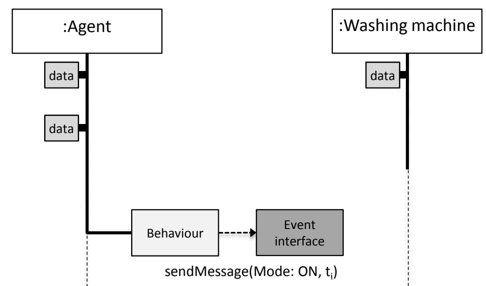

Fig. 9. The Agent sends a message to turn on the Washing Machine

t_i (Figure: 7) delivering the last Outdoor Temperature value stored in the last calculated snapshot.

2) Cyclic coupling with data interfaces: The cyclic dependence is a concrete case of the data dependence. Two objects depending on each other whose Simulation Times are different, is handled with the following rules: the most delayed one will always retrieve the required data while the most advanced will be blocked until the delayed reaches its Simulation Time (Figure: 8). The mutual blocking is not possible since objects retrieve the value for the current Simulation Time to calculate the next Simulation Time value.

In the example shown in the figure 8, the Household requires the power consumption of the Radiator in order to calculate the new temperature value. On the other hand, the Radiator behavior needs the Household temperature value to modify the Radiator state, since the reference temperature at the Radiator thermostat serves as a control mechanism.

3) Coupling with an event interface: The event coupling means that an object receives external messages that contain orders for changing its internal variables. This is the case of objects which are managed by people that are represented as Agents in the model. The Agent interacts with these objects by sending a message using the object event interface. When the message is received by the object interface, the object Simulation Time is developed and then, a new snapshot state is created.

It could happen that the agent develops its simulation time without the intention of sending an order to any object. In this case, the agent behavior must send a ``Notification Time Message˝to the object. In fact, when the agent simulation time develops, the agent behavior must send a Notification Time Message to all objects the agent is controlling. This notification determines how long an object can develop its Simulation Time. This type of relationship means that object's Simulation Time that is controlled by an agent, will never exceed the agent Simulation Time.

Figures 9-12 show an event relationship between a social Agent that turns on the Washing Machine. In this example,

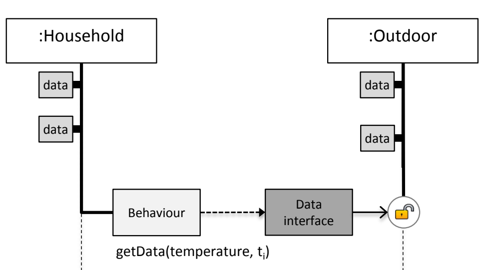

Fig. 7. When the Outdoor Simulation Time reaches the Household one the data is delivered.

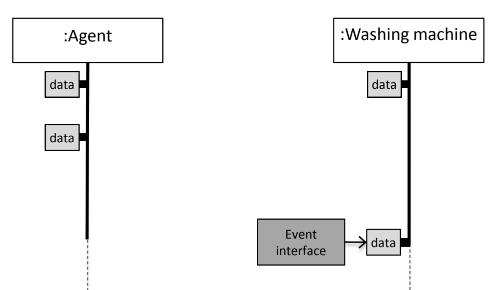

Fig. 10. The Washing Machine event interface changes the object mode to on

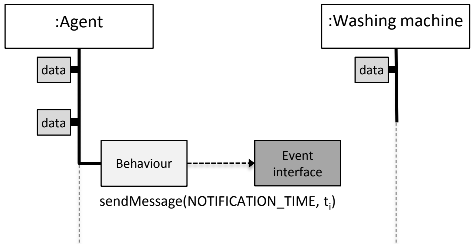

Fig. 11. The Agent indicates the Simulation Time in which it is to its controlled objects

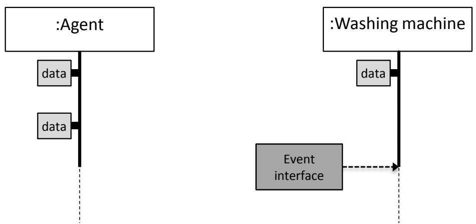

Fig. 12. The washing machine receives the message. Now the washing machine can develop its time until the temporal point indicated in the message

the Washing Machine Simulation Time is always behind the Agent Simulation Time. In other words, the Agent Simulation Time sets a restriction for the Washing Machine Simulation Time.

In the case of the Washing Machine, its power consumption would be 0 at the beginning of the simulation as it's off. Therefore, a new snapshot is created when the Agent turns on the Washing Machine. From that moment, the Washing Machine behavior will calculate the new power consumption with the restriction that the calculations development should not exceed the Agent Simulation Time, in case the agent turns off the Washing Machine.

4) Scales: The dependencies explanation has been focused on the low scale level. This is due to the fact more complex interactions take place at this level in the demand simulation of the power grids. Scaling up from the presented case to power grid levels demonstrates how the time would be developed following a bottom-up approach. In the figure 13, information flows are shown which indicate how the demand power is aggregated from the lowest levels to the highest ones at a concrete time slice. This aggregation is required to calculate the demand at every scale. Assuming that every element of a level makes the same calculations, it could be observed that each level may be delayed with respect to the lower one. This is the typical case since the upper elements are waiting for the information coming from the lower elements. However, it is possible for all of them are in the same time instant. It is not possible for upper levels to be ahead of the lower ones.

D. Object time management

In the previous cases, the discussion was focused on objects with a single type of behavior. However the time-management of an object with several behaviors or/and several event interfaces must be dealt with. Every time a type of behavior is executed, it registers the Next Time Execution, that represents

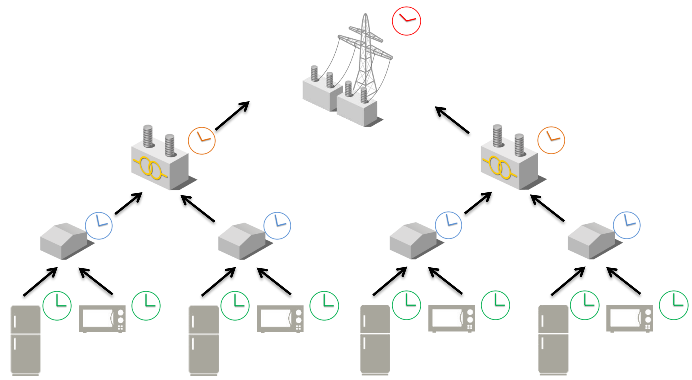

Fig. 13. Demand simulation in a higher scale

when it should be executed. The object time-manager selects the behavior with the nearest Next Time execution to the current Simulation time. The event interfaces are dealt with in the same way, so that the Interface Next Time Execution corresponds with the time defined in the last message received. Whenever a received message concerns a variable value modification, the object behaviors will be executed afterwards allowing a change in their Next Time Execution, according to the new state. Therefore, objects can dynamically develop their Simulation Times : that is, their Pace could vary from one Simulation Time to the next one. To illustrate the internal time-management, the photovoltaic cell behavior is studied. This behavior calculates the generated power, based on the environmental solar radiation. Therefore, the generation power variable will vary along the day until the sunset when the production will become 0. Then, this variable will not change until sunrise. According to this behavior, three solutions can be proposed to avoid systematic calculus along the night:

1) When the sunset is reached the behavior registers the Next Time Execution in the sunrise time, whenever this data is available.
2) When the sunset is reached the behavior registers the Next Time Execution of the previous known sunrise time. This temporal jump may avoid the first solar radiation when the sunrise time is before the already known one. Therefore, the Next Time Execution could be the previous known sunrise time minus ten minutes.
3) The photovoltaic cell outdoors could send messages to the photovoltaic cell event interface whenever solar radiation changes. Following this, the photovoltaic behavior could register its Next Time Execution to infinite (sleep mode). Therefore, solar radiation changes are received by the photovoltaic cell event interface-allowing mode of behavior to access this information.

E. Implementation

In this section, architectural methods to implement this approach are presented. This architectural proposal takes into account the previously described requirements for simulating a power grid, using an asynchronous approach.

A Tafat Thread represents the execution of a single Model Object and from this point of view describes the execution

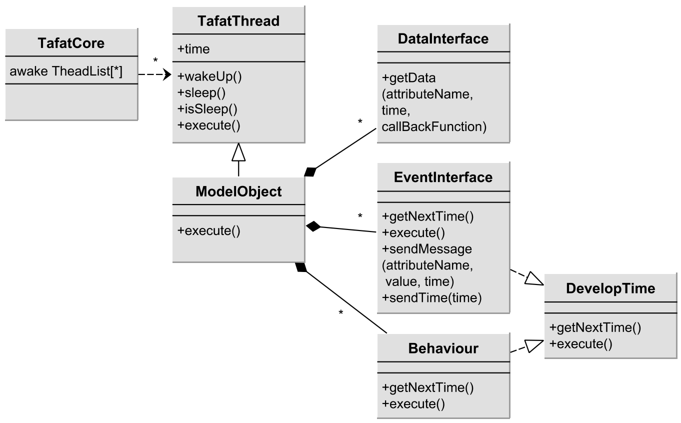

Fig. 14. Tafat asynchronous simulation architecture

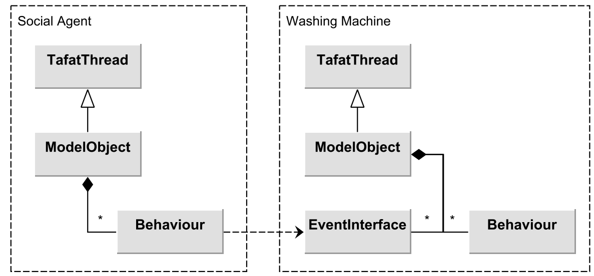

Fig. 15. The Washing Machine Event Interface Next Simulation Time turns from undefined to a defined value when the behavior of the Social Agent sends its Current Time. On the other hand, the Washing Machine behavior Next Simulation Time turns from infinite to a reachable time when its state is changed to ON by the Agent

state, awake or sleeping, and the simulation time in which it is (Figure: 14). During the execution of the whole simulation, Tafat Core request awake Tafat Threads to be executed. After this execution, a Model Object will have changed its simulation time and/or its state. In order to improve the performance, Tafat Core keeps a list of the awake threads and it is listening for state changes in threads to update this list.

A single Model Object has many controllers that can modify the Simulation Time. A controller factor could be either behavior or an Event Interface. These controllers, that implement the Develop Time interface, participate in the Model Object simulation, each of them proposing different Next Simulation times. When the Next Simulation Time of any of these controllers is undefined, the Tafat Thread that represents the Model Object turns into a sleeping mode. Once, the Next Simulation Time of all the Model Object controllers are defined, the thread will wake up. Next Simulation Time of Develop Time Controllers could be set to undefined or a value that should be greater than the current Model Object Simulation Time. A feasible value for a Next Simulation Time could be infinite, meaning that behavior is suspended, pending an external event.

For example, the Next Simulation Time of a Washing Machine behavior can be infinite, so that the washing machine is off and therefore, it is waiting to be turned on (Figure: 15). On the other hand, the Next Simulation Time of this Washing Machine Event Interface is undefined until other Model Object behaviors that use it, set the Next Time Simulation. Since the Washing Machine depends on the Social Agent to be modified, the Social Agent must inform this device of this. This Current Time is transmitted through a message which arrives at the Washing Machine Event Interface. When the Social Agent Current Time arrives, the Washing Machine Event Interface will modify its Next Simulation Time from an undefined value to the one which has arrived in the message. Whenever an event for modifying the state of the Washing Machine arrives, the Washing Machine behavior will be executed once, allowing to it to calculate its Next Simulation Time based on this new state.

Another improvement from the performance point of view is based on the Snapshots removing. A concrete Model Object may have dependences for requesting data or set values in external Model Objects. Similarly, other Model Objects could require this one to be accessed. For this reason, the Model Object must keep the snapshots for all the Model Objects which request data. As this Model Object knows the data requesters, it is able to find out the time in which the requesters are and, therefore, it could delete the Snapshots which are previous to the Current Time of the most underdeveloped Model Object requester.

III. Case study

In order to test the implementation a case study has been developed. This case study is intended to show the trade offs by comparing both synchronous and asynchronous implementations. The scenario represents the power grid demand in a disaggregated manner allowing for the study of DSM policies and its effects can be analyzed from the very bottom level of the grid. Therefore, the policies consequences can be studied at each level (devices, households, buildings...). The effects at the global system can also be studied as the simulation is performed using a complex system approach which allows to see the emergent behavior. The composition of the experimental scene used to comparatively evaluate synchronous and asynchronous approaches is described in the list below:

- 1 outdoor which represents environmental conditions
- 1,000 buildings which are located within the same outdoor
- 20,000 households (20 per building)
- 20,000 agents (each one related to one household)
- 60,000 radiators (3 per household)
- 20,000 washing machines (1 per household)

A. Coupling details

Even though the coupling has been summarized in the list above, this must be clarified as much as possible since the performance results depend on this factor, among others. In figure 16 and 17 relations that exist among the elements of the scenario are presented. In figure 16, the relations are shown at the outdoor-building level whereas 17 presents them at the building-household level. Those relations shown in those figures are defined and justified in the list below:

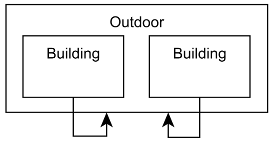

Fig. 16. Coupling details at the outdoor-building level

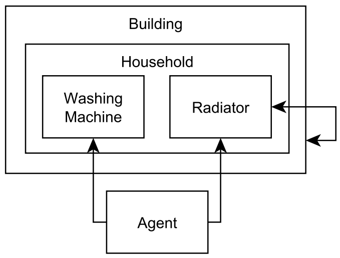

Fig. 17. Coupling details at the building-household level

- Outdoor has no coupling with other entities
- Each building is related to the outdoor and the radiators that are inside. A building get the external temperature from the outdoor which is used to calculate the thermal behavior. The radiators inside the building provide the active power which is also used in the thermal behavior. Both dependencies are required by the thermal behavior in order to calculate the internal temperature
- Households do not have any dependency since they do not need any variable and they are not requested to provide information. In this simulation, households play the role of a device container
- Every agent is related to a household. This relation implies that they are able to modify the state of the devices within its household. This modification consists in changing the device states from off to on and in reverse. Therefore, they are related to a washing machine and a radiator.
- Radiators need the internal temperature of the building in which they are. This information is taken into account by the thermostat in order to find out whether they must heat or not. Furthermore, they depend on the agent to develop their time since they cannot be ahead.
- Washing machines have the same dependency that the radiators have with the agent.

B. Timing

Both synchronous and asynchronous simulations correspond to 24 hours. In the synchronous case, the simulation pace is fixed to the entity that requires the highest pace, which is the building. The building needs a higher pace than the rest due to the fact that the thermal calculus must be performed frequently in order to decrease the accumulated error. Since the asynchronous approach delegates the time management to each entity, the way in which they develop their time can be different. The table I presents the timing configuration used at each simulation.

Moving on to the asynchronous case, the outdoor does not have a pace since it works as a database from where the external temperature can be extracted. Buildings have the same pace as the synchronous case because of the requirement presented in the paragraph above. Agents have a dynamic pace which allows them to develop the time according to their decision making. When they apply a decision they have made, they calculate the next decision and the time when it will be applied. This time will define the temporal jump that the agent will do. Radiators have the same pace of the buildings when they are on and ∞ when off since they are slept until the agent interacts with them. Washing machines behave the same but the pace when they are on is based on their cycles, this is, if the heating water cycle takes thirty minutes the temporal jump will be thirty minutes too when it is in this cycle.

TABLE I
TIMING OF EVERY DEVICE AT EACH SIMULATION

behaviors/Step	Synchronous	Asynchronous
Outdoor	60 seconds	-
Building	60 seconds	60 seconds
Agent	60 seconds	Dynamic
Radiator	60 seconds	ON: 60 seconds / OFF:∞
Washing machine	60 seconds	ON: cycles-based / OFF: ∞

TABLE II
PERFORMANCE COMPARISON BETWEEN SYNCHRONOUS AND ASYNCHRONOUS CASES

	Synchronous	Asynchronous	Benchmark
Execution time (seconds)	192	113	40.84% faster
Average memory usage (megabytes)	671	789	17.65% higher

C. Agent decisions

Every agent at each household will behave the same. Agents behavior has been simplistically developed since they are not the focus of this work. Since radiators are turned on from the beginning of the simulation, agents turn them off at 8 am. The next action they perform is turning on the washing machines at 1 pm. The last action they apply is turning on the radiators at 8 pm.

D. Performance

When considering the performance of an implementation, two important indicators are the execution time and the memory usage. Therefore, both implementations will be compared under both indicators. The table II presents those indicators for each simulation.

There is a significant improvement when applying a asynchronous paradigm in the execution time for this experiment. However, there is a certain penalty in the memory as a result of the use of snapshots. It is also interesting to see the times when every element kind finished the simulation. In the synchronous case, it is evident that all of them finished at the same time. However, in the asynchronous one (Figure: 18), this does not happen since every entity develops its time according to its pace.

Since outdoor does not have pace, it finishes at the very beginning. Agents finish when they stop the events sending.

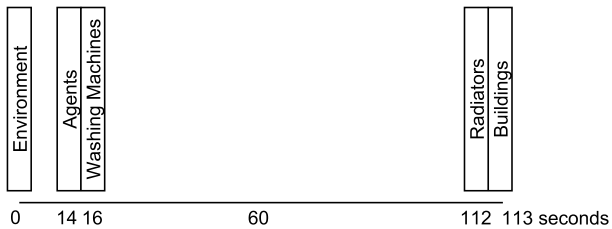

Fig. 18. Times in which each entity kind finished

This makes possible to the radiators and washing machines develop their times. Seconds later, washing machines are able to finish. However, radiators cannot since they are coupled with the buildings. Radiators and buildings finish at the same time involving the simulation end.

IV. Conclusions and Outlook

Going towards asynchronous complex system simulations involves a re-conceptualization. This re-conceptualization affords objects interaction issues which could come from both data and event dependencies. In a synchronized execution environment, every object of the system is in the same time slice and the time-management is usually handled using a single clock. The main advantage of this approach is, among others, the simplicity when accessing or modifying an object since all of them are in the same time slice. However, the main disadvantage of the execution of object calculations, is that some of them are unnecessary because the execution is not going to produce any different output.

The use of an asynchronous approach for simulating complex systems provides flexibility in the object evolution. Objects can freely develop as far as their dependencies are satisfied. Furthermore, object behaviors can be both event and time-based which provides the possibility of having sleeping behaviors. This sleeping behavior could change their status to active by receiving external events. The behavior step may vary from one execution to the next at a dynamic speed. Both sleep mode and dynamic speed are important features to avoid the systematic calculations at fixed steps which produce the same values. Furthermore, we think this approach may facilitate the parallel complex simulation execution.

Concerning the results obtained in the experiment, it must be noted that the performance we have obtained cannot be extrapolated. To obtain general conclusions, more experiments must be executed as future work, in order to extract the relation between the scenario nature and complexity and the relative performance obtained between synchronous and asynchronous approaches.

V. Acknowledgment

This work has been partially supported by European Regional Development Fund (ERDF/FEDER) and Agencia Canaria de Investigacin, Innovacin y Sociedad de la Informacin (ACIISI) of Canary Islands Autonomic Government through the project whose reference is SolSub200801000137, and also through the ACIISI PhD grant funding to José Évora with reference TESIS20100095.

References

[1] AMES Market Package, http://www2.econ.iastate.edu/tesfatsi/AMESMarketHome.htm
[2] DCOPFJ Package, http://www.econ.iastate.edu/tesfatsi/DCOPFJHome.htm
[3] OpenDSS, http://sourceforge.net/projects/electricdss/
[4] Pylon, Power system and energy market analysis with Python, http://pylon.eee.strath.ac.uk/pylon
[5] TSAT - Transient Security Assessment Tool, http://www.powertechlabs.com/software-modeling/dynamic-security-assessment-software/transient-security-assessment-tool
[6] Tefts program, university of waterloo (2000), http://www.power.uwaterloo.ca
[7] Anderson, D., Zhao, C., Hauser, C.H., Venkatasubramanian, V., Bakken, D.E., Bose, A.: A virtual Smart Grid. IEEE Power and Energy Magazine (2012)
[8] Cañizares, C., Alvarado, F.: UWPFLOW Program, University of Waterloo (2000)
[9] Evora, J., Kremers, E., Morales, S., Hernandez, M., Hernandez, J.J., Viejo, P.: Agent-Based Modelling of Electrical Load at Household Level. In: ECAL 2011: CoSMoS - Proceedings of the 2011 Workshop on Complex Systems Modelling and Simulation. p. 12 (2011)
[10] Gabaldon, A., Molina, A., Roldan, C., Fuentes, J., Gomez, E., Ramirez-Rosado, I., Lara, P., Dominguez, J., Garcia-Garrido, E., Tarancon, E.: Assessment and simulation of demand-side management potential in urban power distribution networks. In: Power Tech Conference Proceedings, 2003 IEEE Bologna. vol. 4. IEEE (2003)
[11] Ghosh, J.B.: Asynchronous simulation of some discrete time models. In: Proceedings of the 16th conference on Winter simulation. pp. 466–469. WSC '84, IEEE Press, Piscataway, NJ, USA (1984)
[12] Kaudel, F.J.: A literature survey on distributed discrete event simulation. SIGSIM Simul. Dig. 18(2), 11–21 (1987)
[13] Milano, F.: An Open Source Power System Analysis Toolbox. IEEE Transaction on Power System, vol. 20, no.3 (2005)
[14] Misra, J.: Distributed discrete-event simulation. ACM Comput. Surv. 18(1), 39–65 (1986)
[15] Nwankpa, C.: Voltage Stability Toolbox, version 2, Center for Electric Power Engineering, Drexel University (2002)
[16] Pooley, R.J.: An introduction to programming in SIMULA. Blackwell Scientific Publications, Ltd., Oxford, UK, UK (1987)
[17] Zhou, M.: InterPSS, http://www.interpss.org
[18] Zimmerman, R., Gan, D.: Matpower, Documentation for Version 2, Power System Engineering Research Center, Cornell University (1997)