AAETI A HIERARCHICAL METHODOLOGY TO DESIGN AND TEST MICROGRID SUPERVISOR SYSTEMS

Edoardo CORSETTI
RSE – Italy
corsetti@rse-web.it

Giuseppe A. GUAGLIARDI
RSE – Italy
guagliardi@rse-web.it

Carlo SANDRONI
RSE – Italy
sandroni@rse-web.it

ABSTRACT

This work proposes the AAETI (Agent Automata Extended with TIme constraints) methodology to deal with the design of the control operation and safety of a microgrid. We tested AAETI to design the behaviour of the supervisor system of a rural low voltage microgrid. In particular, the goal of this supervisor is to manage the microgrid during the disconnection of part of loads, it verifies whether there is the possibility to recover the grid equilibrium with local actions or computing a new power balance.

INTRODUCTION

The exponential growth of green (intermittent and non-controllable) generation poses many problems to operation and control of electricity networks. These problems prevent to capture the full benefits arising from renewable generation. In order to overcome these difficulties new powerful and flexible automation and control systems must be designed and developed. The operation of these systems upon low and medium voltage electricity networks can enhance the network management even with a massive presence of renewable generation.

To guarantee the safety of a microgrid it is necessary to develop a specific safety plan, able to cover the possible emergency situations. Typical outages possibly occurring on an electric microgrid include ([5]):

- overloaded lines, and possibly uncontrolled cascades of line tripping;
- voltage reduction, possibly leading to voltage collapse caused by reduced reactive power support;
- local area transient stability problems, related to the increase of long distance interchanges and coupled with the risk of reduced voltage support, inter-area oscillations due to the interconnection, intervention of protecting devices (overcurrent protections, differential protections) causing cascading events;
- hidden failures of the protection system, specific plans are used to ensure that the overall microgrid (or even virtual power plants) is protected against major disturbances involving multiple contingency events. This objective can only be reached with coordinated automatic emergency measures.

In this paper we propose the AAETI - Agent Automata Extended with TIme constraints, a methodology to support the specification, design and test of automation and control systems. This methodology supports the specification phase to represent system functionality by means of a hierarchy of agents interacting through communication channels. An agent is described by means of the operational model of timed automata. AAETI also supports the design phase providing an automatic translation step of agents into executable code and it also supports wrapping an existing functionality into an agent to merge the set of agent/ functionality into an executable program. The control system built can be tested against a given electricity network simulated by DIgSILENT Power Factory©. The control system and the grid simulator are faced with a synchronized evaluation loop and a data exchange protocol.
AAETI, thanks to the notion of agent and to the hierarchy among them, constitutes a flexible platform to specify and test a wide class of control systems architectures: centralized, decentralized and hierarchical control ones [4].

The paper is organized as follows: first an overview of the AAETI methodology is proposed, then the main agent specification features are sketched, in the third chapter an applicative example illustrates the AAETI ability to specify control systems, a final chapter discusses the verification and validation steps supported by the methodology. Conclusions propose the future development.

AAETI METHODOLOGY OVERVIEW

AAETI (Agent Automata Extended with TIme constraints) is a specification and design technique for hierarchical automation systems. In an AAETI control system the basic unit is the agent, which isolates a specific component (functionality), thus the overall control system is a collection of cooperating agents. In AAETI the system specification is given as a pool of agents hierarchically ordered ([6]). These agents monitor and control the plant according to a *macro cycle* consisting of acquiring the plant variables (*monitor)* and *performing* the decision on plant (*actuation*). Then the specification execution computes each module (i.e., agent) according to a twofold analysis of the hierarchy in two distinct phases, called:

- the *upward phase*
- the *downward phase*.

In the former, signals collected in the monitoring phase or variable settings generated by the elaboration of lower level agents are progressively made available to the upper levels of the agent hierarchy, until the top agent/s is/are reached. At this point, the *downward phase* is started: first are computed agents at the top of the hierarchy and progressively those at the lower levels. In downward computation decisions elaborated by the coarser agents are made available to the lower levels of the hierarchy. By means of the connection between agents each agent can make available its own knowledge to several agents, and can access the knowledge of one or more agents. Each agent is associated with four sets of *connections*: <*U-I*; *U-O*; *D-I*; *D-O*>. *U-I* denotes the set of input data owned by lower level agents made available during the upward computation phase. *U-O* denotes the set of *upward output data*, i.e. connections used to make agent information available to the upper level agents during the upward computation phase. *D-I* denotes data made available by agents at higher level of the hierarchy to the current agent during the downward computation phase. Finally, *D-O* denotes the set of output that can be made available to lower level agents during the downward computation phase. Next figure illustrates a representation of the agents and the different types of connections with other agents, coherently with the hierarchy levels and computation direction flow.

Figure 1 - Agent connection specification

The agent evaluation, in the upward or in the downward direction, can start once the set of (corresponding) input are available. Thus, each input data type is a queue. Output data are made available to the connected agents once the agent evaluation has ended.

The agent evaluation strongly order the upward phase and the downward phase. **Errore. L'origine riferimento non è stata trovata.** shows an example of agent hierarchy. It concerns the control system associated to a microgrid that includes two feeders, and associated to each one there are a number of nodes. AAETI methodology associates an agent to each node, section (nodes part of a feeder), feeder and the supervisor.

The set of agents are hierarchically ordered, as shown in **Errore. L'origine riferimento non è stata trovata.**, with reference to the applicative example chapter. The figure represents the upward control flow by blue arrows, and the downward control flow by red arrows.

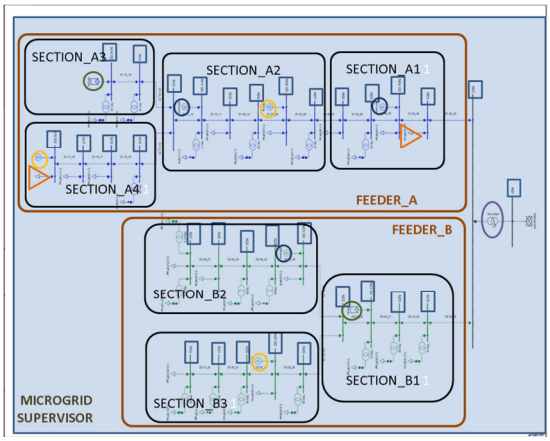

Figure 2 – Agents of a microgrid components

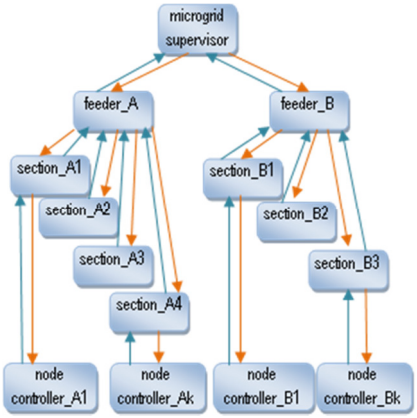

Figure 3 - The hierarchy of agents representing the microgrid control system

At the beginning of the evaluation cycle each node controller receives signals from the controlled system (i.e., transformers, generators, electrical loads, and so on). The upward evaluation of the specification starts considering the whole set of node controller agents: these agents are evaluated without a priority order as they belong to the same hierarchy level. The upward-downward evaluation of an agent consists in executing an automaton and, once the automaton evaluation ends, sending all the output signals to the destination agents. Each section agent waits for the upward inputs coming from node agents evaluation. Node controller agent, once upward evaluation has ended, waits to start the downward computation until all the section agents linked to its downward input connection have sent the specified data. Each section agent starts its own upward computation when all signals coming from node controllers linked to the upward input connection have terminated their computation. Each section agent ends upward evaluation and waits to start downward computation until the connected feeder agents have sent the output data. The upward computation ends when the supervisor agent has been evaluated. Actually, the evaluation of an agent placed at the top of the hierarchy, i.e. without upward output and downward input

connections, is not distinguished in up and down. The methodology does not limit the top of the hierarchy to just one agent. In general, multiple agents at the top level of the hierarchy are permitted. These agents are not hierarchically ordered among them.

The downward computation follows the same schema of the upward one. The end of computation of the supervisor agent enables the downward computation of feeder agents. The downward computation considers as many agent types as foreseen in the specification. The downward computation stops once each node agent has been evaluated.
The node agents at the bottom level actuate orders to the controlled grid components. The computation continues gathering the input coming from grid components enabling a new upward computation.

AGENT SPECIFICATION

Here are proposed a number of language features to specify control agents.

Agent classes and instances

An agent identifies a *type* of functionality owned by an actor involved into the control system of a microgrid. Such a component has specific knowledge, inference abilities, and interacts with other agents in order to pursue a specific goal. AAETI distinguish**es** between the *class* of functionality, representing the function as a general schema, from a specific *instance*.

Agent hierarchy

Among the agents belonging to a specification it is possible to state causal relationships which prescribe fixed causal priorities by means of data connections. Then causal order between agents to fix control evaluation is defined according to the data driven flow. The language provides also the possibility to state a hierarchy concerning time dependencies among agents. In particular, time dependencies can represent the different time constants according to the agents perform their task (see [Corsetti, 2013]). That is, agents operating at lower levels are thought to quickly react to external events, and refer to finer time domains. Conversely, higher level agents, devoted to elaborate information gathered by lower level agents and formulate broader strategies, can require larger time than the previous ones, and thus the time domains they refer to are coarser grain than the other. The language does not provide a correlation between the timed and causal hierarchies, but it is a good practice to maintain a strict correspondence between them.

Connection between agents

Connections are the way agents can exchange data. A connection is a directional channel set between two agents to refer and modify the attributes defined in the connected agent at the current instant. Connections into an agent class are distinct as in Figure 1. The agent class specified within a connection identifies the set of instances where the connection elements will be taken and these elements are identified by means of the variable declared in the definition of the connection. In an agent instance each connection is represented by a specific set of instances of the agent class connected to.

Agent behavior

The specification of an agent behavior is naturally expressed by means of a timed automaton. The general definition of finite state function is composed of a (finite) number of *states*, which can be put into correspondence with the controlled system states. The automaton is also composed of *edges* that allow to move from a state to another in presence of a specific input signal to the function. Associated to each edge there is a condition. Once the condition is satisfied it is possible to move from the current state to the one connected by the edge. We refer to the automaton state change by transition.
Automata are able to represent finite behaviors as well as infinite ones. In the former case a (set of) terminal state is identified. When a transition brings to a terminal state the automaton stops its computation. In the latter case, the automaton is not expected to terminate its computation, in these cases the automaton must follow the system behaviors possibly for an infinite time. In these automata the sequence of states is infinite, and this lead to name them ω-automata. The basic definition of ω-automata were extended in order to represent temporal behaviors of control systems [1].

A timed automaton is a classical finite state automaton which can manipulate clocks, evolving continuously and synchronously with an absolute time. In a *timed automaton* each transition is labeled by a constraint over clock values (sometimes called *guard*), which indicates when the transition can fire, and a set of clocks to be reset when the transition fires. Each clock keeps track of the time elapsed since the last reset, and can be reset independently from the other clocks. The transitions of the automaton put certain constraints on the clock values: a transition may be taken only if the current values of the clocks satisfy the associated constraints. Each location is constrained by an invariant, which restricts the possible values of the clocks for being in the state, which can then enforce a transition to be taken). We recall some notions from [2].

The agent specification is completed with the possibility to declare *integrity constraints*. Integrity constraints allow to rule parameter values. They are evaluated at the beginning of the agent evaluation and after the automata evaluation.

AN APPLICATIVE EXAMPLE

In this section an applicative example is sketched to show the expressive ability of the automata model, and how the methodology can solve a decision problem. The solution will be found taking into account local (node) and global (microgrid) strategies and ensuring the selection of the best one.

The smart grid application domain

The study case considered is a real rural medium voltage (20 kV) network connected to the distribution network. The network, shown in Figure 2, is composed of 32 MV nodes, distributed along two radial feeders, each of them about 25 km long. Some of the loads are connected to medium voltage nodes and some to low voltage nodes, and are classified in 5 classes (industrial, agricultural, residential, tertiary, public lighting), for 13 MW as a whole. There are 8 distributed generators (3 gas turbines, 3 wind turbines, 2 photovoltaics), each of them capable of about 5 MW of active power.
Both loads and generators are associated to hourly profiles of consumption and production, so the entire network can be studied in different working points. In this study case the 7 a.m. working point has been chosen: this situation is characterized by low load and high generation, leading to overvoltage issues. In this context, a particularly hard event has been introduced: the sudden disconnection of two loads (C03 and C18 highlighted in Figure 2 with orange triangles) of feeder A, losing about 2.5 MW in total. This event leads to an instantaneous 2% voltage increase in some nodes of the network close to the occurrence of the events. Actually, if during the outage the voltage value does not remain within a fixed range (±10%) the strategy must be defined and applied within 5 sec. The time constraint is devoted to avoid the intervention of generator protections which disconnects the generator from the grid within this time interval.

Formalization of a study of case

The microgrid supervisor agent is defined at the top of the hierarchy. At a lower level there are the two feeders: feeder A and feeder B. In order to develop better strategies between the feeder level and the node level, the section level has been introduced. The section level gathers a subset of the nodes, belonging to the same feeder, which are geographically near. At last, the node level represents the bottom of the hierarchy and the interface of the control system with the real electrical components.
The whole set of agents representing the microgrid control system is hierarchically linked as shown in **Errore. L'origine riferimento non è stata trovata.**.
The role of microgrid supervisor is to gather the information/decision coming from the lower levels and verify, when a fault occurs, whether the local strategies, taken at section or feeder levels, are optimal or there is the need to compute a new equilibrium point in order to recover the network operation. This analysis is carried on at the end of the upward evaluation. In the former case it enables lower level agents to perform the decisions in the downward evaluation. In the latter case, it denies the lower level agents to proceed to apply decisions. Supervisor agent starts the computation of a new operation point for generation and load. This can require more than one evaluation cycle, but must be carried out within 5 seconds. Once Supervisor has found the solution, communicates to lower level agents the decisions to be executed.

In the following a sketch of the agent specifications is given providing a view of the associated automata. Automata specification will be expressed by means of a graphical language: circles denote automata states, edges denote transitions, labels on edges represent transition specification. Double line circle denote the initial state and blue circles denote final states. Boxes with round corners denote macro states, that is a synthetic view of a number of state and transitions. We adopted macro state as a syntactic sugar in order to detail the specification elsewhere without losing the essential logic.

Node agent
The Node_controller agent is the lowest controller in the system hierarchy, a sketch of the associated automaton is proposed in Figure 4. The initial state is the S1 normal_operation. In this state it checks whether a load_disconnection event has occurred. In this case, the total amount of load loss is computed, scanning each connected load, and then, in a similar way, it computes the total reduction margin of connected generation. If the load loss can be locally compensated with a generation reduction, the agent reaches the S2 state able_to_reduce_power, otherwise it reaches the S3 state unable_to_reduce_power. Transition A is enabled by the section connected agent in order to locally recover the generation reduction. The node agent proceeds to lower the generation level. At each cycle if no load disconnection has occurred the transition B of the agent computes the generation reduction margin. This data can be useful in case this agent is involved in a generation reduction when a load disconnection occurs somewhere else in the network. At the end of the computation it returns to the S1 state normal_operation with the transition C.

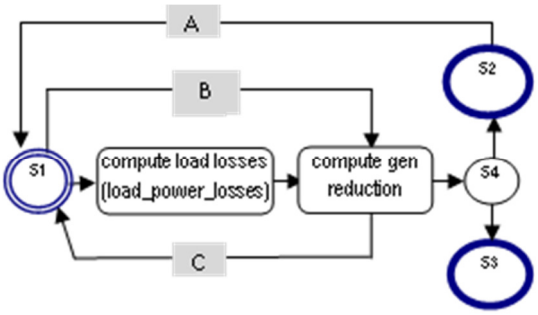

Figure 4 - Node controller agent automaton

Section agent

The automaton associated to the section agent is sketched in Figure 5, the initial state is the S1 normal_operation. In S1 agent checks whether at least one connected node controller has communicated a load_disconnection event. If it is the case it counts the number of nodes having a load disconnection, how many of them are in able_to_reduce_power state and how many are in unable_to_reduce_power state, and finally how much generation reduction margin comes from nodes in normal_operation state. If every node with a load disconnection is in able_to_reduce_power, or if the number of unable_to_reduce_power nodes is lower or equal then 2 and they can be compensated by a reduction in other nodes, the section reaches state S3 able_to_reduce_gen. If the number of unable_to_reduce_power nodes is lower or equal than 2 and they cannot be compensated by a reduction in other nodes, the section reaches state S4 find_recovery_section. If the number of nodes with load disconnections is greater than 2, the section agent reaches S2 new_balance state. Finally, if no load_disconnection event has been received, the agent computes the generation reduction margin of all the connected nodes find_oprating_nodes and moves back to S1 normal_operation state.

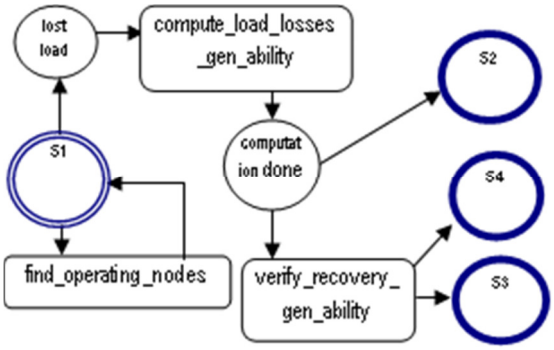

Figure 5 - Section agent automaton

Feeder agent

The automaton associated to the feeder agent is sketched in Figure 6. In S1 normal_operation feeder checks for a load_disconnection event incoming from the connected sections. If it is the case, it verifies if every section with a load disconnection is in able_to_reduce_gen state, the feeder reaches S2 able_to_reduce_gen state. If there's at least one faulty section in find_recovery_section, feeder checks if there are sections which can recover the generation excess. If this is the case feeder reaches the S2 able_to_reduce_gen state, otherwise it reaches S3 new_balance state, sending a request to the supervisor to compute a new grid balance. Feeder moves to the state S3 new_balance state anyway if the number of load disconnections is greater than 2.

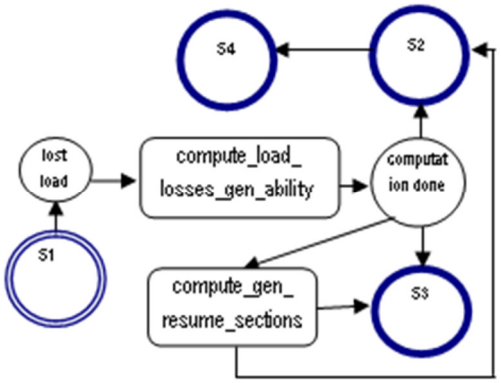

Figure 6 - Feeder agent automaton

Microgrid supervisor agent

The automaton associated to the supervisor agent is sketched in Figure 7. The initial state S1 normal_operation, checks for a load_disconnection event incoming from the connected feeder agents. If it is the case and every feeder with a load disconnection is in able_to_reduce_gen state, and the total number of faults is not greater then 2, the agent reaches S3 local_recovery state, that enables the local actions in each underlying agent. Otherwise, the state S2 new_balance is reached: this state implies an optimal power flow calculation to get a new equilibrium in the whole microgrid.

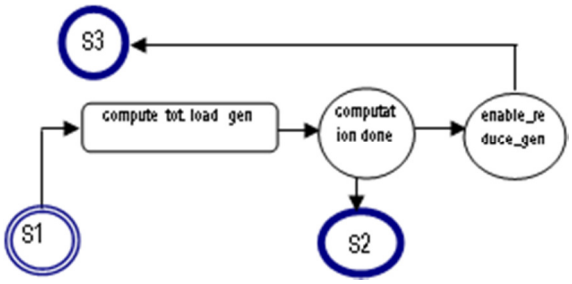

Figure 7 - Supervisor agent automaton

VERIFICATION AND VALIDATION

The AAETI abilities to verify and validate control system specification are supported by an environment including these components:

- an *XML interface* to express the specification by means of a set of agent classes, to define general functionality, and a set of agent

instances that represent a specific component;
- an *automatic translator* to transform a specification in an executable code;
- an *inference engine* to execute the program couple with an electric grid simulator via data exchanging;

Translation of an AAETI specification

A specification is automatically translated to obtain an executable code from the specification. The translation is defined according to a set of specific rules, that put into correspondence each agent construct with a specific object or method of an *object oriented programming* (OOP) environment, exploiting the structural induction on the components of the agents. In particular, the OOP notions of object, attribute and method have been put into correspondence, respectively, with the AAETI agent, attribute and behaviour description.

Multi-agent inference engine

The AAETI inference engine manages the cyclic evaluation of a control system specification by means of a scheduling list of the agents, and provides a number of library functions to support the evaluation of each agent for the current cycle.
The scheduling list order agents first according to the upward hierarchy and then according to the downward hierarchy.
For each evaluation cycle, each agent is selected from the scheduling list, according to the order, and then it is executed. The agent execution consists of:
- evaluating each attribute definition in order to update the attribute value;
- evaluating the set of specified rules.

The agent upward evaluation policy ensures that, once the low hierarchy level agents acquire the controlled system status, the higher level agents are able to plan and actuate the best strategy. However, the policy to evaluate agents can be modified just revising the access to the scheduling list. As an example, we are currently studying the effects of an upward-downward policy on the agents abilities. Indeed, this policy allows both: to gather the power system status (upward evaluation) and to collect the set of plans defined by the coarser level agents (downward evaluation) in order to optimize the actions to be performed.
The inference engine just depicted is mainly tailored on the verification and validation of the defense plan, rather than the deployment of the control system on the microgrid. The AAETI execution abilities presented so far are centered on the specification. The deployment of the control system on a real power system will be the subject of a next research activity.

Validation of a specification

The validation of a defense plan has the role to measure the ability of a set of actions to prevent blackout of a power system in a specified status. In particular, the main aim of this phase is to test whether the *defense plan* is able to:
- recognize the event, or the network status, that can lead to the complete blackout;
- plan and apply suitable actions in order to confine, as far as possible, criticalities;
- allow the fast restoration of the faulted power system components with respect to different power system configurations.

CONCLUSIONS

The paper proposed the AAETI methodology to test the specification and design of automation and control systems for microgrids. The specification of a load shedding has been proposed in order to show the AAETI ability. AAETI can be thought as a general platform to host different microgrid control architectures, such as centralized or distributed architectures. Currently we are taking into account the possibility to build on AAETI the automation and control architecture for smart grids, studying also the agent communication issue not considered till now.

REFERENCES

[1] *Model-checking in dense real-time*, R. Alur, C. Courcoubetis, D.L. Dill, *Information and Computation* 104(1):2-34, 1993;
[2] *A theory of timed automata*, R. Alur, D.L. Dill, *Theoretical Computer Science* 126:183-235, 1994;
[3] *Automata Agents Extended with Time Constraints to deal with Smart Grid Control System Specification and Validation*, E. Corsetti, G. A. Guaglairdi, C. Sandroni, submitted for the revision to Journal of Software and System Modeling, for the special issue of Integrated Formal Methods, 2014;
[4] *Advanced Control Architectures for Intelligent MicroGrids – Part I: Decentralized and Hierarchical Control*, J.M Guerrero, M. Chandorkar, T. Lin Lee, P. C. Loh, IEEE Trans. on Industrial Electronics, vol. 60, n.4, pp 1254-1262, April 2013;
[5] *Microgrids and Active Distribution Networks*, S. Chowdhury, S.P. Chowdhury and P. Crossley, Published by The Institution of Engineering and Technology, London, United Kingdom, 2009;
[6] *Simulation of high-voltage Substations on Parallel Architectures*, Botti O., Cesana M. Corsetti E., Proceedings of the International Conference HPCN96 - High Performance Computing and Networking - Brussels, April 96.