

Performance assessment of an offshore windmill farm with AltaRica 3.0

Michel Batteux¹, Tatiana Prosvirnova², and Antoine Rauzy³

¹ IRT SystemX, Palaiseau, France. michel.batteux@irt-systemx.fr

² ONERA/DTIS, Université de Toulouse, Toulouse, France.
tatiana.prosvirnova@onera.fr

³ Norwegian University of Science and Technology, Trondheim, Norway.
antoine.rauzy@ntnu.no

Abstract. In this publication, we present how the AltaRica 3.0 modelling language can be used to efficiently design a model of an offshore windmill farm and evaluate its performance. The system we consider is composed of combinations of series-parallel components, combining different states for components and different modes for parts of the system and implements complex reconfiguration strategies.

Knowing the syntax and semantics of languages such as AltaRica 3.0 is however not sufficient to efficiently design models. First, models should make it possible to efficiently calculate performance indicators. Second, individual models should be designed quickly (and without bugs!) and modelling knowledge should be capitalized from models to models. With both respects, architectural and behavioural modelling patterns are of great help. The AltaRica 3.0 model we propose in this article for the assessment of an offshore windmill farm achieves both goals. We show that the design of the model is very efficient thanks to the advanced structural constructs of the AltaRica 3.0 modelling language. Finally, we use assessment tools available for AltaRica 3.0, e.g. the stochastic simulator, to evaluate the model of the system.

Keywords: AltaRica 3.0 · offshore windmill farm · production availability.

1 Introduction

In this article we study how to assess the production availability, over a given period of time, of an offshore windmill farm by means of AltaRica 3.0 [6]. Such an industrial production system is composed of several production lines, uses complex reconfiguration and maintenance strategies and so on. Furthermore, on the one hand the power production follows a demand based on houses and industries consumption, which depends on the seasons (spring, winter, autumn, summer) and the different parts of the day (morning, day, evening, night). On the other hand, the power production also depends on the force of the wind.

As of today, AltaRica 3.0 is probably the most advanced modelling language dedicated to probabilistic risk and safety analyses. AltaRica 3.0 results from the

combination of a powerful mathematical framework, guarded transition systems, and a versatile and coherent set of model structuring constructs stemmed from object- and prototype-oriented programming, S2ML [5]. Guarded transition systems provide the expressive power required for the analysis of such production systems [15, 3].

Knowing the syntax and semantics of languages such as AltaRica 3.0 is however not sufficient to efficiently design models. First, models should make it possible to efficiently calculate performance indicators. What is feasible in reliability engineering is actually over-determined by computational complexity issues, see [17] for an in-depth discussion. Second, individual models should be designed quickly (and without bugs!) and modelling knowledge should be capitalised from models to models. With both respects, architectural and behavioural modelling patterns are of great help. Modelling patterns can be thought as ways of organising the model, in a similar way design patterns are used to organise software, see [12] for a seminal book. The AltaRica 3.0 model we propose in this article achieves both goals.

Thus, in this publication, we show how the AltaRica 3.0 modelling language can be used to efficiently design models of production systems like an offshore windmill farm. The model combines the use of different modelling patterns: multi-state components, maintenance policies with shared resources, reconfiguration of the system taking into account the power demand and so on.

Finally, we use assessment tools available for AltaRica 3.0, e.g. the stochastic simulator ([2]), to evaluate performance indicators of the system.

The contribution of this publication is thus twofold: first, it shows how to efficiently design AltaRica 3.0 models of production systems; second it demonstrates the interest of AltaRica 3.0 advanced modelling constructs. Furthermore it continues the presentation of “how to model some features with AltaRica 3.0” started with [6] presenting several modelling patterns, [9] presenting a modelling pattern for phased-mission systems, [7] presenting the modelling of maintenance policies, and [8] presenting how to model large scale Markov chains with AltaRica 3.0.

The remainder of this article is organised as follows. Section 2 presents the case study, an offshore windmill farm that we use throughout the publication. Section 3 briefly presents the AltaRica 3.0 modelling language and its assessment tools. Section 4 explains how to model the case study with AltaRica 3.0. Section 5 provides the results of experiments with stochastic simulation. Finally, section 6 concludes the article.

2 Case study: an offshore windmill farm

In order to illustrate how the AltaRica 3.0 modelling language can be used to efficiently model and assess performance of large scale technical systems, we consider an offshore windmill farm depicted Fig. 1.

The system producing power is composed of five lines of five wind turbines WM1, WM2, ..., WM5 connected in series by cables C1, C2, ..., C5 to an

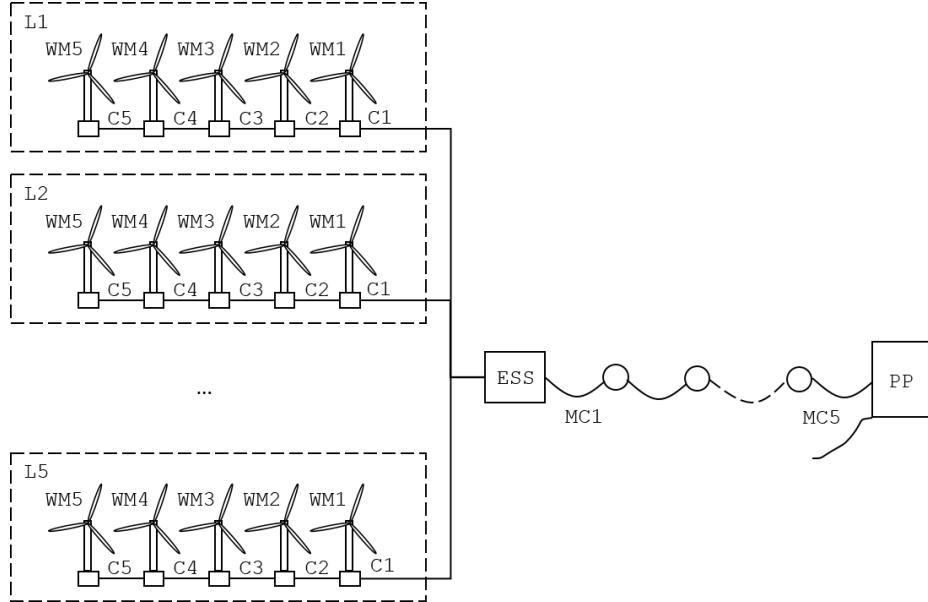


Fig. 1. An offshore windmill farm.

electrical substation ESS located at sea. The cables connecting the wind turbines to each other and the first wind turbine to the electrical substation ESS may be lost. When a wind turbine is out of service, the power can still be transmitted from the wind turbines located upstream to the substation ESS. If a cable is failed, it isolates the wind turbines located upstream. The substation ESS is itself connected to the power plant PP by a series of five cables, which can be failed.

We assume that the failures of wind turbines and cables follow exponential distributions with a failure rate $\lambda = 10^{-5}$. The wind turbines may fail when they are stopped with a failure rate $\lambda^* = 10^{-6}$.

There is a limited number of repairers. If a repair crew is available, the maintenance starts as soon as a line is failed. The end of the maintenance follows a uniform distribution with two parameters: $\alpha = 12h$ (start of the maintenance) and $\beta = 72h$ (end of the maintenance).

The power production of the windmill farm depends on the force of the wind and the power production demand. When the wind is too weak or too strong, the wind turbines do not produce power because they must be stopped. Otherwise, the power production depends on the force of the wind (in first approximation, we consider that it is a linear function).

The power production demand depends on the season of the year and the time of the day.

We would like to estimate the power production of the offshore windmill farm over a year and the difference between the power demand and the power production over a year.

3 AltaRica 3.0 modelling language and assessment tools

AltaRica 3.0 is a high level and stochastic event based modelling language, initially dedicated to the assessment of complex critical systems [6]. The language is based on the mathematical framework GTS (for Guarded Transition Systems [15]-[3]) to describe the behaviour of the system under study. The execution of an AltaRica 3.0 model is quite similar to other event-based formalisms. It means that when a transition is enabled, it is scheduled and will be potentially fired after its associated delay (see [10] and [18] for introductions of such executions of Discrete Event Systems). This behavioural part of AltaRica 3.0, based on GTS, is combined with a structural part named S2ML. S2ML stands for System Structure Modelling Language ([5]), and gathers in a coherent way structuring constructs stemmed from object-oriented programming, (see, e.g., [1]), and prototype-oriented programming, (see, e.g.,[13]).

The AltaRica 3.0 modelling language comes with a versatile set of assessment tools to design and evaluate models:

- The integrated modelling environment AltaRica Wizard ([4]), which provides the expected functionalities of a code editor and a project management;
- An interactive simulator to simulate, by hand, AltaRica 3.0 models;
- A compiler to fault trees in Open-PSA format ([14] and [11]), this compiler is chained with the fault tree engine XFTA ([16]);
- A generator of critical sequences of events leading from an initial state to failed states;
- Finally, a stochastic simulator ([2]).

The design of advanced AltaRica 3.0 models relies on the application of modelling patterns ([6]). The pattern-based approach in model-based safety assessment is strongly inspired from the corresponding approach in software engineering ([12]). Not only patterns make it possible to avoid the “blank page syndrome”, i.e. not to know where and how to start a model, but they unify modelling styles (alleviating maintenance tasks) and they prove to be a very good way to document and to share models.

4 Case study modelling and assessment

Fig. 2 shows the global architecture of the model. It is composed of the model of the environment, the model of the technical system under study and the observers.

Observers are quantities of the model that we would like to evaluate. Basically, in our example it is the power production and the difference between the power demand and the power production.

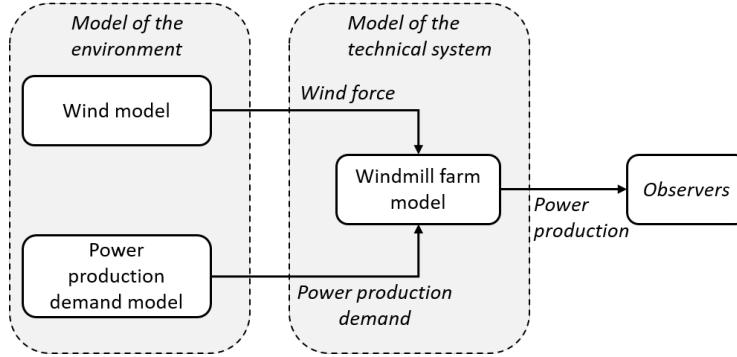


Fig. 2. Model structure diagram of an offshore windmill farm.

The model of the environment includes the simplified models of the wind and of the power production demand. The model of the wind provides the wind force to the windmill farm model. The model of the power production demand provides the value of the production demand or need to the technical system.

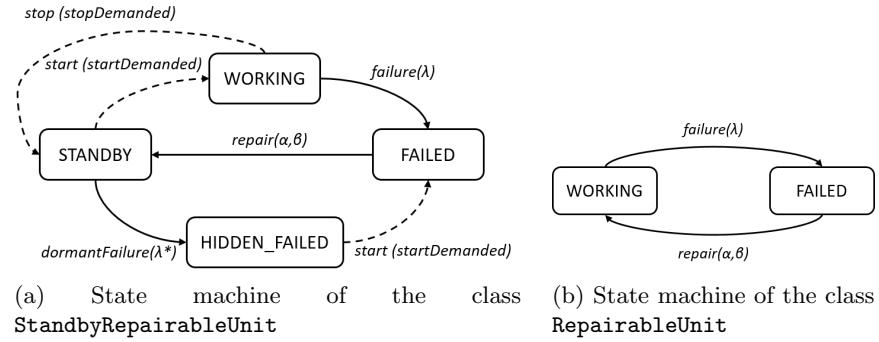
The model of the technical system includes the model of the windmill farm detailed below. It transmits the value of the power production to the observers.

4.1 Modelling the technical system

First, we start with modelling of the basic classes representing the behaviour of the wind turbines, the cables and the electrical station. Second, we assemble these classes to create the model of the lines. Finally, we define controllers to implement reconfiguration and maintenance strategies.

AltaRica 3.0 classes of the wind turbines and cables We model the behaviour of the wind turbines by a state machine `StandbyRepairableUnit` represented in Fig. 3a. Stochastic transitions are represented with plain arrows while the deterministic ones, labeled by the events `start` and `stop`, are represented with dashed arrows.

The AltaRica 3.0 model of the class `StandbyRepairableUnit` is given Fig. 4. It first defines a domain `UnitState` containing four values. It will be used to define the state of a `StandbyRepairableUnit`. Then this class is defined. An AltaRica 3.0 class is an on-the-shelf modelling component that can be instantiated as many times as necessary in the models. This class declares several elements: a state variable `vsState` of type `UnitState`, Boolean flow variables `vfStartDemanded` and `vfStopDemanded`, and several parameters and events. All these elements are used in the transition part to define the behaviour of a `StandbyRepairableUnit`, i.e. the changes of values of the state variable according to the occurrences of the events, as represented in Fig. 3a.

**Fig. 3.** behaviour of the generic classes

An AltaRica 3.0 transition starts with the name of the event, also called a label, then there is a guard (i.e. a Boolean condition on variables), and finally the action, which is an instruction that changes the value of (some of) the state variables. For example, in Fig. 4, the first transition defines the failure of the component. Its label is the event `evFailure`, which is associated with a delay obeying the inverse of a negative exponential distribution of parameter `pFailure`. To fire this transition the unit state must be working, so the guard imposes that the state variable `vsState` must be equal to the value `WORKING`. Finally, when the transition is fired the unit is failed, and the action sets the state variable `vsState` to the value `FAILED`.

A wind turbine may be started or stopped when it receives an order represented by flow variables `vfStartDemand` and `vfStopDemand`. It is modelled by two immediate events `evStart` and `evStop`. The transitions labeled by these events should be fired as soon as their guards become satisfied.

A wind turbine may also fail when it is in standby mode. It is represented by the event `evDormantFailure`, which is associated with a delay obeying the inverse of a negative exponential distribution of parameter `pDormantFailure`.

A wind turbine may be repaired. The event `evRepair` is associated with a uniform distribution with parameters `pStartRepair` and `pEndRepair`. The values of the parameters can be changed at will while performing experiments with models.

The AltaRica 3.0 model of the class `WindTurbine` is given Fig. 5. It extends the class `StandbyRepairableUnit` and defines other parameters, flow variables and an assertion. The production of a wind turbine depends on the wind force, which is represented by a flow variable `vfWindForce`. The dependency is expressed in the assertion. An assertion is an instruction which modifies the value of flow variables. It is executed after each transition firing.

The values of parameters (e.g. `pLowProduction`, `pNormalProduction`) are arbitrary. They can be changed while performing experiments with the model.

The behaviour of the cables and the electrical station is represented by a state machine given Fig. 3b. They may be failed and repaired. Their AltaRica 3.0

```

domain UnitState {WORKING, FAILED, HIDDEN_FAILED, STANDBY}

class StandbyRepairableUnit
    UnitState vsState (init = WORKING);
    Boolean vfStartDemanded (reset = false);
    Boolean vfStopDemanded (reset = false);

    parameter Real pFailure = 1.0e-5;
    parameter Real pDormantFailure = 1.0e-6;
    parameter Real pStartRepair = 12;
    parameter Real pEndRepair = 24;

    event evFailure (delay = exponential(pFailure));
    event evDormantFailure (delay = exponential(pDormantFailure));
    event evStart (delay = Dirac(0.0));
    event evStop (delay = Dirac(0.0));
    event evRepair (delay = uniform(pStartRepair, pEndRepair));

    transition
        evFailure: vsState == WORKING -> vsState := FAILED;
        evDormantFailure: vsState == STANDBY -> vsState := HIDDEN_FAILED;
        evStart: vsState == STANDBY and vfStartDemanded -> vsState := WORKING;
        evStart: vsState == HIDDEN_FAILED and vfStartDemanded -> vsState := FAILED;
        evStop: vsState == WORKING and vfStopDemanded -> vsState := STANDBY;
        evRepair : vsState == FAILED -> vsState := STANDBY;
    end

```

Fig. 4. AltaRica 3.0 class **StandbyRepairableUnit**.

```

class WindTurbine
    extends StandbyRepairableUnit;

    parameter Real pLowProduction = 40.0;
    parameter Real pNormalProduction = 70.0;
    parameter Real pHighProduction = 100.0;

    Real vfProductionOut (reset = 0.0);
    WindForce vfWindForceIn (reset = NULL);

    assertion
        vfProductionOut := if vsState == WORKING then
            (if (vfWindForceIn == NULL or vfWindForceIn == STORM) then 0.0
             else if vfWindForceIn == LOW then pLowProduction
             else if vfWindForceIn == NORMAL then pNormalProduction
             else pHighProduction) else 0.0;
    end

```

Fig. 5. AltaRica 3.0 class **WindTurbine**.

model is quite similar to the model of the wind turbines given Fig. 5. It uses the same principles: we, first, define a generic class and then a class which extends it, and define its flow variables and its assertion.

Modelling reconfiguration and maintenance strategies Fig. 6 shows the structural diagram of the block **WindmillFarm**. It is composed of 5 lines of wind turbines **Line1**, ..., **Line5**, an electrical station **ESS**, a series of cables **MC1**, ..., **MC5** and a global controller **Controller**.

The global controller is used to implement reconfiguration and maintenance strategies. It receives the diagnosis on the state of each line, the power production demand and the production of the lines. Based on these data, it sends the commands to start or to stop the line to the local controllers of each line.

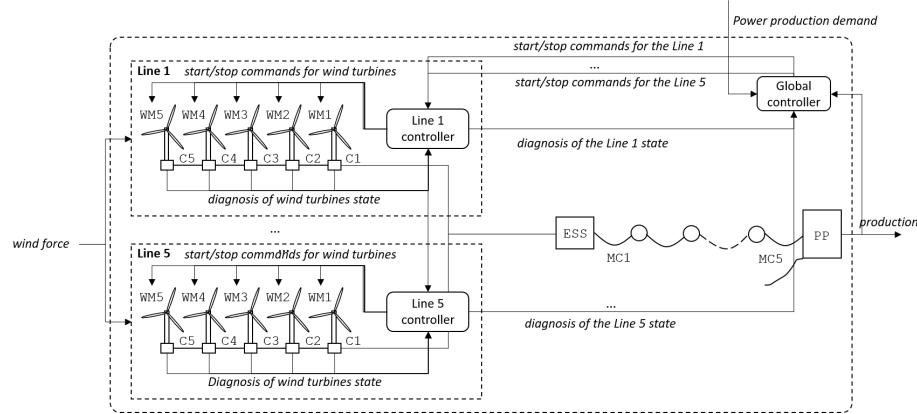


Fig. 6. Structure diagram of the block `WindmillFarm`.

We assume that there is a priority between the lines. The controller always starts with the Line 1, then if the demand is not satisfied, it starts the Line 2, and so on. If the demand gets lower, the controller first stops the Line 5, then the Line 4 and so on in the reverse order.

The global controller also implements the maintenance policy. There is a limited number of repairers. We assume that if a line is failed, and a repairer is available then the maintenance of the line can start immediately. The end of the maintenance is modelled by a uniform distribution with two parameters `pEndMaintenanceTimeMin` equal to 24 hours and `pEndMaintenanceTimeMax` equal to 72 hours. To implement this maintenance policy we use a modelling pattern presented in [7].

Modelling the lines The structure of the AltaRica 3.0 model of the class `Line` is given Fig. 7. Each line is composed of 5 wind turbines, 5 cables and a local controller. Connections between cables and wind turbines are defined in the assertion part. First, each wind turbine WM_i receives the value of the wind force. Second, each wind turbine WM_i is connected to its cable C_i . Finally, the production of the line `vfProductionOut` is calculated taking into account the fact that if a cable C_i is failed then the wind turbines located upstream $WM_{i+1}, WM_{i+2}, \dots$ are isolated.

The line receives the orders to stop and to start the wind turbines from the global controller. We assume that there is a priority between the wind turbines. First, the wind turbine WM_1 is attempted to start, then the wind turbine WM_2 , and so on. This priorities are implemented in the assertion part of the class `Line`, when the variables `WMi.vfStartDemanded` and `WMi.vfStopDemanded` are assigned.

The diagnosis part is also defined in the assertion. The value of the variables `vfNumberOfStoppedWM`, `vfIsFailed` and others are calculated in the assertion

```

class Line
// System composition
    WindTurbine WM1, WM2, WM3, WM4, WM5 (vsState.init = STANDBY, evRepair.hidden = true);
    Cable C1, C2, C3, C4, C5(evRepair.hidden = true);

// Flow input variables
    WindForce vfWindForceIn (reset = NULL);
    Boolean vfStartDemanded (reset = false);
    Boolean vfStopDemanded (reset = false);

// Flow output variables
    Real vfProductionOut (reset = 0.0);

// Flow output variables for diagnosis
    Integer vfNumberOfWorkingWM (reset = 0);
    Integer vfNumberOfFailedWM (reset = 0);
    Integer vfNumberOfStoppedWM (reset = 0);
    Boolean vfIsFailed (reset = true);

// Controller
    block Controller
// ...
    end

    assertion
// assertions to connect components
    WM1.vfWindForceIn := vfWindForceIn;
// ...
    C1.vfProductionIn := WM1.vfProductionOut;
// ...
    vfProductionOut := if not C1.vfWorking then 0.0
                        else if not C2.vfWorking then C1.vfProductionOut
                        else if not C3.vfWorking then C2.vfProductionOut +
                            C1.vfProductionOut
                        else if not C4.vfWorking then C3.vfProductionOut +
                            C2.vfProductionOut + C1.vfProductionOut
                        else if not C5.vfWorking then C4.vfProductionOut +
                            C3.vfProductionOut + C2.vfProductionOut + C1.vfProductionOut
                        else C5.vfProductionOut + C4.vfProductionOut +
                            C3.vfProductionOut + C2.vfProductionOut +
                            C1.vfProductionOut;
// assertions to start or stop wind turbines implementing the priority
    WM1.vfStartDemanded := vfStartDemanded;
    WM2.vfStartDemanded := vfStartDemanded and not (WM1.vsState == STANDBY);
// ...
    WM1.vfStopDemanded := vfStopDemanded and not (WM5.vsState == WORKING) and not
                        (WM4.vsState == WORKING) and not (WM3.vsState == WORKING) and not (WM2.vsState
                        == WORKING);
    WM2.vfStopDemanded := vfStopDemanded and not (WM5.vsState == WORKING) and not
                        (WM4.vsState == WORKING) and not (WM3.vsState == WORKING);
// ...
// assertions implementing the diagnosis
    vfNumberOfStoppedWM := #(WM1.vsState == STANDBY, WM2.vsState == STANDBY, WM3.vsState
                           == STANDBY, WM4.vsState == STANDBY, WM5.vsState == STANDBY );
    vfIsFailed := (WM1.vsState == FAILED or WM2.vsState == FAILED or WM3.vsState ==
                  FAILED or WM4.vsState == FAILED or WM5.vsState == FAILED
                  or C1.vsState == FAILED or C2.vsState == FAILED or C3.vsState ==
                  FAILED or C4.vsState == FAILED or C5.vsState == FAILED) ;
// ...
end

```

Fig. 7. AltaRica 3.0 class Line.

and sent to the global controller. Then these values are used to define global reconfiguration and maintenance strategies.

The block **Controller** is used to define when the maintenance can be started and stopped. It uses synchronisations as explained in [7].

4.2 Modelling the environment

Simplified model of the wind The wind force can be modelled by a discrete-time Markov chain.

In our model we assume that the wind force may be NULL, LOW, NORMAL, HIGH and STORM. When the wind force is NULL or STORM, the wind turbines must be stopped. The wind force changes every 4 hours. The discrete-time Markov chain modelling the force of the wind is given Fig. 8.

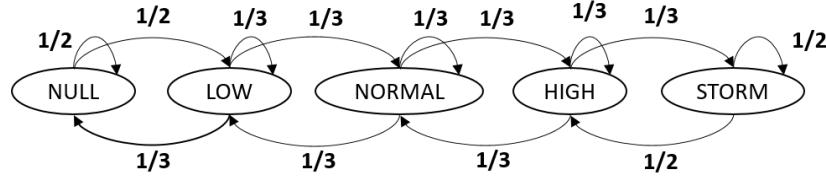


Fig. 8. Discrete-time Markov chain representing the wind force.

In AltaRica 3.0 this Markov chain is partially represented Fig. 9. First, we define a domain `WindForce`, which is an enumeration of five values: NULL, LOW, NORMAL, HIGH, STORM. Second, we define a class `Wind`. This class declares a state variable `vsWindForce`, which takes its values in the domain `WindForce`, to represent the internal state of the class, i.e. the wind force. It declares a flow variable `vfWindForceOut` which takes the same value. The parameter `pWindChangePeriod` is used to represent when the force of the wind changes. Its value can be changed to perform experiments. Events `evStayNull`, `evStayStorm`, ... and transitions represent how the state changes. The probability of the event is given by the attribute `expectation`. For instance, if the wind force is NULL, then there is a probability of 0.5 for the events `evStayNull` and `evIncreaseLow` to occur. The values of expectations may be declared using parameters and can be changed at will.

Of course, it is possible to use a more realistic model of the wind force taking into account real data coming from weather reports for a given region.

Model of the power production demand We assume that the power production demand depends on the season of the year and the time of the day. In our model the power production demand can be NULL, LOW, NORMAL or HIGH. Table 1 summarises how the power production demand depends on the season (spring, summer, autumn, winter) and the time of the day (morning, day, evening, night).

To represent this demand in AltaRica 3.0 we use a deterministic model. It can also be modeled by a discrete-time Markov chain by adding some uncertainties to the events.

```

domain WindForce {NULL, LOW, NORMAL, HIGH, STORM}

class Wind
    WindForce vsWindForce (init = NORMAL);
    WindForce vfWindForceOut (reset = NULL);

    parameter Real pWindChangePeriod = 4; // hours

    event evStayNull(delay = Dirac(pWindChangePeriod), expectation = 1.0/2.0);
    event evStayStorm(delay = Dirac(pWindChangePeriod), expectation = 1.0/2.0);
    event evIncreaseLow (delay = Dirac(pWindChangePeriod), expectation = 1.0/2.0);
    event evDecreaseHigh (delay = Dirac(pWindChangePeriod), expectation = 1.0/2.0);
    event evStay (delay = Dirac(pWindChangePeriod), expectation = 1.0/3.0);
    event evIncrease (delay = Dirac(pWindChangePeriod), expectation = 1.0/3.0);
    event evDecrease (delay = Dirac(pWindChangePeriod), expectation = 1.0/3.0);

    transition

// ...
    assertion
        vfWindForceOut := vsWindForce;
end

```

Fig. 9. AltaRica 3.0 model of the Wind.**Table 1.** Power production demand.

Heading level	Spring	Summer	Autumn	Winter
Morning (5a.m. - 9a.m.)	NORMAL	LOW	NORMAL	HIGH
Day (9a.m. - 4p.m.)	LOW	LOW	LOW	NORMAL
Evening (4p.m. - 10p.m.)	NORMAL	LOW	NORMAL	HIGH
Night (10p.m. - 5a.m.)	LOW	NULL	LOW	NORMAL

5 Experiments

As previously introduced in part 3, the AltaRica 3.0 modelling language comes with a versatile set of assessment tools to design and evaluate models. Thus for our specific offshore windmill farm case study, we use the stochastic simulator to perform evaluations. Of course, the modelling environment AltaRica Wizard, and the interactive simulator are used during the design and evaluation to get qualitative and quantitative information about the model.

Table 2 resumes the quantitative information about the model. We consider the designed AltaRica 3.0 model and the flattened one resulting from the compilation process. The main interpretation of these quantitative information is the huge difference between the number of models elements. In fact, when designing the AltaRica 3.0 model, we intensively used the powerful mechanisms of the language: the classes and the inheritance, the synchronisation of events, etc. For instance, the design of the fives lines, each one composed of five wind turbines and five cables is realised with different classes: a class for wind turbines with 5 atomic variables, a class for cables with 3 atomic variables, and a class Line (instantiating five wind turbines and five cables) with 8 atomic variables. The number of declared atomic variables is thus 16, whereas the real number of variables for all the lines, when the classes are instantiated, is 80.

Table 2. Quantitative information about the model.

AltaRica 3.0 model	Designed	Flattened
Number of lines	450	1260
Number of classes	5	-
Number of state variables	6	67
Number of flow variables	26	244
Number of parameters	25	175
Number of events	39	159
Number of transitions	46	190
Number of instructions in assertion(s)	60	244

Quantitative results are obtained with the AltaRica 3.0 stochastic simulator. We define different observers in the model and indicators to evaluate these observers. Then statistics are made on the indicators by the stochastic simulator. Fig. 10 defines the three main observers used to obtain the results. The two observers `oProd` and `oDiffProdNeed` are real observers. `oProd` gets the value of the production, whereas `oDiffProdNeed` gets the difference between the value of the production and the value of the need. Finally the observer `oNeedSatisfied` is Boolean and checks if the production is more than the demand, meaning “the demand is satisfied”.

```

block WindmillFarm
  block ProductionNeed
    // ...
  end
  block ProductionSystem
    // ...
  end
  // ...
  observer Real oProd = ProductionSystem.out;
  observer Real oDiffProdNeed = ProductionSystem.out - ProductionNeed.need;
  observer Boolean oNeedSatisfied = ProductionSystem.out >= ProductionNeed.need;
end

```

Fig. 10. AltaRica 3.0 observers.

We define indicators on these observers. For the two real observers `oProd` and `oDiffProdNeed`, we define mean-value indicators, meaning the mean value of the observer through the time period $[0, t]$, with t representing a time instant (the mission time for these experiments). For the Boolean observer `oNeedSatisfied`, it is a sojourn-time with the value `true`, meaning the time the observer had the value `true` from the time instant 0 to the time instant t (the mission time for these experiments). Then mean, standard deviation and confidence range are computed for these indicators.

The stochastic simulator produced 100000 runs for a mission time of 43800 hours (representing 5 years). It calculated statistics on indicators not only at the end of the mission time, but also at intermediate time instants: every year at

8760 hours, 17520 hours, 26280 hours and 35040 hours). Table 3 presents these results divided in two parts.

Table 3. Quantitative results.

Fired events	Mean	Min	Max
	128943.0	123381	134143
Indicator	Date	Mean	Standard Deviation
mean-value(oProd)	8760 17520 26280 35040 43800	770.871 770.562 770.457 770.391 770.384	18.7466 13.2972 10.868 9.38924 8.39527
mean-value(oDiffProdNeed)	8760 17520 26280 35040 43800	-434.722 -435.031 -435.137 -435.203 -435.209	18.7466 13.2972 10.868 9.38924 8.39527
sojourn-time(oNeedSatisfied)	8760 17520 26280 35040 43800	5100.25 10200.6 15300.6 20400.2 25501.0	135.047 191.6 234.779 271.06 302.954

The first part indicates the number of fired events: 128943 on average. This number seems high, but it can be explained by the fact that after each modification of the demand, starts or stops of wind turbines are fired. For instance, at the beginning of each simulation, the need is set to 2000 and with an initial wind force set to NORMAL, all the 25 wind turbines must be started, which means that 25 events are triggered.

The second part indicates statistics for the different indicators at every year. We only consider the mean and the standard deviation. What one can observe is that the need is satisfied only during 25501 hours for the five years (43800), and the difference between the production and the power demand is always negative (on average). To analyse these results, we can consider the Table 4 summarising the sojourn times for a given wind force and a maximum production (when the 25 wind turbines are started). We indicate in gray when the production is lower than the power demand during the 5 years. By summing all these sojourn-times in gray, we obtain the sojourn-time when the need is not satisfied. Thus it means that during a certain period of time (the sum of the sojourn-times) the wind force is not sufficient to satisfy the power demand with all the 25 wind turbines started.

Table 4. Sojourn times for a given wind force and a maximum production.

		Wind force				
		NULL	LOW	NORMAL	HIGH	STORM
Production	NULL	501.3	749.69	740.6	731.59	496.79
	LOW	3174.73	4774.95	4757.46	4738.58	3184.28
	NORMAL	2385.97	3548.67	3558.77	3565.07	2391.52
	HIGH	695.66	1039.13	1036.85	1036.18	692.17

6 Conclusion and perspectives

In this publication, we presented how the AltaRica 3.0 modelling language can be used to efficiently design models of large scale reconfigurable systems and to assess their performances using the stochastic simulation. The presentation was based on an example of an offshore windmill farm composed of several lines of wind turbines connected in series. We have shown that the advanced constructs of AltaRica 3.0 help to easily model several different features of such a system: combination of multi-states, resource sharing, reconfiguration, maintenance, etc. We calculated some indicators on this model using the stochastic simulator.

Modelling of these features involved a pattern based approach: resource sharing, limited number of repairers, reconfiguration, etc. This not only increases the set of already presented AltaRica 3.0 modelling patterns but it also confirms that modelling patterns are a very efficient mean to design models and, more fundamentally, they are a way to reason about the system under study.

Finally, the designed AltaRica 3.0 model can be easily extended so to introduce new elements, such as new lines or more repairers, or new features, such as different maintenance policies or more quantitative production values of the units according, for instance, to their degradation and failures. Different experiments may be conducted with different models of the wind force, of the power production demand, with different number of lines and different number of wind turbines per line, and of course with different values of parameters.

References

1. Abadi, M., Cardelli, L.: *A Theory of Objects*. Springer-Verlag, New-York, USA (1998)
2. Aupetit, B., Batteux, M., Rauzy, A., Roussel, J.M.: Improving performance of the AltaRica 3.0 stochastic simulator. In: Podofillini, L., Sudret, B., Stojadinovic, B., Zio, E., Kröger, W. (eds.) *Proceedings of Safety and Reliability of Complex Engineered Systems: ESREL 2015*. pp. 1815–1824. CRC Press (September 2015)
3. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica 3.0 assertions: the why and the wherefore. *Journal of Risk and Reliability* **231**(6), 691–700 (September 2017). <https://doi.org/10.1177/1748006X17728209>
4. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica wizard: an integrated modeling and simulation environment for altarica 3.0. In: *Actes du congrès Lambda-Mu 21 (actes électroniques)*. IMdR, Reims, France (October 2018)

5. Batteux, M., Prosvirnova, T., Rauzy, A.: From models of structures to structures of models. In: 4th IEEE International Symposium on Systems Engineering, ISSE 2018. Rome, Italy (October 2018)
6. Batteux, M., Prosvirnova, T., Rauzy, A.: Altarica 3.0 in 10 modeling patterns. *International Journal of Critical Computer-Based Systems* **9**(1–2), 133–165 (2019). <https://doi.org/10.1504/IJCCBS.2019.098809>
7. Batteux, M., Prosvirnova, T., Rauzy, A.: Modeling patterns for the assessment of maintenance policies with altarica 3.0. In: Papadopoulos, Y., Aslansefat, K., Katsaros, P., Bozzano, M. (eds.) *Model-Based Safety and Assessment*. LCNS, vol. 11842, pp. 32–46. Springer, Thessaloniki, Greece (2019)
8. Batteux, M., Prosvirnova, T., Rauzy, A.: Efficient Modeling of large Markov chains models with AltaRica 3.0. In: Proceedings of the 31st European Safety and Reliability Conference (ESREL). Angers, France (Sep 2021), <https://hal.archives-ouvertes.fr/hal-03429225>
9. Batteux, M., Prosvirnova, T., Rauzy, A., Yang, L.: Reliability assessment of phased-mission systems with altarica 3.0. In: Proceedings of the 3rd International Conference on System Reliability and Safety (ICSRs). pp. 400–407. IEEE, Barcelona, Spain (November 2018). <https://doi.org/10.1109/ICSRs.2018.00072>
10. Cassandras, C.G., Lafourche, S.: *Introduction to Discrete Event Systems*. Springer, New-York, NY, USA (2008)
11. Epstein, S., Reinhart, M., Rauzy, A.: The open psa initiative for next generation probabilistic safety assessment. In: Proceeding of 9th International Conference on Probabilistic Safety Assessment and Management 2008, PSAM 2008. vol. 1, pp. 542–550. IAPSAM, Hong-Kong, China (June 2008)
12. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley professional computing series, Addison-Wesley, Boston, MA 02116, USA (October 1994)
13. Noble, J., Taivalsaari, A., Moore, I.: *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, Berlin and Heidelberg, Germany (1999)
14. Prosvirnova, T., Rauzy, A.: Automated generation of minimal cutsets from altarica 3.0 models. *International Journal of Critical Computer-Based Systems* **6**(1), 50–79 (2015). <https://doi.org/10.1504/IJCCBS.2015.068852>
15. Rauzy, A.: Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability* **222**(4), 495–505 (2008). <https://doi.org/10.1243/1748006XJRR177>
16. Rauzy, A.: Anatomy of an efficient fault tree assessment engine. In: Virolainen, R. (ed.) *Proceedings of International Joint Conference PSAM'11/ESREL'12*. pp. 3333–3343. Helsinki, Finland (June 2012)
17. Rauzy, A.: Notes on computational uncertainties in probabilistic risk/safety assessment. *Entropy* **20**(3) (2018). <https://doi.org/10.3390/e20030162>
18. Zimmermann, A.: *Stochastic Discrete Event Systems*. Springer, Berlin, Heidelberg, Germany (2008)