Control Logic Encoding using RiskSpectrum ModelBuilder

Pavel Krcal[a], Helena Troili[a], and Ola Bäckström[a]
[a] Lloyd's Register, Stockholm, Sweden
{Pavel.Krcal, Helena.Troili, Ola.Backstrom}@lr.org

Abstract: Model-based safety assessment brings dependability modeling closer to the system design and allow for automated analysis of high-level models. RiskSpectrum ModelBuilder uses an object-oriented modeling language with elements of declarative programming called Figaro for describing the dependability logic of component types. We demonstrate how the Figaro language, and the concept of knowledge bases empowers dependability experts to create generic component descriptions that can be used in model-based safety assessment of various system types. In particular, we focus on the possibilities to encode complex control logic. One can create a single component type that can be utilized in systems with different topologies or in systems that contain different components.

1. INTRODUCTION

A goal of model-based safety assessment is to bring dependability modeling closer to the system design and allow for automated analysis of these high-level models. A system design description consists of system components and their relations. In many applications, a dependability model can copy the system design very closely. The dependability logic can be specified in a generic form per component type, applicable to all instances of this component type. A model might require a limited amount of specific, irregular, dependability information, such as relations or conditions affecting failures and their propagation. To prepare a model for an analysis, it remains to specify a configuration and safety/availability/production criteria.

RiskSpectrum ModelBuilder is a tool for model-based safety assessment based on the KB3 software, which has been developed by Électricité de France (EDF) since 1995 and grew into a platform proven by industrial use. Safety systems of a nuclear power plant can be modeled in KB3 so that their Probabilistic Safety Assessment (PSA) fault trees can be automatically generated from the KB3 model.

The modeling language for describing the dependability logic of component types used in RiskSpectrum ModelBuilder is called Figaro [5] and has evolved and matured over decades. It is an object-oriented modeling language with elements of declarative programming. It allows specifying interactions between components in the first-order logic. By this, a general description applies to all valid system topologies. The expressive power of this language has been demonstrated by numerous applications especially in the nuclear safety domain.

In this paper, we demonstrate how the Figaro language, and the concept of knowledge bases empowers dependability experts. It allows them to formalize and codify dependability knowledge for a specific domain or application type. It can be then used by non-experts in the form of a component library to build any model from this domain. The knowledge base can be systematically updated or extended whenever there is a need.

We focus on the possibilities to encode complex control logic in the component definitions. In general, one can specify any logic that can be described by a finite state machine or by a flow-chart. Communication between components, interactions between the state of a component and the state of related components and interleaving between stochastic events and control actions necessary for the control are also discussed.

We exemplify the power of Figaro on Digital I&C for Nuclear Power Plants, where the ModelBuilder approach allows to relatively easily extend the modeling to include intelligent voting. Automatic fault tree generation avoids the tedious and error-prone process of manual modeling for this complicated feature. We also develop main features of a control unit for a heterogenous power generating station scheduling different power sources to match the demand. As the last example, we consider a control of a Spent Fuel Pool that takes the water level in the pool into account. The latter two applications utilize Monte Carlo simulations for the analysis.

Apart from the RiskSpectrum ModelBuilder/KB3 and Figaro, there are several other established Model-Based Safety Assessment (MBSA) frameworks which offer a high-level modeling approach for dependability studies. AltaRica [1, 18] supports hierarchical modeling of components and their interaction. Models ultimately translate to guarded transition systems with a mathematical semantics and several analysis tools. Another MBSA framework, Safety Analysis Modeling Language (SAML) has been proposed by [13]. Hierarchically Performed Hazard Origin and Propagation Studies (Hip-HOPS) [16] equips individual components with failure modes and mechanisms of their propagation. This allows for standard Fault Tree Analysis [17] and Failure Modes and Effects Analysis. This formalism offers also dynamic analyses, e.g., based on Petri Nets [14]. Formal Safety Analysis Platform (FSAP) [12] utilizes symbolic model checking to perform safety analysis of a system model. This platform has been succeeded by xSAP [4] which extends it by providing general libraries for modeling and additional tools for safety analysis.

2. PRELIMINARIES

This section presents the modeling language Figaro and the concept of knowledge bases. Figaro is a text-based, object-oriented general-purpose modeling language for dependability analyses. Its semantics is a timed state-transition system with both deterministic and stochastic transitions. The language offers modelers great support to abstract away from low-level transition systems and think in terms of components and their interactions. Components can be defined as classes in a standard class-hierarchy used in object-oriented languages. A detailed description of all features can be found in [10]. Figaro can serve as a general language to describe standard formalisms used in dependability studies, such as Fault Trees, Markov Processes, Generalized Stochastic Petri Nets, Digraphs and Reliability Block Diagrams. It can also be used to build customized modeling libraries for various types of systems, such as thermohydraulic, electric, or production systems.

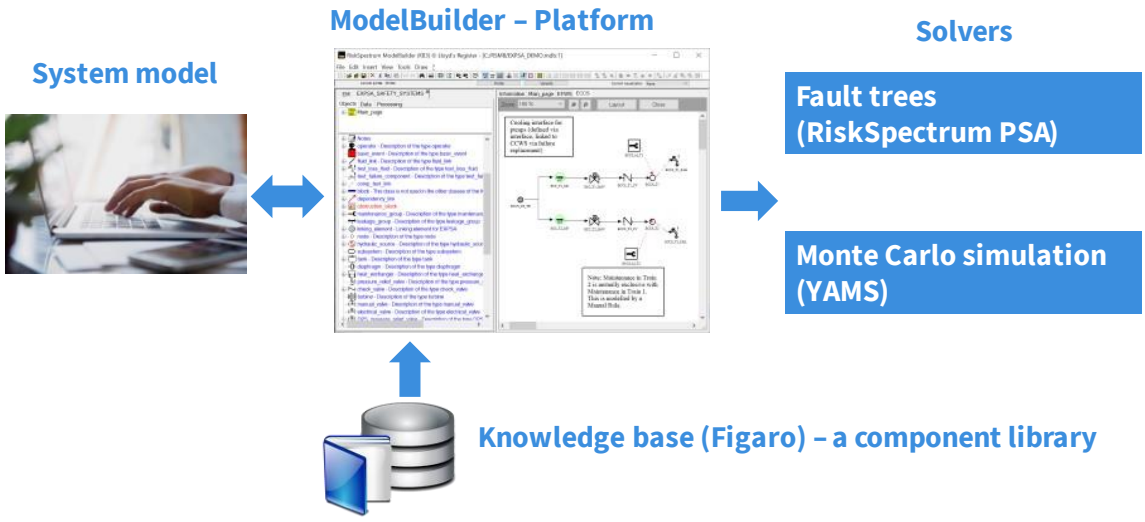
Knowledge bases consist of component descriptions (class definitions in the Figaro language) and a graphical part that turns components into graphical icons that enable graphical building of system models, their linking through graphical links or by setting instance specific properties and visualization of the component state. Defining a model then consists of selecting correct components, placing them on a canvas, linking them, and setting instance specific data. This also includes definitions of system configurations – initial states, relevant failure modes, and success criteria.

Models can be analyzed in various ways, depending on the types of behaviors encoded in the knowledge base [3, 11]. Failure occurrences that propagate through the system in a combinatorial way can be used to generate fault trees. More dynamic behavior patterns give rise to Markov Processes that can be analyzed by, e.g., Monte Carlo simulations. The structure of the ModelBuilder platform is depicted in Figure 1.

In Figaro libraries, classes may have many specific features that go beyond a combinatorial characterization of the failure behavior. For example:

- Repairs, including complex repair strategies with shared resources and imperfect repairs.
- Stand-by dependencies and multistage reconfiguration strategies.
- Deterministic behavior: clocks, grace periods, deterministic timing of failures.
- Production / processing / throughput

Figure 1. A schematic illustration of the ModelBuilder platform.



Our focus here is on another aspect of system modeling that can be described using Figaro, namely plant control logic. It is often necessary to capture decision schemes processing input and measurements and actuating plant components. Figaro offers powerful constructs that provide a reliability expert with a great flexibility to include control logic in a knowledge base.

Figaro classes describe component behavior with the following constructs:

- Interfaces: related components
- Occurrence rules: events that can occur stochastically and influence the component state
- Interaction rules: propagation of state changes among components

Interfaces define relationships between components by specifications in the Figaro classes. The specifications define a kind which provides the name of the related class but are otherwise flexible; The knowledge base does not have to specify the number of related components of this kind. The number can then be from zero to many. The interfaces are inherited between Figaro classes as other component properties.

Occurrence rules capture stochastic events in the model. An occurrence is conditioned by a certain state and can occur randomly according to a defined distribution or dependent on time the system has been in that state. An occurrence has consequences on the state of the component described by actions and by firing of transitions. Figaro supports various standard time distributions, including also a fixed period between events. Events can be observed by other components and update the state of the system.

Interaction rules are a declarative way to model communication, discrete control logic, and changes to the component. Various control logic corresponding to a flow chart, or a state machine structure can be implemented by imperative iteration of condition rules in a specified order. In an interaction rule of a component, it is possible to extract information from, as well as updating state of other components that are linked through interfaces. The rules can perform first order quantification and iterate over related components extracting required information. In general, the interaction rules are executed in rounds until a fixed-point is reached. However, they can be defined in different steps, that decide the grouping and sequential ordering of the execution of the rules. The order of the steps is specified in the steps order section of the knowledge base. Interaction rules of the same step are run in top-down order. Interaction rules are executed at any occurrence of an event.

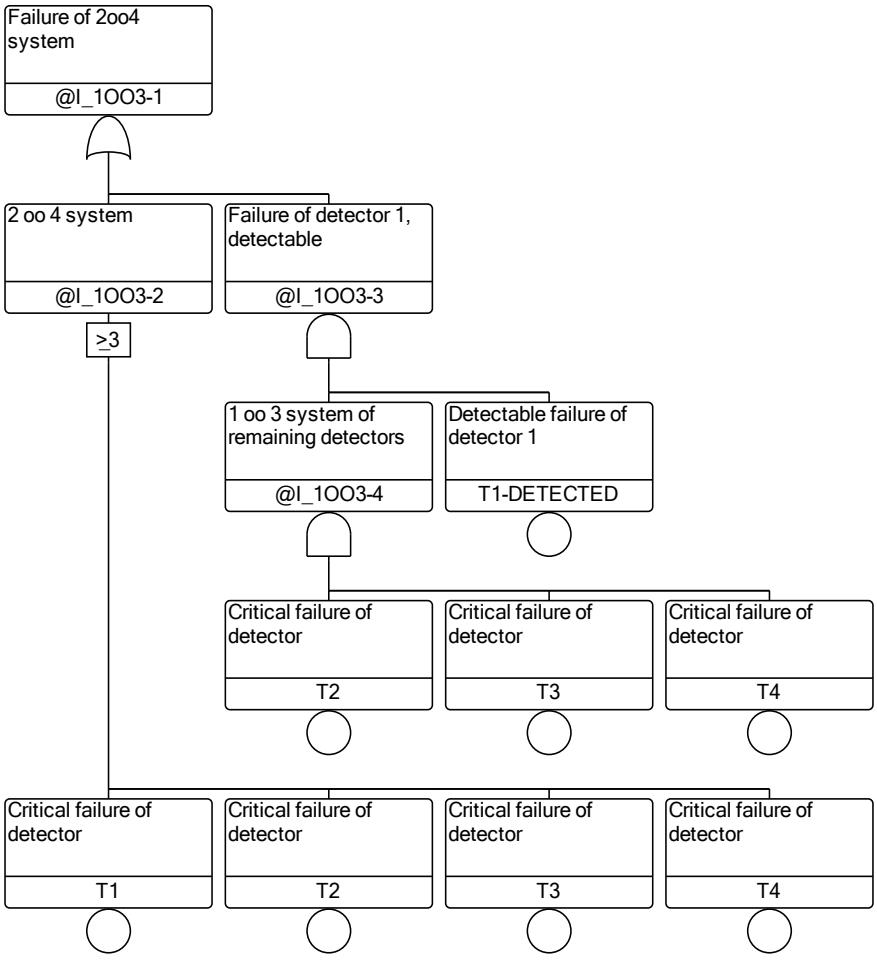
3. DIGITAL I&C: INTELLIGENT VOTING

Safety systems of a nuclear power plant can be modeled using knowledge bases for thermohydraulic, electric and Instrumentation&Control (I&C) parts of the design. Fault trees can be automatically generated from high-level graphical models closely corresponding to Piping and Instrumentation Diagrams (P&IDs) or Single-Line Diagrams. This section presents control logic modeling in such setup, aiming at fault tree analysis.

Digital I&C systems belong to common components in modernizations and building of new reactors, regardless of type. There are challenges with representation of such systems in reliability assessment, both in terms of hardware and software. In this paper we focus on one of the positive aspects of digital I&C, "intelligent voting". The digital I&C enables management of detected faulty input signals with a possibility to respond according to the detected faulty input. Such "intelligence" can be difficult to model fully in PSA and simplifications are often made.

An example of intelligent voting is a system with the 2-out-of-4 logic in the initial state for detection of low water level. Now assume that a failure of one of the transmitters is detected (signal out of bound, for example). Furthermore, let us assume that the faulty signal is triggering a high water level indication. In this case the system would, without intelligent voting, become a 2-out-of-3 system. With an intelligent voting, the system designer could say that the system should switch into a 1-out-of-3 system in case of a detected fault. The fault tree in Figure 2 is indicating how this can be modelled. Note that only detected failure of detector 1 is represented.

Figure 2. A fault tree for a switch from 2-out-of-4 to 1-out-of-three in an intelligent voting system.



The configuration, the type of failure modes and the system behavior triggered by a detected failure that you may like to introduce in the model of digital I&C may differ from system to system, and hence it could be interesting to have a knowledge base that would allow for your own definition of how the system should behave in different types of situations. The knowledge base fragment in Figure 3 is based on the example above. The first interaction rule covers the situation with no detectable failure. The voting logic can be specified in the model, individually for each voting system. If we set the value of the attribute *detectors_requires_all_OK* to be equal to two, then we obtain 2-out-of-4 logic for four detectors. The condition in the second interaction rule is true if and only if exactly one detector has a detectable failure. In this case, the voting logic is determined by the attribute *detectors_required_one*_failed. If we set it to one, then the resulting logic will be 1-out-of-3. A fault tree generated from this knowledge base would be logically equivalent to the fault tree in Figure 2. Analogically, one can define interaction rules for higher numbers of detectable failures in the voting system. These interaction rules can be also easily extended to cover other types of failure modes or other types of intelligent voting algorithms and hence such a knowledge base provides an extended flexibility to the modelling of Digital I&C systems.

Figure 3. Example interaction rules for an intelligent voting system.

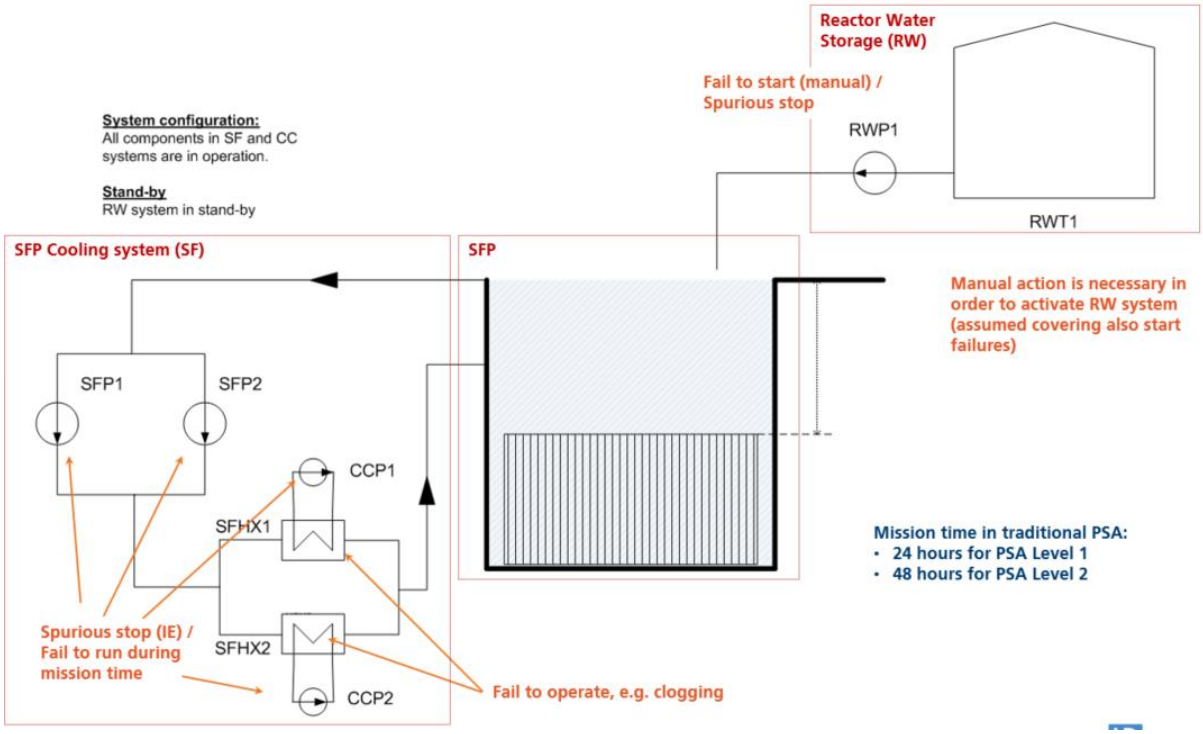
```
(* All detectors functioning - original voting *)
IF NOT (THERE_EXISTS detector INCLUDED_IN linked_detectors
        SUCH_THAT detectable_failure(detector))
   AND NOT (THERE_EXISTS AT_LEAST detectors_required_all_OK(voting_system)
            detector INCLUDED_IN linked_detectors
            SUCH_THAT NOT critical_failure(detector))
THEN
    voting_system_loss ;

(* One detector defect - degraded voting defined in the voting system *)
IF (THERE_EXISTS detector INCLUDED_IN linked_detectors
        SUCH_THAT detectable_failure(detector) AND
            FOR_ALL other_detector INCLUDED_IN linked_detectors
            SUCH_THAT other_detector <> detector WE_HAVE
            NOT(detectable_failure(other_detector)) )
   AND NOT (THERE_EXISTS AT_LEAST detectors_required_one_failed(voting_system)
            detector INCLUDED_IN linked_detectors
            SUCH_THAT NOT critical_failure(detector))
THEN
    voting_system_loss ;
```

4. SPENT FUEL POOL: A HYBRID ANALYSIS

A Spent Fuel Pool (SFP), depicted in Figure 4, contains active systems for cooling the stored fuel which has a relatively low decay heat. In case of a failure of the cooling systems, a radioactive release does not occur early, i.e., within 24 or 48 hours. The time until the water starts boiling is approximately one week. Once boiling has started it will take approximately three more weeks before the fuel elements are uncovered. Within this time, cooling system components can be repaired, fail again, or other components can fail later during the accident sequence.

A realistic safety assessment of a spent fuel pool using static fault trees is challenging. Dynamic methods based on a minimal cut set list might be applied to reduce the conservatism of the static approach [15]. Another alternative would be to resort to Monte Carlo simulations which can take the dynamics of the physical phenomenon and its interaction with the plant into account. This includes also an interaction between the discrete control logic and physical phenomena that occur in the plant. We show how this can be implemented in a ModelBuilder knowledge base on an example of the water level and temperature control.

Figure 4. A scheme of a spent fuel pool.



The spent fuel pool cooling system (SF) pumps water from the top of the pool, cools it in heat exchangers and feeds it back to the pool. If the water temperature reaches a certain level (T_c), the SFP system stops functioning. To simplify the example, we set T_c equal to the boiling temperature. When the water temperature reaches T_c, then the Reactor Water system (RW) is started. It feeds cold water to the pool until the temperature drops below a certain level T_n and the water level in the pool is restored (in case some water has already boiled off). If the cooling fails completely, then the water boils off and eventually the water level drops so that the fuel is uncovered. In this situation, we have reached a state where the undesired consequence occurs. The control logic of the RW system is depicted in Figure 5.

Modeling patterns in Figaro. We illustrate modeling of the interactions between the water temperature, the water level, and the plant, where a control unit decides when the RW system is started or stopped. The water level and temperature are also affected by component failures and repairs. Evolution of the plant is determined by occurrence of events. We add an object that issues a clock tick event which occurs repeatedly with a fixed constant period. By this, we discretize time in the Figaro model [7]. The state of continuous variables (water level, water temperature) is then calculated at each event (either a stochastic failure or repair, or a clock tick) from the state at the previous event, plant state, and the amount of time passed since the previous update. Each event also leads to an update of the discrete variables by the means of interaction rules. This gives the control unit a possibility to react on changed values of continuous variables.

Figure 6 shows the rule for an update of the pool temperature and water level. If we set the date of the last update, which is stored in the last_event_date variable, to the current date, obtained from CURRENT_DATE, after executing the temperature and water level update then we execute the update only once after each new event.

Figure 5. A generic flow chart describing the control logic of the Reactor Water system.

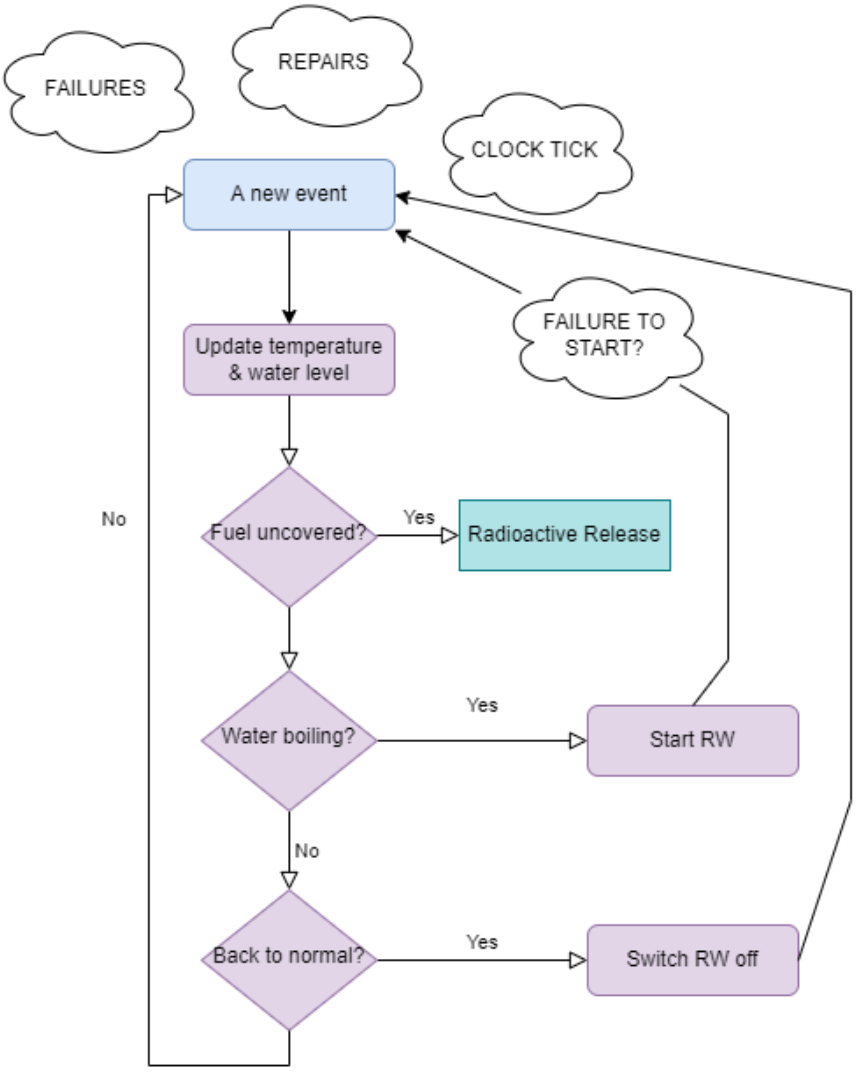


Figure 6. Interaction rules for updating the spent fuel pool temperature and level after each event. Setting the flag water_boiling as the last rule remembers if the water has been boiling for the whole time interval.

```
(* A simple linear approximation of the heat exchange *)
IF last_event_date <> CURRENT_DATE
THEN T <-- MIN( T + (heat(spent_fuel) - heat_removal(cooling)) *
           (CURRENT_DATE - last_event_date), 100 );

(* If the water is boiling since the last event then update the water level *)
IF water_boiling = TRUE AND T = 100 AND last_event_date <> CURRENT_DATE
THEN L <-- MAX( L - boil_off_factor * (CURRENT_DATE - last_event_date), 0 );

(* RW system is filling the pool with water if running *)
IF state(RW) = 'RUNNING' AND last_event_date <> CURRENT_DATE
THEN L <-- MIN( L + flow_rate(RW) * (CURRENT_DATE - last_event_date), L_MAX );

IF T = 100
THEN water_boiling <-- TRUE
ELSE water_boiling <-- FALSE;
```

The plant controller can then react upon the change in the temperature and start additional cooling – the Reactor Water system – when the temperature rises above a certain level. By this, the heat removal capacity of the plant cooling and the water level increase. When the temperature drops under a certain level and the water is refilled then the RW system can be switched off again. The controller sends the corresponding signal to the RW system. Figure 7 shows how switching the RW system on and off is implemented by setting the state of the RW system to 'required' or to 'stand-by'. If RW succeeds to start, then its state changes to RUNNING. This is defined in the interaction rules of the RW system. If the plant is back to a normal state then the controller switches off the RW system.

An analysis of this spent fuel pool model based on Monte Carlo simulations can estimate the frequency of uncovering the fuel for scenarios defined by specific plant configurations, reliability data, repair strategies, etc. This type of analysis considers the grace delay between the cooling failure and the consequence in a way that is much more realistic than a static analysis with a fixed mission time. It includes additional scenarios also when compared to dynamic methods considering repairs and grace delays, such as Initiators and All Barriers [2, 8, 9]. The price to pay is the analysis time required for simulations and the lack of exhaustiveness, which is offered by dynamic methods based on minimal cut sets.

Figure 7. Interaction rules of a controller that requests a start of the RW system when the temperature reaches the critical level.

```
(* Send a start signal to the RW system if the
temperature reaches the critical level*)
IF T(pool) >= T_critical AND state(RW) = 'STAND_BY'
THEN state(RW) <-- 'REQUIRED';

(* Stop the RW system if the temperature is OK
and the pool is full *)
IF T(pool) <= T_normal AND L(pool) = L_MAX(pool)
                       AND state(RW) = 'RUNNING'
THEN state(RW) <-- 'STAND_BY';
```

5. HETEROGENOUS POWER PRODUCTION: PRODUCTION PRIORITIES

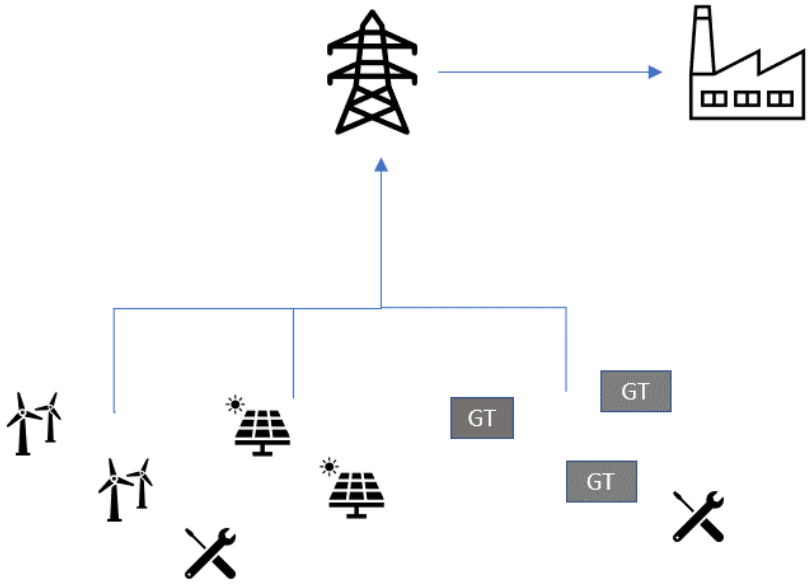
This section presents modelling of the control logic of a heterogenous power production system with different sources of electricity supply. We show how complex control structures can be encoded in a generic knowledge base written in the Figaro language. Models using this knowledge base can be analyzed by Monte Carlo simulations in order to make optimal design decisions for such power plants.

The system to model consists of the components presented in Figure 8:

- a consumer representing the electricity demand function that varies over time
- a set of renewables, i.e., wind turbines and solar power plants producing power depending on weather
- a set of backup gas turbines
- a power station controlling the production and connecting all power production systems

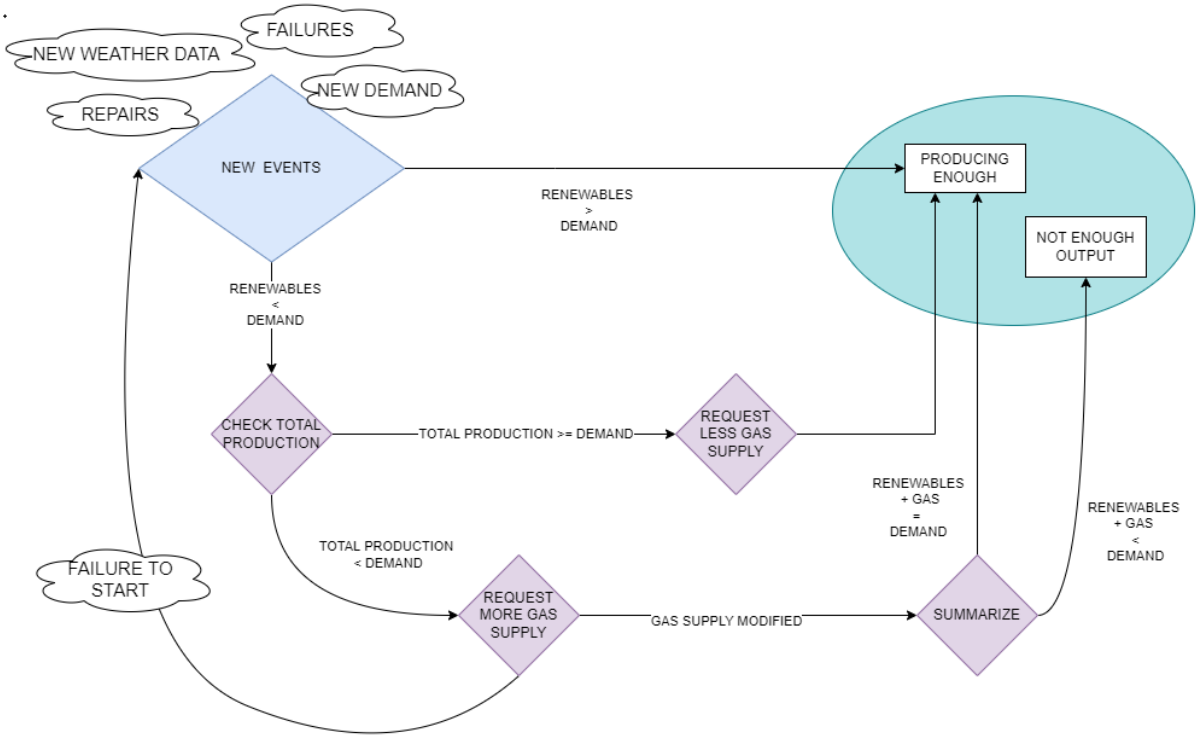
The power station aims to ensure sufficient energy production so that it can meet the demand at minimal costs (minimizing financial costs, CO_2 production, etc.). Many design decisions can be taken to achieve this; How to minimize starting and running gas turbine backups? What mean repair times and priorities between different components are allowed regarding availability versus repair costs?

Figure 8: Power production system example



For simplicity, we show a limited functionality in this example. It is also possible to include different energy storages into the model. This can be implemented using time discretization, similar to the Spent Fuel Pool example, i.e., charging at overproduction and supply when needed. This part aims to present how a control flow can be implemented. Figure 9 captures the control logic of the power station in a flow chart.

Figure 9: A generic flow chart of control logic of the power station



Dynamics of the whole system are determined by changing weather conditions and consumer demand, as well as occurrences of failures and their repairs. At each new event, a new consumer demand is given, and a supply level from all wind- and solar production systems depending on weather and a possible

failure of components. The power station uses this information to control the energy production from the gas turbines.

We must consider interleaving of stochastic events, such as failures, communication between components, and discrete control decisions. The decision logic in the state “CHECK TOTAL PRODUCTION” in Figure 9 can be implemented as a series of interaction rules. First, calculating the variable *current_output* is essential for determining the current supply, see Figure 10. This is done by iterating all related linked backup turbines using the FOR_ALL statement.

Figure 10: Summing up of current output from all existing electricity suppliers

```
(* Sums up the existing output from renewables and previous
gas production to find level to request from gas *)
station_sum_current_supply
STEP inp_renewables_gas GROUP Simu_group
THEN current_output <-- inp_renewables
                (SUM FOR_ALL x A backups OF_TERMS current_output(x));
```

The value of the current output is compared to the demand to determine the change of gas turbine production. Increasing the production is represented by the interaction rules station0 and station1 in Figure 12. The step order specified in this knowledge base lets the *inp_renewables_gas* step in Figure 10 be executed before the *request/decrease_backups* steps in Figure 11 and the step station0 before station1. This ensures that the capacity of running backup turbines is used before requesting a new one to start up.

Figure 11: The logic for decision on whether to request a start-up of a backup gas turbine

```
(* when the output drops too much then try to increase production
from running backups *)
station0
STEP request_backups GROUP Simu_group
GIVEN x A backups
IF current_output ...

(* when the output drops too much then start one backup *)
station1
STEP request_backups GROUP Simu_group
GIVEN x A backups
IF current_output + expected_new < demand AND state(x) = 'standby'
THEN state(x) <-- 'required', (* Raises an ocurrence for start-up *)
requested_output(x) <--MIN(maximal_output(x), (demand - (current_output + expected_new))),
expected_new <-- expected_new + requested_output(x);

(* when output from renewables is sufficient then switch off a backup*)
station2
STEP decrease_backups GROUP Simu_group
GIVEN x A backups
IF current_output - current_output(x) >= demand AND state(x) = 'producing'
THEN state(x) <-- 'standby',
current_output <-- current_output - current_output(x);
...
```

When requesting a change, the station rules in Figure 11 also set the state of linked backup gas turbines by using state(x). These turbine states are in turn used in the gas turbine class, where its output is set accordingly, respecting their own properties and resources, see Figure 12.

Figure 12: Code snippet of gas turbine class with the interactions run in conjunction with the rules changing the instance's state:

```
CLASS gas_turbine EXTENDS prod_system;
        ATTRIBUTE
        requested_output DOMAIN REAL DEFAULT 10;
        state DEFAULT 'standby';

        INTERACTION
        (* If a gas turbine produces it will provide requested but not more than its maximum *)
        gas1
        STEP default_step decrease_backups GROUP Simu_group
        IF state = 'producing'
        THEN current_output <-- MIN( requested_output, maximal_output );

        (* If a gas turbine is not producing, nothing is provided *)
        gas2
        STEP default_step decrease_backups GROUP Simu_group
        IF NOT(state = 'producing')
        THEN current_output <-- 0;
```

The different objects also have different stochastic events depending on the state they are in, represented by occurrences. Figure 13 shows that a failure on demand of a gas turbine in the state required can occur. Analogically, a failure in operation of any power producing unit can occur. All these stochastic events trigger a new evaluation of interaction rules, i.e., starting from top in Figure 9, following the step order until a steady state of the system is reached, which also determines the power production level.

Figure 13: Occurrences for sub classes to prod_system

```
OCCURRENCE
(*  Occurrences for prod systmes, i.e. renewables like wind, sun, and
 gas turbines. *)

fail_in_op
GROUP Simu_group
IF state = 'producing'
MAY_OCCUR
FAULT fail DIST EXP(fail_rate)
INDUCING state <-- 'failed',
FOR_ALL r A rep DO rank <-- max_rank(r) + 1 ;

fail_to_start
GROUP Simu_group
IF state = 'required'
MAY_OCCUR
FAULT fail DIST INS(fail_start_prob)
INDUCING state <-- 'failed',
FOR_ALL r A rep DO rank <-- max_rank(r) + 1
OR_ELSE
TRANSITION startup
INDUCING state <-- 'producing' ;
```

7. CONCLUSION

Model-based safety assessment encapsulates dependability expert knowledge and provides a dependability or system engineer with a high-level tool to build and analyze a system dependability model. In case of RiskSpectrum ModelBuilder (KB3), this tool is a knowledge base offering a list of component types defined by a dependability expert in the modeling language Figaro. A faithful modeling of complex systems might also require including control logic in a knowledge base. We have shown on several examples that Figaro presents a flexible and powerful way of describing control logic and its interaction with the plant and the environment.

References

[1] Arnold, A. Griffault, G. Point and A. Rauzy. "*The AltaRica formalism for describing concurrent systems*", Fundam. Inf. 40, issue 2-3, pages 109-124, IOS Press, (2000).

[2] Bäckström, M. Bouissou, R. Gamble, P. Krcal, J. Sörman, and W. Wang, "*Introduction and Demonstration of the I&AB Quantification Method as Implemented with RiskSpectrum PSA*", Proc. of PSAM'14, (2018).

[3] Bäckström, M. Bouissou, P. Krcal, and P. Wang, "*Flexibility of Analysis Through Knowledge Bases*", Proc. of ESREL'21, (2021).

[4] Bittner B., Bozzano M., Cavada R., Cimatti A., Gario M., Griggio A., Mattarei C., Micheli A., and Zampedri G. (2016). "*The xSAP Safety Analysis Platform*". Proc. of TACAS'16.

[5] Bouissou M., Bouhadana H., Bannelier M., Villatte N. "*Knowledge modeling and reliability processing: presentation of the Figaro language and associated tools*", Proc. of SAFECOMP'91. (1991).

[6] Bouissou M., Bon J. L. "*A new formalism that combines advantages of fault-trees and Markov models: Boolean logic driven Markov processes*". Reliability Engineering & System Safety 82, 149-163, (2003).

[7] M. Bouissou, "*Comparison of two Monte Carlo schemes for simulating Piecewise Deterministic Markov Processes*", In Proc. of MMR'07, (2007).

[8] M. Bouissou and O. Hernu. "*Boolean approximation for calculating the reliability of a very large repairable system with dependencies among components*", Proc. ESREL 2016, Glasgow, (2016).

[9] M. Bouissou. "*Extensions of the I&AB method for the reliability assessment of the spent fuel pool of EPR*" ESREL 2018, Trondheim, June 2018.

[10] Bouissou, M., Humbert, S. and Houdebine, J. "*Reference manual for the FIGARO probabilistic modelling language (Version-E)*", EDF, (2019)

[11] Bouissou, M., Khan, S., Katoen J.-P., and Krcal P. "*Various Ways to Quantify BDMPs*". In Proc. of MARS'20, (2020).

[12] Bozzano, M., Villafiorita, A. "*The FSAP/NuSMV-SA safety analysis platform*". International Journal on Software Tools for Technology Transfer (STTT) 9, 5-24, (2006).

[13] Güdemann M., Ortmeier F. "*A framework for qualitative and quantitative model-based safety analysis*". Proc. of HASE'10, (2010).

[14] Kabir S., Walker M., and Papadopoulos Y. "*Dynamic system safety analysis in HiP-HOPS with Petri Nets and Bayesian Networks*". Safety Science 105:55-70, (2018).

[15] [Olsson] A. Olsson, "*Leaving mission times backstage and taking repair into account in long term scenarios*", Probabilistic Safety Assessment and Management PSAM 12, Honolulu, Hawaii, (2014).

[16] Papadopoulos Y., McDermid J. "*Hierarchically performed hazard origin and propagation studies*". In Proc. of SAFECOMP'99, (1999).

[17] Papadopoulos Y, Maruhn M. "*Model-based automated synthesis of fault trees from Matlab-Simulink models*". Proc. of International Conference on Dependable Systems and Networks (DSN'01), (2001).

[18] Point G., Rauzy A. "*AltaRica: Constraint Automata as a Description Language*", Journal Européendes Systémes Automatisés 33(8–9):1033–52, (1999).