Dynamic probabilistic risk assessment at a design stage for a sodium fast reactor.

V. Rychkov and H.Chraibi

EDF R&D, EDF Lab Paris-Saclay, Palaiseau, France

E-mail contact of main author: valentin.rychkov@edf.fr

Abstract. A Sodium Fast Reactor is a good candidate for an application of dynamic probabilist risk assessment due to long mission times and possible system recoveries. In this paper, we present a new approach to construct the dynamic probabilistic model of the Decay Heat Removal (DHR) function of a Sodium Fast Reactor. We use Distributed Stochastic Hybrid Automata concept with the PyCATSHOO modeling tool. The proposed approach allows to construct a dynamic probabilistic model of a SFR DHR function that incorporates the time evolution of physical parameters and dynamic changes in the states of DHR systems (failure or recovery of DHR system components) and presents enough flexibility to be applied at the design stage.

Key Words: Dynamic PSA, DHR modeling, PyCATSHOO, Python

1. Introduction

The design of a new reactor is an iterative process. At a design stage, each reactor system may exist in several variants : combined together, these variants give different reactor designs. Probabilistic risk assessment (PRA) is a tool that may help to compare different reactor designs. Conservative or macroscopic way to perform probabilistic risk assessment at a design stage may be insufficient to provide enough information to distinguish between different plant design variants [1].

The classical way to construct a PRA model with boolean Event Trees/Fault trees (ET / FT) is well applicable for a PWR type reactors at a design stage. Nevertheless ET/FT formalism finds its limits for a Sodium Fast Reactor if one considers long mission times and possible system recoveries. Due to thermal inertia and simplicity of thermal-hydraulic behavior, the Decay Heat Removal (DHR) function of a Sodium Fast Reactor (SFR) is a good candidate for dynamic probabilistic risk assessment [2, 3].

In this paper, we present a new approach to construct the dynamic probabilistic model for the DHR function of a Sodium Fast Reactor with PyCATSHOO modeling tool. PyCATSHOO is a tool dedicated for dependability analysis of hybrid systems [4, 5, 6], i.e. systems where deterministic continuous phenomena and stochastic discrete behaviour are equally important [7].

The paper is organised as follows: we introduce PyCATSHOO and its modeling capabilities in more details, then we introduce an example of an SFR DHR systems that perform the DHR function, we present the PyCATSHOO model of the DHR systems and simulation results.

2. PyCATSHOO modeling tool

The safety requirement of its nuclear and hydraulic fleet, has allowed EDF to have long-standing experience in using and developing PRA and PSA tools of complex systems. PyCATSHOO is one of the tools developed over last few decades at EDF R&D.

PyCATSHOO implements the concept of Piecewise Deterministic Markov Process (PDMP) using distributed stochastic hybrid automata. The principles of this paradigm are described in details in [4]. Back there, PyCATSHOO was in a prototype stage whereas a nearly industrial version is currently in use at EDF to perform several safety studies. During 2017, EDF is going to release PyCATSHOO under a freeware licence.

PyCATSHOO is a C++ written library and has two APIs (Application Programming Interfaces) in Python and C++. These APIs provide a set of tools, based on distributed hybrid stochastic automata to model and assess the complex hybrid systems. PyCATSHOO combines the power of Python language[1] and LEGO type modelling approach.

A hybrid stochastic automaton may exhibit random transitions between its states according to a predefined probability law. It may also exhibit deterministic transitions governed by the evolution of physical parameters.

One can summarize a modelling approach with PyCATSHOO as follows:

- A system is divided in a functional, or any other way, into elementary subsystems, components.
- Each of elementary subsystem, component is described as a set of hybrid stochastic automata, state variables and message boxes.
- Message boxes ensure message exchanges between subsystems, components and hence ensure their dependencies.
- The system behaviour is simulated using Monte Carlo sampling.
- Sequences that lead to desirable end states are traced and clustered.

PyCATSHOO offers a very flexible modeling framework that allows to define generic components (classes) of hybrid stochastic automata that can be reused in different reliability studies. It can greatly reduce model development costs and thus make Dynamic PRA accessible at the design stage.

3. Decay Heat Removal system of a SFR

We consider a pool type SFR with three different DHR systems: an active DHR system S1 with two trains, a passive (based on the natural circulation) system S2 with two trains, and a third Reactor Vault Auxiliary Cooling System (RVACS) S3, through the main vessel. The primary sodium mean temperature T_m evolution as a function of time can be described by Eq.(1):

$$MC_p \frac{dT_m}{dt} = P_{res}(t) + P_{pumps}(t) - N_{S1}P_{S1}(T_m) - N_{S2}P_{S2}(T_m) - N_{S3}P_{S3}(T_m) \quad (1)$$

where MC_p is the thermal inertia of the sodium pool, P_{res} is the residual power of the reactor, P_{pumps} is the power produced by the pumps. N_{SX} is the number of SX trains available. Performance of each SX train $P_{SX}(T_m)$ is a known function of the pool temperature and possibly other parameters [3].

4. Modeling of a DHR function with PyCATSHOO

As we discussed above, to design a system with PyCATSHOO we define generic classes that describe different elements of the model. For the needs of this case study we create the set of the PyCATSHOO classes that are described below.

4.1. Generic component

aComponent class describes the behaviour of a generic DHR component (a pump, a heat exchanger, a valve) that can be in an operating or in a failed state. It includes failure on demand, failure to operate and recovery transitions, see Listing 1.[2]

```
1 import Pycatshoo as Pyc
2 class aComponent(Pyc.CComponent):
3     def __init__(self, name, pycSystem,train,*args,**kwargs):
4       #States
5       self.addState("S")   #S=STARTING
6       self.addState("O")   #O=OPERATING
```

[1] Python is a high level open-source programming language. An interested reader can easily learn Python following one of the free tutorials on the internet like http://www.diveintopython.net

[2] Here and in the following listings, a green text starting with a hash-tag is a comment.

```
        self.addState("F")  #F=FAILED
        #Default Transition parameters
        #on-demand failure probability
        self.gamma=self.addPortOut("gamma",Pyc.VarType.double,0.01)
        #failure to operate frequency
        self.lmbda=self.addPortOut("lmbda",Pyc.VarType.double,0.00001)
        #recovery rate mu
        self.mu=self.addPortOut("mu",Pyc.VarType.double,0.000001)
        #Transitions:
        #On-demand failure transitions:
        tr=self.getState("S").addTransition("SOF")
        tr.addTarget(self.getState("F"),Pyc.TransType.fault)
        tr.addTarget(self.getState("O"),Pyc.TransType.trans)
        tr.setDistLaw(Pyc.IDistLaw.newLaw(self,Pyc.TLawType.inst,self.gamma))
        #Failure to operate transition
        tr=self.getState("O").addTransition("O_F")
        tr.addTarget( self.getState("F"),Pyc.TransType.fault)
        tr.setDistLaw(Pyc.IDistLaw.newLaw(self,Pyc.TLawType.expo,self.lmbda))
        #Component recovery transition
        tr=self.getState("F").addTransition("F_O")
        tr.addTarget( self.getState("O"),Pyc.TransType.rep)
        tr.setDistLaw(Pyc.IDistLaw.newLaw(self,Pyc.TLawType.expo,self.mu))
```

Listing 1: Definition of **aComponent** class

4.2. Generic system train

aTrain class describes a generic SFR system train. It can be a DHR train, a support system train, or I&C train. From an engineering point of view a train is an assembly of the components that is capable to perform a specific function: cooling (a DHR system trains, ventilation system trains), AC power supply (normal or emergency), or system control (I&C). For simplicity we assume that all train components are necessary to perform the train function (no redundancy inside a train). Listing 2 shows an example of implementation of **aTrain** class.

```
class aTrain(Pyc.CComponent):
    def __init__(self,name,pycSystem,,**kwargs):
        Pyc.CComponent.__init__(self,name,pycSystem)
        self.components=[] #should contain pointers to system components
        self.support=[] #support systems needed for a train to function
        self.systemName=name.split('_')[0]
        self.pool=[]
        if kwargs:
            for key in kwargs:
                if key=='components':
                    for componentName in kwargs[key]:
                        self.components.append(aComponent(componentName,
  pycSystem,self))
                if key=='support':
                    self.support.extend(kwargs[key])
                if key=='pool':
                    self.pool.append(kwargs[key])
        #Train states:
        self.addState("N") #Non-operating
        self.addState("O") #Operating
        #Train transitions:
        tr=self.getState("N").addTransition("N_O")
        tr.setConditionFct("logicalFunction",self.logicalFunction,False)
        #The transition to the operating state happens when logicalFunction
    takes "True" value
```

```
        tr.addTarget(self.getState("O"),Pyc.TransType.trans)
        tr=self.getState("O").addTransition("O_N")
        tr.setConditionFct("logicalFunction",self.logicalFunction,True)
        #The transition to non-operating state happens when logicalFunction() takes "False" value
        tr.addTarget(self.getState("N"),Pyc.TransType.trans)
        #The train is operational only if all the components and support systems are operational
    def logicalFunction(self):
        state=True
        for component in self.components:
            state=state*component.state()
        for supSystem in self.support:
            state=state*supSystem.state()
        return bool(state)
```

Listing 2: Definition of **aTrain** class

4.3. Primary circuit

The **Pool** class implements the Eq.(1) that governs the evolution of the primary sodium temperature as a function of number of available DHR trains. Listing 3 shows an example of implementation of the **Pool** class. The pool is in *OK* state if the temperature is below maximum sodium temperature e.g. $650C$, otherwise we consider a core damage and transition into a *CD* ("Core Damage") state. Instead of temperature criteria one can easily implement a Larson-Miller type of damage criteria [8].

```
import Pycatshoo as Pyc
class poolClass(Pyc.CComponent):
    def __init__(self, name, system):
        Pyc.CComponent.__init__(self, name, system)
        # Declaration of differential equation within PyCATSHOO:
        self.PDMP_name = "poolMeanTemp"
        self.system().addPDMPManager(self.PDMP_name)
        # Ordinary Differential equation
        self.PDMP_ODE_name = "ODE"
        self.addODE(self.PDMP_name, self.PDMP_ODE_name, self.ODE, 0)
        # add the variable that will be tracked by PDMP controller
        self.addODEPort(self.PDMP_name, self.meanTemp)
        self.addState("OK")
        self.addState("CD")
        self.setInitState("OK")
        tr=self.getState("OK").addTransition("OK_CD")
        #The condition of the core damage
        tr.setConditionFct("coreDamage",self.coreDamage,False)
        tr.addTarget(self.getState("CD"),Pyc.TransType.trans)
        tr.setDistLaw(Pyc.IDistLaw.newLawVal(self,Pyc.TLawType.inst,1))
    def coreDamage(self):
        if self.meanTemp.dValue()> 650: #meanTemp>650C
            return True
        else: return False
```

Listing 3: Definition of the **Pool** class

The following part (see Listings 4) of the **Pool** class contains different functions (the residual power as a function of time, performance of SX trains) to integrate the differential Eq. (1) over time.

```
    def ODE(self):
        # Here we define the differential equation PyCATSHOO will integrate
        # self.po_NSX - is the number of operational SX trains
        self.meanTemp.setDvdtODE(
```

```
            self.Pres() - self.po_NS1.dValue() * self.S1() - self.po_NS2.
    dValue() * self.S2() - self.po_NS3 * self.S3())
    def Pres(self):
        # residual heat power as a function of time
        return Residual_Heat_at_time_t
    def S1(self):
        # Power evacuated by a S1 train
        return power_S1
    def S2(self):
        # Power evacuated by a S2 train
        return power_S2
    def S3(self):
        # Power evacuated by a S3 train
        return power_S3
```

Listing 4: **Pool** class, differential equation and residual heat data

4.4. Common Cause Failures

aCCF class implements a Binomial Failure Rate Common Cause Failure model [9]. The details of this implementation are not relevant for the purpose of this paper.

5. An example of a model for DHR function

Above we defined generic classes for a generic component, a generic system train, the evolution of physical parameters inside the primary circuit as function of available SX trains, and Common Cause Failures. In this section we show how one can construct architectures of different DHR systems based on these classes.

To construct a model of DHR function, we assume that different trains of the same DHR system (S1 or S2 or S3) are consisted of identical components. The components of the trains may have or may not have common failure modes. For example, in Listing 5, active components of the DHR trains like Electromagnetic Pump (PEM) of S1 system or Opening vent of S2 system or a regular Pump of S3 system do have common failure modes. And passive components like Heat Exchangers (HEX) do not have common failure modes.

```
import Pycatshoo as Pyc
from SFR_CCF import *    #Import of Common Cause failure model
from SFR_Pool import *   #Import of the Pool class
from SFR_Train import *  #Import of the aTrain class
from copy import deepcopy
class SFR(Pyc.CSystem):
    def __init__(self):
        Pyc.CSystem.__init__(self, "SFR") #name of the system Class
        self.pool = poolClass("Pool", self) #name of the poolClass instance
        #list of components for each system
        self.CCFcomponentsS1=["PEM"]
        self.CCFcomponentsS2=["VENT"]
        self.CCFcomponentsS3=["PUMP"]
        self.componentsS1=["HEX"]
        self.componentsS2=["HEX"]
        self.componentsS3=["HEX"]
```

Listing 5: Composition of different SX trains

We construct each train of a SX system separately. Listing 6 shows an example of the model of S1 trains one and two. They have the components with and without common cause failures, they are connected to the primary pool to extract residual heat and they depend on the electrical **`self.systemELECx`** system trains one and two.

```
        # S1 system train one
        self.toS11['components']=deepcopy(self.componentsS1)
        self.toS11['components'].extend(deepcopy(self.CCFcomponentsS1))
        self.toS11['pool']=self.pool
        self.toS11['support']=self.ELEC1
        # S1 train two
        self.toS12['components']=deepcopy(self.componentsS1)
        self.toS12['components'].extend(deepcopy(self.CCFcomponentsS1))
        self.toS12['pool']=self.pool
        self.toS12['support']=self.ELEC2
        #S1 system that has two trains
        self.S1=[]
        self.S1.append(aTrain("S1_1",self,**self.toS11))
        self.S1.append(aTrain("S1_2",self,**self.toS12))
```

Listing 6: Construction of S1 system with the dependencies

In the same way we can construct S2, S3 and ELEC systems.

Finally, we create CCF groups for the components that have common failure modes. We use Binomial Failure Rate model [9] to represent Common Cause Failures.

5.1. Launching simulations

In order to simulate the behaviour of DHR function, we create an instance of **SFR** class (see List. 5), load a parameter file and launch a parallel simulation. We span the simulation runs on all available cores. Different HPC architectures can be used to perform these simulations in an efficient way.

```
def sysExec():
    system = SFR()
    system.loadParam("poolTest.xml")
    system.simulate()
    if system.MPIRank() > 0:
        exit(0)
if __name__ == '__main__':
    try:
        sysExec()
    except Exception as e:
        print(type(e), e)
        Pyc.printMessages()
```

Listing 7: Simulation of the model

5.2. Design Variants

The scripting way to create system trains, tracking of the dependencies and common cause failures (see Listing 6) together with the coupling to the simple thermalhydraulic model of the primary circuit (Listings 3,4) allows easy study of design variants.

To change the composition of a *SX* train, one need to change the list **`self.componentsSX`**; to change the performance of DHR trains one has to modify **`def SX(self)`** functions; to change components in CCF groups one has to modify **`self.CCFcomponentsSX`** lists.

6. Example of a simulation results

To launch a simulation with PyCATSHOO, we need to fix the reliability parameters of different system components.

The parameter file **"poolTest.xml"** (see an example in listing 8) contains different reliability data e.g. failure rates, failure probabilities on demand, common cause failure parameters etc.

```
<PY_PARAM
      NAME="MyParameters"
      TRACE="1" SEQ_NB="100"
      TMAX="200000"
      RES_FILE="out.txt">
   <PRT NAME="Pool.convectionFactor"       INITV="0.02"/>
   <PRT NAME="S1_1_PEM.gamma"        INITV="0.01"/>
   <PRT NAME="S1_1_PEM.lmbda"        INITV="0.0001"/>
</PY_PARAM>
```

Listing 8: Parameter file of a PyCATSHOO model

The simulated sequences of a DHR function with PyCATSHOO contain events that lead to the core damage, evolution of temperature as a function of time. The probability of the core damage is extracted from simulated sequences.

Figure 1 shows the time evolution of primary sodium temperature for 100 different sequences. Those sequences that reach 650C are considered as core damage sequences. We can note the importance of component recoveries. There are sequences that do not reach the core damage state because some of DHR trains were recovered.

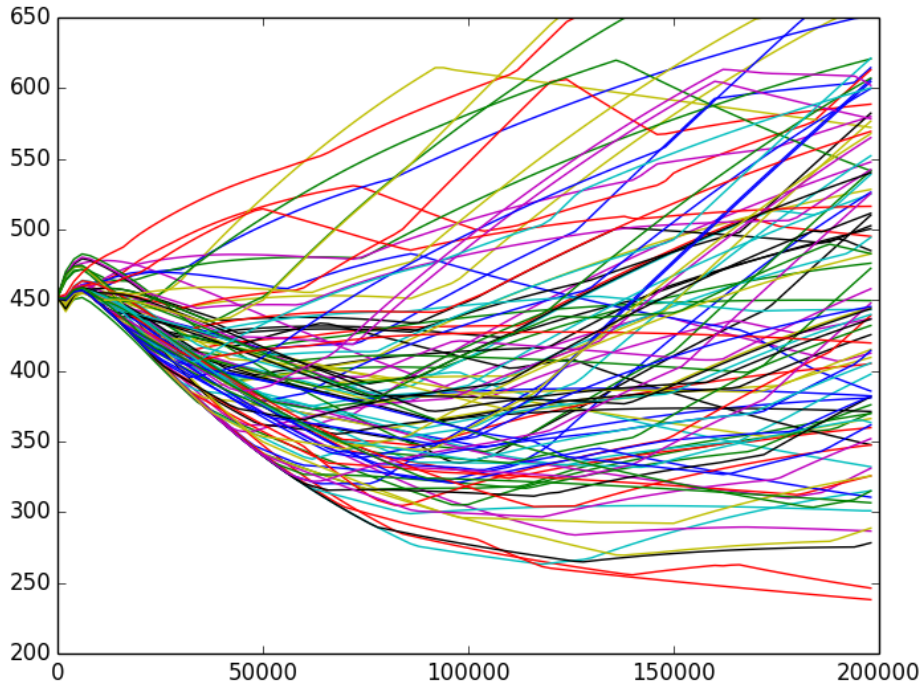

FIG. 1: *An example of time evolution of primary sodium temperature (y-axis in Celsius) as a function of time (x-axis in seconds) for a set of simulated sequences.*

Table I (see next page) contains a sequence that leads to the core damage. This sequence involves a non-lethal CCF event. The table I provides the time stamps of the events (transitions) as well as corresponding components names. E.g. at 21808.9 sec, the pump of train 2 of S3 system failed etc.

The similar (in chronology) sequences can be clustered. The probability of a macro-sequence (the cluster of sequences with the similar chronology of events) is the ratio between the number of the sequences in the cluster and the total number of simulated sequences. The total probability of core damage is the sum of probabilities of macro-sequences.

TABLE I: An example of the sequence that leads to the core damage.

n	Average times-tamp	Name	Final state	type	law
1	12731.5	S2_2_VENT.O_F	F	def	exp
2	21808.9	S3_2_PUMP.O_F	F	def	exp
3	38631.3	S3_1_PUMP.O_F	F	def	exp
4	39585.2	S2_1_VENT.O_F	F	def	exp
5	49361.6	S1_2_PEM.O_F	F	def	exp
6	58138.6	S2_3_VENT.O_F	F	def	exp
7	63260.1	CCF_S2_VENT.ON_NL	ONL	tr	exp
8	131373	S1_1_PEM.O_F	F	def	exp
9	190357	Pool.OK_CD	CD	tr	inst

7. Conclusions

This paper briefly describes an approach to create a dynamic model of an SFR DHR system using PyCATSHOO tool.

We can treat support systems, dependencies, Common Cause Failures, and physical phenomena inside the same model. The scripting way to create the model allows us to easily study different variants of DHR architecture to support the design of an SFR.

Acknowledgments

Authors would like to acknowledge M. Marquès and F. Aubert for fruitful discussions, and Sodium Fast Reactor R&D project of CEA that have supported this work.

References

[1] GAUTHE P. et al. "Use of simplified PSA studies in support of the ASTRID design process" (Proc. FR13, Paris, 2013)

[2] CURNIER, F. et al., "Symbiosis of static and dynamic probabilistic approaches to support the design process and evaluate the safety of SFR", (Proc. PSA 2015, , Sun Valley ID).

[3] MARQUES M. et al. "Consideration of Physical Behavior and Possibility of Repair in the Long Term Reliability Evaluation of Decay Heat Removal Systems", (Proc. PSA 2015, , Sun Valley ID).

[4] CHRAIBI H., "Dynamic reliability modeling and assessment with PyCATSHOO: Application to a test case", (Proc. Probabilistic Safety Analysis and Management, Tokyo, 2013).

[5] CHRAIBI H. et al. "PyCATSHOO:Toward a new platform dedicated to dynamic reliability assessments of hybrid systems", (Proc. PSAM 13, Seoul, 2016).

[6] BROY P., et al. "A new methodology to model and assess reliability of large dynamic hybrid systems", (Proc. of Mathematical Methods in Reliability (MMR) , Stellenbosch, South Africa 2013).

[7] RYCHKOV V., KAWAHARA K., "PSA Level 2 with dynamic event trees. Lessons learned and perspectives", (Proc PSA 2015, Sun Valley ID).

[8] https://en.wikipedia.org/wiki/Larson-Miller_Parameter

[9] ATWOOD C. L. "The binomial failure rate common cause model", ,Technometrics, 28(2), 139-148, (1986).