Utilisation du standard FMI (Functional Mock-up Interface) avec PyCATSHOO - Application en fiabilité dynamique

Hassane Chraibi, Jean-Christophe Houdebine, Claudia Picoco, Rychkov Valentin

▶ **To cite this version:**

Hassane Chraibi, Jean-Christophe Houdebine, Claudia Picoco, Rychkov Valentin. Utilisation du standard FMI (Functional Mock-up Interface) avec PyCATSHOO - Application en fiabilité dynamique. Congrès Lambda Mu 23 " Innovations et maîtrise des risques pour un avenir durable " - 23e Congrès de Maîtrise des Risques et de Sûreté de Fonctionnement, Institut pour la Maîtrise des Risques, Oct 2022, Paris Saclay, France. hal-03878545

HAL Id: hal-03878545
https://hal.science/hal-03878545v1

Submitted on 29 Nov 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Institut pour la Maîtrise des Risques
Sûreté de Fonctionnement - Management - Cindyniques

Utilisation du standard FMI (Functional Mock-up Interface) avec PyCATSHOO Application en fiabilité dynamique

Use of the FMI (Functional Mock-up Interface) standard with PyCATSHOO Use in dynamic reliability

CHRAIBI Hassane
EDF
7, avenue Gaspard Monge
91120 Palaiseau
hassane.chraibi@edf.fr

PICOCO Claudia
EDF
7, avenue Gaspard Monge
91120 Palaiseau
claudia.picoco@edf.fr

HOUDEBINE Jean-Christophe
ARISTE
2 Rue du Marché,
92160 Antony
jean-christophe.houdebine@wanadoo.fr

RYCHKOV Valentin
EDF
7, avenue Gaspard Monge
91120 Palaiseau
valentin.rychkov@edf.fr

***Résumé* —** L'outil PyCATSHOO met en œuvre les processus Markoviens Déterministe par Morceaux sous forme d'automate stochastiques hybrides. Son but est de permettre la réalisation d'études probabilistes qui mixent des modèles stochastiques discrets et des modèles physiques déterministes continus. Bien que cet outil offre les fonctionnalités nécessaires à la réalisation de ces derniers, le couplage avec des modèles physiques externes existants reste une nécessité. Pour y répondre nous avons choisi d'y intégrer le standard FMI (Functional Mock-up Interface). Cet article rend compte de cette intégration par une illustration basée sur le cas d'étude bien connu du HeatedTank. Il montre ainsi que la plateforme PyCATSHOO est prête pour une mise en œuvre en cosimulation au format FMI, ce qui permet d'élargir grandement le champ de son utilisabilité.

Mots-clefs — PDMP, PyCATSHOO, FMI

***Abstract*—** PyCATSHOO is a tool that implements the Piecewise-Deterministic Markovian processes as distributed hybrid stochastic automata. It is aimed at probabilistic evaluations that mix discrete stochastic models with deterministic physical models. Although this tool offers functionalities for the conception of the latter, the ability to couple discrete stochastic models with existing external physical models is still required. To meet this requirement, we have integrated the FMI (Functional Mock-up Interface) standard into PyCATSHOO. This paper reports on this integration by an illustration based on the well-known case study of the HeatedTank. It shows that the PyCATSHOO platform is ready for co-simulation according to the FMI standard, which greatly improves its usability.

Keywords — PDMP, PyCATSHOO, FMI

I. INTRODUCTION

Le besoin d'intégrer les phénomènes physiques dans les modèles dédiés à l'évaluation de la sûreté de fonctionnement de certains systèmes complexes n'est plus à démonter. Ce besoin se manifeste entre autres dans les études probabilistes de sûreté nucléaires. Dans ce domaine, plusieurs méthodologies ont été mises au point. Comme le montre l'inventaire qui en a été fait dans [1], ces méthodes s'inspirent principalement des approches classiques basées sur les arbres de défaillances et arbres d'événements dont elles étendent le formalisme pour tenir compte de la dynamique du système étudié et des phénomènes physiques thermo-hydrauliques qui s'y déploient. Ces méthodologies reposent sur des variantes des arbres d'événements dynamiques et, pour la plupart d'entre elles, prennent en considération le temps de manière discrète. Quant aux méthodes avec temps continu, cet inventaire qui date de 2013, en cite principalement deux et fait état de leur limitation. La première est la technique du « continuous cell-to-cell-mapping (CCCMT) » décrite dans [2]. Cette technique ne considère que des changements instantanés lorsque, par exemple, un point de consigne est atteint. La seconde a été mise au point par [3]. Elle introduit la notion

de stimulus qui constitue le préalable à une action et donne la possibilité de tenir compte d'un délai, aléatoire ou déterministe, avant son exécution.

Depuis, EDF a mis au point une méthodologie qui contribue à la levée des limitations actuelles des approches probabilistes de sûreté qui intègrent des modèles physiques déterministes. Cette méthodologie est actuellement implémentée dans l'outil PyCATSHOO [4]. En plus de comporter tous les ingrédients qui en font un outil de modélisation et d'évaluation probabiliste de systèmes discrets, cet outil dispose de fonctionnalités évoluées pour concevoir et simuler des modèles 0D/1D. Il offre également des outils qui facilitent l'intégration et la synchronisation des comportements discrets stochastiques et des comportements déterministes et continus.

Si ces modèles sont performants en raison de l'étroite intégration de ces deux types de comportement, il est important de permettre la réutilisabilité de modèles physiques externes, lorsqu'ils existent. C'est le cas aussi bien dans le domaine nucléaire que d'autre domaines.

La réutilisation par PyCATSHOO de modèles physiques externes est déjà possible et a été mise en œuvre dans [5]. Elle bénéficie de l'ensemble des fonctionnalités offertes par cet outil dont la capacité de simulation parallèle. Elle nécessite cependant un couplage manuel dédié qui doit être conforme à son schéma de fonctionnement. C'est pourquoi nous avons ajouté à ses fonctionnalités des capacités de cosimulation qui répondent au standard FMI (Functional Mock-up Interface).

Nous allons, dans ce papier faire un bref rappel des fondements théoriques de l'outil PyCATSHOO et de leur mise en pratique. Nous allons ensuite passer en revue les principales questions posées par la problématique de couplage avec un code physique dans une modélisation destinée à une évaluation probabiliste.

Dans la section 4, nous présenterons brièvement le standard FMI et, en particulier, son volet cosimulation.

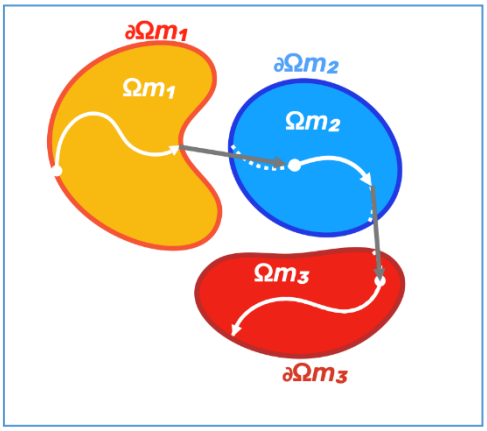
Avant d'exposer la mise en œuvre de la cosimulation au standard FMI dans PyCATSHOO, nous donnerons dans la section 5 une description du cas d'étude HeatedTank qui servira de support à cet exposé. Ce cas d'étude est aujourd'hui bien connu et a été traité par de nombreux auteurs dont par exemple [6] & [7]. Nous avons également traité ce cas d'étude à l'aide de PyCATSHOO à travers un modèle unique qui intègre aussi bien son comportement discret que les équations de la physique qui régissent l'évolution déterministe de niveau du liquide et de la température dans le réservoir.

Dans la section 6 nous expliciterons l'approche mise en œuvre dans PyCATSHOO pour dialoguer avec un code physique dans le cadre d'une évaluation probabiliste et ce, conformément au standard FMI. Nous expliciterons dans la section 7 comment l'analyste fait usage d'un modèle externe qui encapsule les équations de la physique qui régissent le niveau et la température dans le réservoir et ce, sans faire appel explicitement aux fonctionnalités du standard FMI qui ont été intégrées à PyCATSHOO.

II. INTRODUCTION À PYCATSHOO

PyCATSHOO est fondé sur la cadre mathématique des Processus Markoviens Déterministes par Morceaux (PDMP) introduit par [8].

Fig. 1. Sauts spontanés ou forcés entre modes d'un PDMP

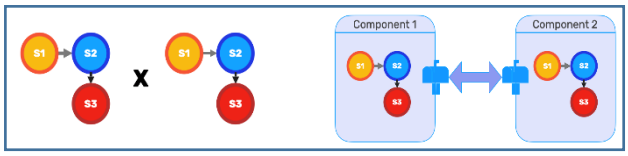


Le cadre mathématique des PDMP repose sur la décomposition du processus modélisé en une liste de modes de fonctionnement du système étudié. Il associe à chacun de ces modes un espace Ω_{mi} dans lequel les variables d'état réelles et continues évoluent de manière déterministe selon des équations de la physique. Ces équations peuvent être différentielles ordinaires, différentielles algébriques ou des équation explicites. L'évolution des variables d'état peut être interrompue de deux manières. La première est stochastique et pourrait correspondre à une défaillance en fonctionnement régie par un taux d'occurrence fixe ou variable. Cette interruption provoque un saut spontané vers un autre mode de fonctionnement. Le deuxième type d'interruption correspond à l'atteinte par les variables d'état de la frontière $\partial\Omega_{mi}$ de l'espace dans lequel elles sont supposées évoluer. Ce deuxième type d'interruption provoque un saut forcé vers un autre mode de fonctionnement. Le saut forcé pourrait correspondre à une atteinte d'une consigne qui provoque l'activation d'un actionneur sujet ou pas à des défaillances à la sollicitation.

Pour des systèmes simples, les modes de fonctionnement pourraient naturellement être considérés comme des états. La figure 1 pourrait ainsi correspondre à un automate stochastique car les transitions entre ses états discrets également stochastiques. Cet automate prend également le caractère hybride car, tant qu'un état discret est actif, les variables continues évoluent de manière déterministe.

Cette transposition ne peut cependant pas être appliquée telle quelle à un système complexe qui comprend une multitude de composants en interaction. En effet, un mode de fonctionnement du système est dans ce cas une combinaison des états des composants élémentaires. La représentation en automate globale risque d'aboutir à une explosion combinatoire difficile à aborder pour l'analyste et à simuler par les moyens de calcul actuels. La solution proposée par PyCATSHOO consiste en l'introduction de la notion d'automates stochastiques hybride distribués (ASHD) au sein d'objets autonomes et communicants comme illustré dans la figure 2. La description formelle de cette approche est publiée dans [9].

Fig. 2. La distribution alternative à la composition d'états



Ainsi, pour résumer, la modélisation d'un système avec PyCATSHOO débute en général par la production de modèles des objets qui le composent. La granularité de cette décomposition dépendra évidemment des objectifs de l'étude et de la nature des données disponibles. Comme illustré dans la figure 3, un objet est un acteur autonome communiquant. Il sera modélisé par les éléments suivants :

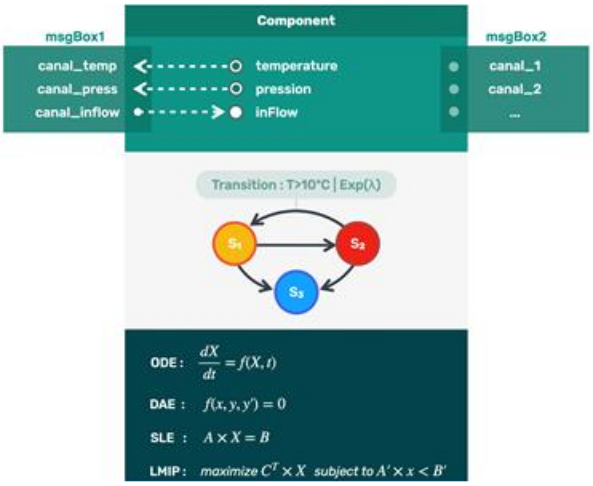
- Des variables d'états continues ou discrètes dont certaines peuvent être exposées au système extérieur à l'objet. Ces variables sont dites « intrinsèques » dans le vocabulaire PyCATSHOO.
- Des variables conteneurs d'information provenant du système extérieur à l'objet. Ces variables sont dites de « références » dans le vocabulaire de PyCATSHOO.
- Des boites à messages avec des canaux sortants qui exposent à l'extérieur les valeurs des variables intrinsèques de l'objet et des canaux entrants qui alimentent les variables références par des informations venant de l'extérieur de l'objet.
- Des automates stochastiques hybrides constitués d'état et de transitions. Ces dernières peuvent être déterministes instantanées, déterministes retardées, stochastiques ou instantanées selon des distributions de probabilités discrètes. Les transitions peuvent également être conditionnées par des expressions booléennes qui mettent en jeu aussi bien des variables intrinsèques que des variables références.
- Des équations explicites ou différentielles ordinaires ou algébriques qui régissent l'évolution des variables d'états continues. Ces équations peuvent être intégrées dans des systèmes globaux qui assurent la résolution simultanée des différentes équations déclarées dans les différents objets.
- Des contraintes linéaires sur certaines des variables de l'objet ainsi qu'une fonction objectif à maximiser par programmation linéaire entière et mixte. Ces équations peuvent également être intégrées dans des systèmes globaux qui assurent la résolution simultanée des différentes équations déclarées dans les différents objets.

L'ensemble des informations citées plus haut sont à formuler de manière déclarative. Les équations différentielles ordinaires sont par exemple déclarées en donnant des méthodes qui calculent les dérivées des variables. La résolution de ces équations de manière synchronisée avec le reste de la dynamique du système est prise en charge par PyCATSHOO.

De même, les transitions sont déclarées en spécifiant leurs états de départ et d'arrivé, leurs conditions sous forme de méthodes ayant un retour booléen et en désignant leur loi de probabilité.

Contrairement aux approches exclusivement dédiées à la modélisation et la simulation 0D/1D, PyCATSHOO n'exige du développeur de modèles aucune manipulation explicite de calcul probabiliste de type inversion de fonction de réparation ou autre.

Fig. 3. Représentation schématique d'un objet PyCATSHOO



Un complément procédural (Méthodes de réaction) peut être ajouté à ces informations. Il est à associer à des événements discrets. Ceci permet, par exemple, de propager les conséquences du changement d'état d'un automate.

En termes informatiques, les modèles des objets réalisés se concrétisent sous forme de classes écrites au choix en langage Python ou C++ et pourraient être automatiquement générées par un langage de modélisation graphique. L'ensemble des classes constitue ce que l'on appelle une base de connaissances. Une base de connaissances peut être dédiée à une catégorie particulière de systèmes (Réseaux, Télécommunication, thermo hydraulique, etc.). Elle peut aussi implémenter un formalisme générique comme celui des diagrammes de fiabilité.

Une base de connaissance peut ensuite être utilisée pour modéliser un système particulier. Ceci se fait en créant des objets par simple instanciation des classes de cette bases de connaissance. L'architecture du système sera ensuite représentée par l'établissement de liens entre les boites de messages de ces objets.

III. COUPLAGE AVEC DES MODELES PHYSIQUES DANS LES ETUDES PROBABILISTES

Dans le panorama des approches dynamiques d'évaluation probabiliste qui font appel à des modèles numériques de la physique que nous avons cité plus haut, on trouve principalement des approches qui étendent le formalisme des arbres d'événements. Les plus utilisées parmi ces approches introduisent des variantes qui se distinguent par la manière de déterminer les points de branchement et par la méthode de calcul de la probabilité de l'événement indésirable. On peut par exemple distinguer les variantes suivantes :

- Les points de branchement correspondent à des intervalles fixés par l'analyste. Les probabilités de branchement se déduisent de courbes de fragilité en utilisant des valeurs des variables d'état continues calculées par le modèle numérique de la physique.

- Les point de branchement ne correspondent qu'à des atteintes de seuil (Exemple point de consigne devant mener à une action du système technique ou humain). L'instant de branchement est obtenu via le modèle physique en résolvant un problème de franchissement.
- Le branchement intervient à un instant tiré de manière aléatoire à l'aide des taux de défaillance et de réparation.
- La probabilité de l'événement indésirable est obtenue en sommant les probabilités des séquences qui mènent à cet événement. Ces séquences sont obtenues par une exploration systématique des points de branchements protégée contre l'explosion combinatoire par un seuil de troncature minimale sur la probabilité de la séquence.
- La probabilité de l'événement indésirable est calculée par simulation Monte Carlo comme c'est le cas dans les arbres d'événements dynamiques de Monte Carlo (Monte Carlo Dynamic Event Tree).

A son tour, PyCATSHOO, peut produire des pilotes qui implémentent ces fonctionnalités. Il offre également la possibilité d'une génération à la volée des points de branchement qui peuvent indifféremment être dus aux comportements déterministes reproduits par le modèle physique, ou stochastiques pilotés par des lois de probabilité qui peuvent dépendre des variables d'état du modèle physique.

L'ensemble de ces approches ont en commun le traitement de deux problématiques concernant le couplage avec les modèles numériques de la physique :

- La première problématique concerne la sauvegarde de l'état du système à un instant d'intérêt donné. Cette sauvegarde contribue à l'efficacité des calculs. Il peut arriver en effet que des états de branchements soient visités dans les mêmes conditions et à plusieurs reprises par le solveur. Il est intéressant dans ce cas de ne pas avoir un refaire un calcul déjà réalisé.
- La deuxième problématique fait largement appel à la première. Elle concerne la détection de franchissement de seuils par les variables d'état d'un système. Ce franchissement peut en effet faire appel à une dichotomie qui nécessite un retour arrière pour reprendre la simulation avec un pas plus petit que le précédent.

Ces deux problématiques sont parmi les plus saillantes lorsqu'il s'agit, dans une évaluation probabiliste, de réaliser un couplage avec un modèle numérique de la physique du système étudié. Ce couplage est susceptible d'occasionner un travail conséquent qui est souvent très dépendant du modèle numérique utilisé. Et, en cas de changement de ce dernier, le couplage doit être repris au moins partiellement.

IV. STANDARD FMI ET COSIMULATION

Dans certains pilotes de couplage comme RAVEN [11], une interface de programmation propriétaire (API) est proposée pour faciliter le couplage avec les codes physiques et répondre aux deux problématiques décrites plus haut. Le couplage est alors réalisé en donnant un codage de cette API dédié au code physique utilisé. Ceci assurera la communication entre ce dernier et le pilote du couplage.

Dans une telle situation le changement du pilote comme le changement du code physique nécessitera un nouveau codage de l'API.

Notre objectif a donc été de munir PyCATSHOO d'une API similaire. Nous avons basé cette API sur le standard FMI dans le but de minimiser les efforts de couplage.

Avant d'illustrer l'usage du FMI par PyCATSHOO, nous donnerons dans ce qui suit, les principes clés de ce standard que nous avons extrait du document [12] des spécifications de la version 2.02.

Le standard FMI (Functional Mock-up Interface) a été développé par le projet européen ITEA2 MODELISAR de 2008 à 2011[1]. Il évolue depuis 2011 dans le cadre d'un projet permanent de l'Association Modelica. Le standard FMI se présente sous forme d'une spécification accompagnée d'une interface de programmation (API) qui doit être implémentée par un exécutable appelé FMU (Functional Mock-up Unit). Ce dernier jouera le rôle de tout ou partie du modèle numérique de la physique à utiliser dans une EPS dynamique. Un modèle numérique pourra ainsi être nativement développé comme une unité FMU c-à-d selon spécifications FMI ou disposer d'une enveloppe logicielle qui implémente les fonctionnalités standards d'une FMU. Cette enveloppe se charge dans ce cas de faire le lien avec les fonctionnalités du modèle numérique. Dans les deux cas, la structure interne des données d'une FMU et ses algorithmes ne sont pas exposés et peuvent évoluer sans remettre en cause leur utilisabilité. En effet, une unité FMU est faite pour être utilisée par un environnement de simulation maître qui peut piloter plusieurs FMU grâce à l'API du standard FMI.

Selon le standard FMI, les FMU peuvent être de deux types :

- Les FMU pour échange de modèles qui ne disposent pas de solveurs et qui délèguent au pilote de la simulation le contrôle complet de la simulation et de la résolution équations du modèle. Ils présentent deux inconvénients :
  a. Ces FMU ne sont pas autonomes puisqu'ils n'intègrent pas de solveur ; ils ne peuvent donc pas être validés indépendamment du pilote,
  b. Le pilote est complexe à construire puisqu'il regroupe toute la résolution des équations.
- Les FMU pour cosimulation qui disposent de leurs propres solveurs et leurs propres pas d'intégration. C'est ce type de FMU que nous décrirons et que nous avons mis en œuvre dans PyCATSHOO.

V. TRAITEMENT DU CAS D'ÉTUDE HEATEDTANK PAR PYCATSHOO

A. *Traitement du cas d'étude HeatedTank par PyCATSHOO*

Le système évalué dans le cas d'étude HeatedTank représente un réservoir qui comprend les éléments suivants :

[1] https://itea4.org/publication/download/910-modelisar-seizing-the-high-ground.pdf

- Un réservoir (Tank) pouvant contenir un liquide.
- Une source (Heater) de puissance thermique constante plongée dans le liquide contenu dans le réservoir.
- Deux pompes (Pump) qui fonctionnent en tout ou rien qui, lorsqu'elles sont ouvertes, apportent au réservoir un liquide frais avec un débit nominal constant afin de refroidir le réservoir et éviter son dessèchement.
- Une vanne (Valve) d'évacuation qui permet d'évacuer l'eau chaude et d'éviter le débordement du réservoir.

Les pompes sont asservies au niveau du liquide $L_{tank}(t)$ dans le réservoir. Elles s'ouvrent lorsque $L_{tank}(t) < L_{min}$ et se ferment lorsque $L_{tank}(t) > L_{max}$.

La vanne est également asservie au niveau du liquide dans le réservoir. Elle s'ouvre lorsque $L_{tank}(t) > L_{max}$ et se ferme lorsque $L_{tank}(t) < L_{min}$.

Les deux pompes et la vanne sont sujettes à deux modes de défaillances : « bloqué ouvert » et « bloqué fermé ». Le taux d'occurrence de ces défaillances est fonction de la température $T_{tank}(t)$ dans le réservoir et évolue selon une équation de la forme suivante :

$$\lambda(t) = \lambda_0[b_1 e^{b_c(T_{tank}(t)-20)} + b_2 e^{-b_d(T_{tank}(t)-20)}] \quad (1)$$

La probabilité de transition avant un instant t vers l'un des deux modes de défaillances est donnée par l'équation suivante :

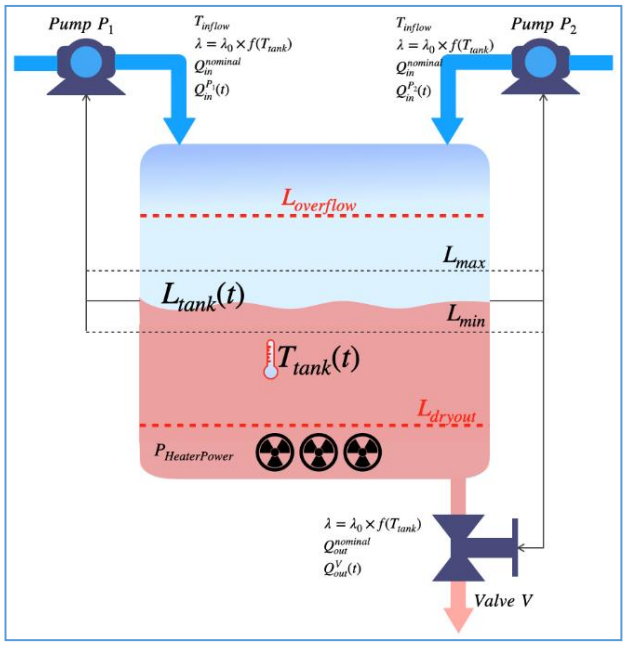
$$P(T < t) = 1 - e^{-\int_0^t \lambda(u)du} \quad (2)$$

Cette équation permet à PyCATSHOO de tirer les instants de saut en se basant uniquement sur une méthode qui implémente l'expression donnée par l'équation (1).

Aucune réparation n'est envisagée dans la version initiale de ce cas d'étude[2].

L'objectif de ce cas d'étude est d'évaluer l'évolution au cours du temps des probabilités pour le niveau dans le réservoir de sortir des limites minimale L_{dryout} et maximale $L_{overflow}$ et pour la température du réservoir d'aller au-delà de la limite $T_{burning}$.

Fig. 4. Schéma de principe du HeatedTank



B. *Modélisation intégrée avec PyCATSHOO*

Comme indiqué en section 2, la première étape de modélisation consistera à produire un modèle de comportement pour chacun des composants Tank, Pump, Valve et Heater. Ces modèles seront implémentés dans 4 classes différentes. Le code source de cette modélisation est donné en Python sur le site pycatshoo.org. Nous ne décrirons ici que les concepts principaux et ceux qui devront être modifiés lors de l'usage d'un FMU, à savoir quelques extraits pour les classes Pump et Tank.

1) Extraits du modèle de la classe Pump

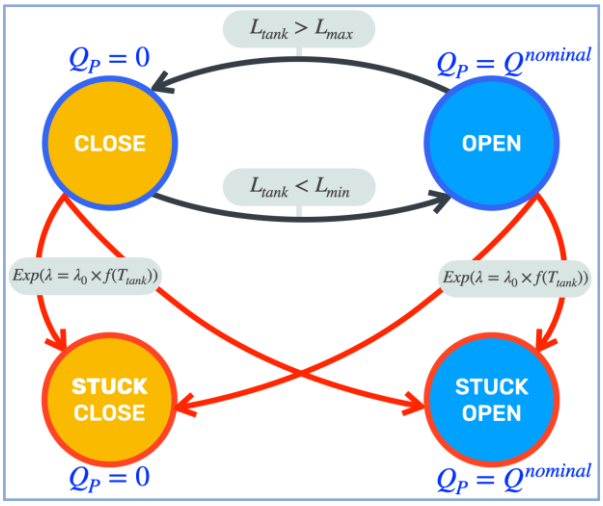
La déclaration des variables d'état et des paramètres se fait de la manière suivante :

```
self.p_nominalFlow       =
  self.addVariable("nominalFlow",
  Pyc.TVarType.t_double, 1.5)

self.v_flow              =
  self.addVariable("flow"       ,
  Pyc.TVarType.t_double, 1.5)

self.p_lambda0           =
  self.addVariable("lambda0"    ,
  Pyc.TVarType.t_double, 0.001)

self.v_lambda            =
  self.addVariable("lambda"     ,
  Pyc.TVarType.t_double, 0.001)
```

[2] Nous avons ajouté des réparations à une autre version du même cas d'étude dans le cadre d'un benchmark sur la simulation de modèles stochastiques que nous avons décrit dans [13].

La classe Pump aura un comportement qui correspond à l'automate de la figure 5.

Fig. 5. Automate de la classe Pump



Le code correspondant à la création de cet automate est le suivant :

```
self.a_automaton       =
  self.addAutomaton("FuncAutomation")

self.s_openState       =
  self.addState("FuncAutomation", "OPEN"      , 0)

self.s_closeState      =*
  self.addState("FuncAutomation", "CLOSE"     , 1)

self.s_stuckOpenState  =
  self.addState("FuncAutomation", "STUCKOPEN" , 2)

self.stuckCloseState   =
  self.addState("FuncAutomation", "STUCKCLOSE", 3)

self.a_automaton.setInitState(self.s_openState)

self.a_automaton.addSensitiveMethod
  ("updateFlow", self.updateFlow)
```

Cet extrait désigne également l'état initial par défaut ainsi que le nom de la méthode de réaction qui modifie la valeur du flux en sortie de la pompe à chaque changement d'état de l'automate.

L'extrait suivant indique la manière de créer les transitions :

```
trans_o2c  =
  self.s_openState.addTransition("OPEN_to_CLOSE")

trans_o2c.setCondition("closeCondition",
  self.closeCondition)

trans_o2c.addTarget(self.s_closeState,
  Pyc.TTransType.trans)
……
trans_o2so =
  self.s_openState.addTransition
  ("OPEN_to_STUCKOPEN")

trans_o2so.setDistLaw(Pyc.IDistLaw.newLaw(self,
  Pyc.TLawType.expo, self.v_lambda))

trans_o2so.addTarget(self.s_stuckOpenState,
  Pyc.TTransType.fault)

trans_o2so.setModifiable
  (Pyc.TModificationMode.continuous_modification)
```

La première transition donnée par cet extrait est déterministe. Sa condition est donnée par la fonction booléenne ***closeCondition***. La deuxième transition est stochastique. Elle obéit à une loi exponentielle dont l'intensité λ dépend de la température dans le réservoir. La variable λ est donc une variable continue qui sera prise en charge par un gestionnaire de processus markovien déterministe par morceaux (PDMP) déclaré comme suit :

```
self.addPDMPManager            ("pdmpManager")

self.addPDMPExplicitVariable
  ("pdmpManager", self.v_lambda)

self.addPDMPEquationMethod
  ("pdmpManager","odeMethod", self.odeMethod)

self.addPDMPWatchedTransition
  ("pdmpManager", trans_o2c)
```

Dans cet extrait, on précise au PDMP qu'il devra gérer la variable continue lambda et que celle-ci devra évoluer selon l'équation explicite (1) implémentée dans la méthode ***odeMethod***. On précise également au PDMP qu'il devra surveiller la transian ***trans_o2c*** qui correspond au passage de l'état ouvert à l'état fermé car ce passage correspond au franchissement d'un seuil qui représente est une condition d'arrêt de l'intégration. Cet arrêt permettra à PyCATSHOO de provoquer un changement de mode du PDMP.

Contrairement à des outils ou langages uniquement dédiés à la modélisation et la simulation 0D/1D comme Modelica, ces déclarations suffisent à PyCATSHOO pour prendre en charge l'intégration et les tirages des instants de transition de manière coordonnée avec le reste du comportement modélisé.

2) Extraits du modèle de la classe Tank

Les deux principales variables d'état continues du réservoir sont la température et le niveau du liquide. Elles sont déclarées dans la classe Tank de la manière suivante :

```
self.v_temperature =
  self.addVariable("temperature",
  Pyc.TVarType.t_double, 20.)

self.v_level       =
  self.addVariable("level"       ,
  Pyc.TVarType.t_double, 7.)
```

Ces variables devront être gérées par le même PDMP que les autres composants du système. Elles seront donc attachées à ce PDMP mais cette fois-ci via un système d'équations différentielles ordinaires.

```
self.addPDMPEquationMethod
  ("pdmpManager", "odeMethod", self.odeMethod)

self.addPDMPODEVariable
  ("pdmpManager", self.v_level)

self.addPDMPODEVariable
  ("pdmpManager", self.v_temperature)
```

La méthode ***odeMethod*** devra dans ce cas donner les expressions des dérivées des deux variables. Ceci est fait comme dans l'extrait suivant :

```
def odeMethod(self):
    iFlow    = self.r_inFlow.sumValue()
    oFlow    = self.r_outFlow.sumValue()

    self.v_level.setDvdtODE((iFlow - oFlow) /
                  self.p_area.dValue())

    sumiFiT = 0
    for i in range(self.r_inFlow.nbCnx()):
        sumiFiT = sumiFiT +
                  self.r_inFlow.dValue(i) * \
                  (self.r_temperature.dValue(i) -
                  self.v_temperature.value())

    self.v_temperature.setDvdtODE(
        (sumiFiT + self.r_power.dValue(0)) /
                  (self.v_level.dValue() *
                  self.p_area.dValue()))
```

A partir de ces déclarations, PyCATSHOO met en œuvre un de ses schémas d'intégration pour suivre l'évolution dans le temps de l'ensemble des variables d'état du système en coordinations avec les événements discrets qui peuvent se produire de manière déterministe ou aléatoire.

VI. PRINCIPES DU DIALOGUE ENTRE PYCATSHOO ET UNE FMU

Actuellement, PyCATSHOO ne met en œuvre que le protocole de cosimulation du standard FMI.

PyCATSHOO peut servir à développer des FMU de cosimulation et dispose pour cela de fonctionnalités d'aide à la génération quasi automatique de descripteur de FMUs et du codage de l'API FMI. Mais nous ne décrirons pas ici ces fonctionnalités. Nous allons nous concentrer uniquement sur le rôle de l'outil en tant que pilote de FMUs en mode cosimulation.

Dans une approche simpliste de la cosimulation, l'utilisation d'une telle FMU peut consister à en effectuer une simulation en s'arrêtant à des instants prédéfinis pour monitorer les valeurs de certaines variables du modèle physique. Ce mode de fonctionnement est possible avec PyCATSHOO. Il a été expérimenté sur le modèle original intégré du Heated Tank que nous avons décrit dans la section 5. Dans cette expérimentation, un pilote externe se contente de récupérer de la FMU des valeurs de variables d'intérêt à des instants précis. Cette approche peut être utile lorsque les outils de modélisation utilisés pour réaliser les FMU ne permettent pas un fonctionnement efficace dans un cadre de simulation Monte Carlo en raison, par exemple, de lenteurs des calculs ou d'impossibilité de paralléliser les simulations afin de les distribuer sur les nœuds d'un cluster par exemple.

Dans une approche plus complexe, il est possible de faire en sorte que le pilote tienne compte d'événements pouvant se présenter dans la FMU elle-même. La FMU est alors censée stopper sa simulation lorsque l'événement survient avant la fin du pas de simulation qui lui a été demandé. Cette approche décrite dans [14] nécessite que le pilote réagisse aux événements présentés par la FMU. Elle présente plusieurs contraintes :

- Les événements modélisés dans la FMU doivent correspondre à ceux attendus par le pilote,
- Le point d'arrêt de la FMU doit avoir les propriétés attendues par le pilote en termes de précision, de dépassement ou de non-dépassement,
- Dans le cas d'une gestion parallèle de plusieurs FMU, il est nécessaire que ceux-ci puissent gérer des reculs de leur simulation pour que le pilote puisse les synchroniser au niveau du premier instant d'arrêt d'une FMU.

L'approche que nous avons adoptée ici est différente. Elle fait en sorte que la FMU ne soit pas conçue spécifiquement pour le pilote. La détection des événements que le pilote doit gérer est entièrement effectuée par le pilote lui-même. La FMU peut ainsi être un simple modèle de simulation dont on ne peut anticiper l'état futur. La détection des franchissements de seuils ne peut dans ce contexte être effectuée qu'en dépassant le seuil et en effectuant une recherche de l'instant de franchissement en repartant du pas précédent. Ainsi, la seule contrainte que cette approche impose à la FMU c'est qu'elle puisse reculer sa simulation au pas précédent dont il doit avoir sauvegardé l'état.

L'intégration dans PyCATSHOO d'outils lui permettant d'utiliser des FMUs de cosimulation a été réalisée avec l'objectif de minimiser l'effort à fournir par le concepteur du pilote. La FMU est chargée par PyCATSHOO grâce à une commande simple ***addFMU(cheminDuFMU)*** qui effectue l'ensemble des opérations et des contrôles de chargement et place la FMU en situation d'initialisation. La FMU chargée se présente alors sous la forme d'un composant spécifique d'une forme voisine de celle des composants PyCATSHOO standard et présente une liste de variables de forme identique aux variables PyCATSHOO standard.

Le concepteur du pilote peut alors utiliser tous les outils de PyCATSHOO pour modifier les valeurs initiales des variables, ajouter des méthodes de réaction à la modification de variables discrètes, demander des calculs d'indicateurs, demander le monitoring, etc.

La simulation d'une séquence complète par un modèle PyCATSHOO qui utilise des FMU reste également transparente pour le concepteur du pilote.

Dans le cas où le pilote ne doit pas gérer les franchissements des seuils de variables issues des FMU, cette simulation comprend les étapes suivantes :

- Démarrage de la séquence : les FMU sont sortis du mode d'initialisation,
- Pour chaque intervalle de temps avant le prochain instant de transition déterminé par PyCATSHOO comme en f :
  a. Intégration des équations différentielles jusqu'à l'instant souhaité ou l'instant d'un franchissement de seuil sur les variables qui sont à la charge du solveur du pilote,
  b. Avancement des FMU jusqu'à l'instant courant,
  c. Application des transitions actives,
  d. Gestion des méthodes de réaction,
  e. Contrôle des conditions d'arrêt de la séquence,
  f. Calcul de la prochaine date de transition,
  g. Reprise en a si les conditions d'arrêt de la séquence ne sont pas remplies,
- Fin de la séquence : les FMU sont réinitialisés, replacés en mode d'initialisation et les valeurs initiales des variables sont réappliquées.

Dans le cas où le pilote doit gérer des franchissements de seuils mettant en jeu les variables de la FMU, cette dernière doit avoir été ajoutée au solveur courant du pilote. Ce dernier se charge de détecter la réalisation de la condition de franchissement puis de la recherche du point de franchissement. Cette recherche se fait par dichotomie en demandant à la FMU en charge de la variable concernée par le franchissement de reprendre la simulation à partir du l'instant précédent et avec un pas de calcul plus petit. L'arrêt de la simulation intervient immédiatement après le franchissement comme attendu par PyCATSHOO après synchronisation de l'ensemble des FMU et, éventuellement des résolutions prises en charge par le pilote lui-même.

VII. HEATEDTANK : UNE PREMIERE MISE EN ŒUVRE DU FMI DANS PYCATSHOO

A. *Détails de mise en œuvre*

Dans l'illustration qui suit nous avons fait en sorte que la résolution des équations différentielles qui régissent l'évolution des variables d'état du réservoir soit prise en charge par une FMU externe qui ait ainsi le comportement de la classe Tank. Le reste de la modélisation comprend le pilotage des événements discrets déterministes est stochastiques. Il restera au niveau du pilote qui continue à héberger les classes Pump, Valve, et Heater. Le seul changement sera au niveau de la classe Tank qui ne sera plus qu'un proxy de la FMU au sein du pilote. Nous donnerons donc dans ce qui suit les modifications apportées à la classe Tank du pilote.

- ***Modification des déclarations des variables :***

  Les variables Température et niveau dans le réservoir seront ici attachées à celles de la FMU qu'il faudra avoir chargé. Ceci est illustré dans l'extrait suivant :

```
self.fmu           =
  self.system().addFMU
     ("fmufolder/Tank.fmu", "MyFMU")

self.v_temperature =
  self.fmu.variable
     ("FMUTank.Temperature")

self.v_level       =
  self.fmu.variable("FMUTank.Level")
```

  Les variables d'entrée discrètes de la FMU devront également être déclarées comme dans l'exemple suivant :

```
self.v_power       =
  self.fmu.variable('FMUTank.Power')
```

- ***Modification du PDMP du pilote***

  Le PDMP local devra être informé de la prise en charge d'une FMU externe.

```
pdmp.addFMU( self.system().FMU("MyFMU"))
```

  Le PDMP devra également disposer d'une méthode qui sera automatiquement appelée par PyCATSHOO au début de chaque séquence déterministe. Elle servira à transmettre aux différentes FMU les modifications des variables discrètes qu'elles utilisent.

```
self.addPDMPBeginMethod
  ("pdmpManager",
   "beginMethod",
    self.beginMethod)
```

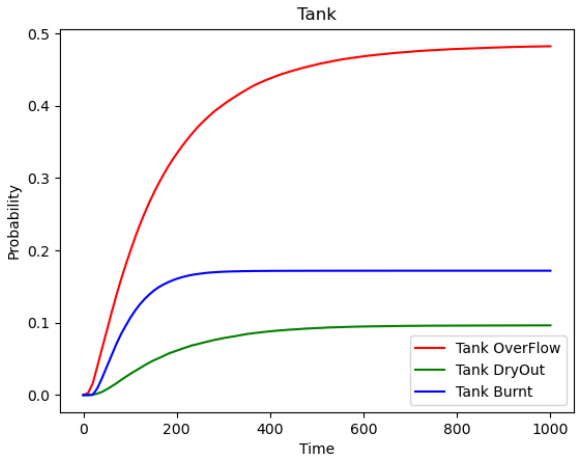
La méthode de transmission des valeurs discrètes sera implémentée comme dans l'extrait suivant :

```
def beginMethod(self):
    ......

     self.v_power.setDValue
         (self.r_power.sumValue())
```

B. *7.2 Résultats de l'illustration*

Cette illustration nous a permis d'obtenir les mêmes résultats que ceux obtenus par une modélisation PyCATSHOO monolithique sans usage de FMU. Les temps de calcul n'ont cependant pas été identiques. L'usage des FMU ralentit les simulations. Ce ralentissement est principalement dû aux actions de sauvegarde de l'état demandées par le pilote. Les FMU produits par PyCATSHOO ont une manière de procéder à cette sauvegarde qui reste à optimiser même si la réalisation de FMU n'est pas l'objectif principal de PyCATSHOO dans un processus de couplage. PyCATSHOO se positionne en effet sur le rôle de pilote qui apporte les comportements stochastiques discrets et multimodaux à des codes physiques déterministes existants.

Fig. 6. Résultats identiques entre simulation monolithique et simulation avec FMU



Le tableau 1 donne les temps de calcul en secondes pour différents nombres de séquences simulées en modélisation monolithique et en modélisation avec FMU. Ces simulations ont été réalisées avec une version du modèle en C++ compilé. Elles ont été distribuées sur 10 files d'exécutions sur un PC portable avec un processeur Intel(R) Core (TM) i7-8750H de six cœurs et 32 Go de mémoire.

TABLE I. TEMPS DE CALCUL EN FONCTION DU NOMBRE DE SEQUENCE SIMULEES

	Nombre de séquences simulées			
	10^3	**10^4**	**10^5**	**10^6**
Modélisation avec FMU	1.7 s	7.7 s	69 s	727 s
Modélisation monolithique	1.2 s	2.2 s	16.4 s	180 s

Il est à noter que dans les deux cas, le temps de calcul progresse de manière pratiquement linéaire en fonction du nombre des séquences simulées.

VIII. CONCLUSION

Cette étude est une illustration des capacités d'utilisation des FMU par PyCATSHOO. Elle a mis en évidence une manière transparente, pour le concepteur de pilotes de coordonner l'exécution de FMUs sans faire d'appels directs à l'API du standard FMI.

L'approche que nous avons adoptée pour mettre en œuvre le standard FMI dans PyCATSHOO fait que les pilotes restent en charge de la conduite des événements qui peuvent correspondre aux instants de branchement et ce sans contrainte quant à la nature de ces événements et donc quant à la nature des branchements que le pilote prend en charge. La mise en place de l'enveloppe FMU, selon cette approche n'implique pas de modifications sur le code du modèle numérique utilisé autres que celles qui répondent à la nécessité de sauvegarder, à la demande, l'état du sous-système géré par la FMU à un pas de calcul donné et à la nécessité de disposer d'une capacité de reprise de calcul à un pas antérieur. L'augmentation du temps de calcul constatée par rapport à une modélisation monolithique est principalement due aux procédures de sauvegarde de l'état à différents pas de calcul. Elle n'est cependant pas systématique et dépend de l'efficacité du code de calcul à réaliser ces procédures. Dans la version actuelle de PyCATSHOO, ces procédures restent à optimiser. Mais ceci n'est pas le but principal de PyCATSHOO qui se positionne plus sur le rôle de pilote qui apporte le comportement stochastique discret et multimodal à des codes physiques déterministes existants.

L'augmentation du temps de calcul, quand elle se confirme, reste cependant acceptable pour plusieurs raisons :

- Elle évolue de manière linéaire avec le nombre de séquences simulées.
- Elle n'empêche pas l'usage de cluster pour paralléliser les simulations.
- Elle est compensée par le gain en la modularité et par la capacité d'interchangeabilité des codes numériques des modèles physiques utilisés.

Le système que nous avons étudié dans cet article est assez représentatif quant aux fonctionnalités que l'intégration du standard FMI doit offrir. Par conséquent, les résultats de l'illustration que nous avons présenté dans ce papier montrent que le recours au FMI pourrait faciliter grandement les taches de couplage dans les études probabilistes dynamiques qui font appel à des modèles hybrides. Nos perspectives restent cependant de confronter notre travail à des études de taille industrielle. Il nous reste également à évaluer l'intérêt de la deuxième façon de mettre en œuvre le standard FMI, à savoir la méthode de l'échange de modèles.

REFERENCES

[1] T. Aldemir. "A survey of dynamic methodologies for probabilistic safety assessment of nuclear power plants", Annals of Nuclear Energy, 52, pp.113-124, (2013).

[2] B. Tombuyses and T. Aldemir. "Continuous Cell-To-Cell Mapping". Journal of Sound and Vibration, 202(3), pp395-415, (1997).

[3] P. E. Labeau and J. M. Izquierdo. "Modeling PSA Problems—I: The Stimulus-Driven Theory of Probabilistic Dynamics", NUCLEAR SCIENCE AND ENGINEERING, 150, pp 115–139, (2005).

[4] H. Chraibi, J.C. Houdebine, A. Sibler."PyCATSHOO: Toward a new platform dedicated to dynamic reliability assessments of hybrid systems", PSAM13 (2016).

[5] N. Brînzei, C. Duval, H. Chraibi, M. Hassanaly. "Modeling the Consequences of Feared Event by Stochastic Hybrid Automata", Proceedings of the 30th European Safety and Reliability Conference and the 15th Probabilistic Safety Assessment and Management Conference, pp 4875-4882, (2020).

[6] Zhang H., De Saporta B., Dufour F., Deleuze G., "Dynamic Reliability by Using Simulink and Stateflow", Chemical Engineering Transactions, 33, 529-534, (2013).

[7] M. Bouissou, X. de Bossoreille, "From Modelica models to dependability analysis", IFAC PapersOnLine, Volume 48, Issue 7, Pages 37-43, (2015).

[8] M. H. A. Davis. "Piecewise-Deterministic Markov Processes: A general class of non-diffusion stochastic models", J. R. Stat. Soc. B 46:353-388, (1984).

[9] L. Desgeorges, P.Y. Piriou, T. Lemattre, H. Chraibi. "Formalism and semantics of PyCATSHOO: A simulator of distributed stochastic hybrid automata". RESS, Volume 208, (April 2021), 107384.

[10] C. Picoco, T. Aldemir, V. Rychkov, A. Alfonsi, D. Mandelli, C. Rabiti. "Coupling of RAVEN and MAAP5 for the Dynamic Event Tree analysis of Nuclear Power Plants". Proceedings of the 27th European Safety and Reliability Conference, June 18-22, PORTOROZ SLOVENIA (2017).

[11] C. Rabiti, A. Alfonsi, J. Cogliati, D. Mandelli, R. Kinoshita, S. Sen, C. Wang, P. W. Talbot, D. P. Maljovec, M. G. Abdo, "RAVEN User Manual", INL/EXT-15-34123 (2022)

[12] Modelica Association. "Functional Mock-up Interface for Model Exchange and Co-Simulation", Document version 2.02 (2020). https://github.com/modelica/fmi-standard/releases/download/v2.0.3/FMI-Specification-2.0.3.pdf.

[13] A. Abate, H.A.P Blom, M. Bouissou, N. Cauchi, H. Chraibi, J. Delicaris, S. Haesaert, A. Hartmanns, H. Ma and More Authors. "ARCH-COMP21 Category Report: Stochastic Models". In G. Frehse, & M. Althoff (Eds.), 8th International Workshop on Applied Verication of Continuous and Hybrid Systems, EPiC Series in Computing, 80, pp. 55-89, (2021).

[14] F. Cremona, M. Lohstroh, D. Broman, M. Di Natale, E. A. Lee, and S. Tripakis. "Step revision in hybrid Co-simulation with FMI", conference paper (2016).